

COMPUTATIONAL APPROACHES TO *D. RERIO* RETINAL ORGANOGENESIS

by

Michael Mattocks

A thesis submitted in conformity with the requirements  
for the degree of Doctor of Philosophy  
Graduate Department of Cell and Systems Biology  
University of Toronto

# Abstract

Computational Approaches to *D. rerio* Retinal Organogenesis

Michael Mattocks

Doctor of Philosophy

Graduate Department of Cell and Systems Biology

University of Toronto

2021

Increasing integration of statistical and computational techniques into the analysis of cell and molecular biological data has created opportunities for explaining organogenic phenomena by numerical analysis. This thesis first examines the suitability of the most developed of these explanations for the development of the *D. rerio* eye, the stochastic mitotic mode explanation (SMME) for retinal progenitor cell (RPC) function, and demonstrates that it is neither the best available explanation, nor theoretically sound. It is shown that a deterministic alternative model is an improved explanation for the data, and that the SMME cannot explain postembryonic RPC function. These studies support earlier hypotheses of a linear progression of competency phases over the hypothesized “stochastic” nature of RPC lineage commitment.

A Bayesian model comparison framework, relying centrally on the Galilean Monte Carlo nested sampling algorithm for estimation of model evidence and posterior parameter distributions, is subsequently elaborated as an improved method for testing models in organogenesis. The utility of this approach is proved by comparing models of *D. rerio* RPC populations in the postembryonic Circumferential Marginal Zone (CMZ). The inferential framework is used to show that the activity of CMZ RPCs in the postembryonic period is best characterised by a decaying cell cycle rate, with similar activity across morphological axes, despite asymmetrical population dynamics. Changes to RPC lineage commitment over this time suggest an explanation for the changing photoreceptor mosaic. These studies establish that CMZ RPC behaviours change over time, whose postembryonic function is not adequately described as a recapitulation of an embryonic program.

Finally, the *D. rerio* mutant *rys* is shown to arise from a lesion in the npat locus, and the apparently paradoxically enlarged CMZ RPC population observed in these animals is demonstrated to arise from a failure of these cells to specify and exit the proliferative niche, and not from a proliferative defect. Nested sampling is used to show that unique populations of nucleosome positions in *rys* mutants and sibling arise from separate causal processes. A mechanism for these observations involving simultaneous changes to the subunit composition of the nucleosome pool, and loss of translational control, is suggested.



This work is dedicated to the memory of my brother Gareth Akerman.

## Acknowledgements

The completion of this thesis was possible only due to three teachers: my supervisor Dr. Vince Tropepe, who supported it with beatific patience and generosity, my advisor Dr. Umar Faruq Abdullah, who taught me that my scientific problems were in fact metaphysical and gave me the courage to proceed, and my sifu Liz Parry, who taught me how to stand and breathe in a fight. It further required the longsuffering care of my lovely wife Wing. I am indebted in innumerable ways to members, past and present, of the Tropepe, Bruce, Tepass, and Godt labs, as well as to other 6th floor denizens and the broader CSB community. I am particularly indebted to Monica Dixon and Loksum Wong for their extensive teachings and careful scientific example. Henry Hong and Audrey Darabie helped in innumerable ways with the imaging work and were a consistent source of encouragement and good cheer. All of the data generated here relied on the contributions of many hands, not least of which are the many involved with the long hours in care and feeding of the fish. I have gratefully included experimental work entirely generated by Monica Dixon and Maria Augusta Sartori in [Chapter 5](#); the underlying mapping work was performed by Jason Willer of the University of Louisville. I thank my committee members, particularly Dr. Ulrich Tepass and Dr. Rod Bremner, for their time and helpful suggestions, throughout the project. The additional members of my examining committee, Dr. Gary Bader, Dr. Kevin Knuth, and Dr. Alan Moses, graciously offered their time in reading, evaluating, and improving this document. I am particularly grateful to Dr. Knuth for an exhaustive set of comments and suggestions. I am finally indebted to Dr. John Skilling, both for delivering me to a coherent statistical understanding in his published work, and for a lengthy email which enabled me to complete `GMC_NS.jl` and to use it extensively in the final analyses.

# Contents

<b>Introductory Notes</b>	<b>1</b>
<b>I Modelling Studies</b>	<b>2</b>
<b>1 Canonical retinal progenitor cell phenomena and their explanations</b>	<b>3</b>
1.1 The Stochastic Mitotic Mode Explanation (SMME) . . . . .	3
1.2 Explanations for RPC function in 2009 and the drive to unification . . . . .	4
1.3 Canonical vertebrate RPC phenomena: the RPC “morphogenetic alphabet” . . . . .	5
1.3.1 Proliferative phenomena . . . . .	6
1.3.2 Fate specification phenomena . . . . .	8
1.3.3 Other morphogenetic phenomena . . . . .	10
1.4 Macromolecular mechanistic explanations for RPC phenomena . . . . .	10
1.4.1 Transcription factor networks . . . . .	11
1.4.2 Intercellular signalling networks . . . . .	12
1.4.3 Patterning mechanisms . . . . .	13
1.4.4 Chromatin dynamics . . . . .	14
1.5 A unified theory of RPC function? “Blurring” to order . . . . .	14
1.6 Explanatory Strategy and Intent of the SMME . . . . .	15
<b>2 “Stochastic mitotic mode” models do not explain zebrafish retinal progenitor lineage outcomes</b>	<b>18</b>
2.1 Introduction . . . . .	18
2.2 Metatheoretical analysis . . . . .	20
2.2.1 Gomes SSM: Ancestral Model of the SMME SSMs . . . . .	21
2.2.2 He SSM: Explaining variability in zebrafish neural retina lineage size . . . . .	22
2.2.3 Boije SSM: Explaining variability in zebrafish RPC fate outcomes . . . . .	24
2.3 Model selection demonstrates the SMME is not the best available explanation for RPC lineage outcomes . . . . .	25
2.4 SMME SSMs cannot explain the post-embryonic phase of CMZ-driven zebrafish retinal formation . . . . .	29
2.5 Conclusion . . . . .	32

<b>3 Toward a computational CMZ model comparison framework</b>	<b>35</b>
3.1 SMME Postmortem: a wrong turn at Gomes . . . . .	35
3.2 Implications of the SMME study for modelling CMZ RPCs . . . . .	37
3.3 CMZ models in a putative model comparison framework . . . . .	39
3.3.1 Spatial dimension of the models: the "slice model" . . . . .	39
3.3.2 Temporal resolution of the models . . . . .	42
3.4 Structuring models under uncertainty . . . . .	42
<b>4 Bayesian characterisation of postembryonic CMZ activity</b>	<b>44</b>
4.1 Independent Log-Normal modelling of CMZ parameters . . . . .	44
4.2 Survey of CMZ population and gross retinal contribution . . . . .	45
4.3 Two-phase periodization of postembryonic CMZ activity by phased difference equation modelling . . . . .	48
4.4 Phased slice-models of CMZ population dynamics suggest subtle shifts across stable-population contour . . . . .	52
4.5 Slice models of decaying cell cycle support anatomical homogeneity of RPC behaviour . .	58
4.6 Bayesian inference of cycle parameters from cumulative thymidine labelling experiments supports slice model cycle time estimates . . . . .	60
4.7 The CMZ contributes stably to each cellular layer with time-variable lineage composition	62
4.8 Early retinal cohorts of the <i>D. rerio</i> retina are turned over at a low rate by 4C4-positive microglia . . . . .	68
4.9 Summary: Modelling the postembryonic CMZ . . . . .	70
4.10 Future directions . . . . .	70
<b>5 Mutant npat results in nucleosome positioning defects in <i>D. rerio</i> CMZ progenitors, blocking fate specification but not proliferation</b>	<b>72</b>
5.1 Introduction . . . . .	73
5.2 Results . . . . .	74
5.2.1 The <i>rys</i> CMZ phenotype is characterised failure of RPCs to specify, altered nuclear morphology, aberrant proliferation and expanded early progenitor identity . . . . .	74
5.2.2 The microphthalmic zebrafish line <i>rys</i> is an npat mutant . . . . .	83
5.2.3 <i>rys</i> siblings and mutants have unique sets of nucleosome positions, best explained by different sequence preferences and increased sequence-dependent positioning in mutants . . . . .	89
5.3 Discussion . . . . .	97
<b>6 Inferring and modelling specification dynamics in retinal progenitors</b>	<b>101</b>
6.1 Specification as a function of non-proliferative chromatin dynamics . . . . .	101
6.2 Future directions . . . . .	102
<b>II Software Technical Reports</b>	<b>104</b>
<b>7 GMC_NS.jl: Nested sampling by Galilean Monte Carlo</b>	<b>105</b>
7.1 Implementation notes . . . . .	105

7.2	Ensemble, Model, and Model Record interfaces . . . . .	106
7.3	Usage notes . . . . .	107
7.3.1	Setting up for a run . . . . .	107
7.3.2	GMC and PID tuner settings . . . . .	107
7.3.3	Parallelization . . . . .	108
7.3.4	Displays . . . . .	108
7.3.5	Eggbox diagnostic . . . . .	109
7.3.6	Example use . . . . .	110
7.4	Future Directions . . . . .	112
<b>8</b>	<b>CMZNicheSims.jl: <i>D. rerio</i> CMZ RPC niche simulators</b>	<b>114</b>
8.1	Introduction . . . . .	114
8.2	CMZ_Ensemble model and usage . . . . .	115
8.3	Slice_Ensemble model and usage . . . . .	116
8.4	Decay_Ensemble model and usage . . . . .	117
8.5	MultiSlice_Ensemble model and usage . . . . .	117
8.6	Thymidine_Ensemble model and usage . . . . .	118
<b>9</b>	<b>BioBackgroundModels.jl: Parallel optimization of Hidden Markov Model zoos of genomic background noise</b>	<b>120</b>
9.1	Genome partitioning and sampling . . . . .	121
9.2	Optimizing BHMMs by EM algorithms . . . . .	122
9.3	BHMM analysis and display . . . . .	124
9.4	Numerical accuracy and tests . . . . .	130
9.5	Future directions . . . . .	130
<b>10</b>	<b>BioMotifInference.jl: Independent component analysis motif inference by nested sampling</b>	<b>131</b>
10.1	Introduction . . . . .	131
10.2	Implementation of the nested sampling algorithm . . . . .	131
10.3	Usage notes . . . . .	133
10.3.1	Preparation of observation set . . . . .	133
10.3.2	IPM_Ensemble assembly . . . . .	134
10.3.3	Permute routine setup . . . . .	135
10.3.4	Nested sampling of IPM_Ensembles . . . . .	138
10.4	Test of model comparison logic with spiked motifs . . . . .	138
10.5	Future Directions . . . . .	138
<b>III</b>	<b>Supplementary Materials</b>	<b>140</b>
<b>11</b>	<b>Supplementary materials for Chapter 2</b>	<b>141</b>
11.1	Materials and methods . . . . .	141
11.1.1	Zebrafish husbandry . . . . .	141
11.1.2	Proliferative RPC Histochemistry . . . . .	141

11.1.2.1	Anti-PCNA histochemistry . . . . .	141
11.1.2.2	Cumulative thymidine analogue labelling for estimation of CMZ RPC cell cycle length . . . . .	142
11.1.2.3	Whole retina thymidine analogue labelling of post-embryonic CMZ con- tributions . . . . .	142
11.1.3	Confocal microscopy and image analysis . . . . .	143
11.1.4	Estimation of CMZ annular population size . . . . .	143
11.1.5	Modelling lens growth . . . . .	143
11.1.6	Estimation of 3dpf CMZ cell cycle length . . . . .	143
11.1.7	CHASTE Simulations . . . . .	144
11.1.7.1	Project code . . . . .	144
11.1.7.2	SPSA optimisation of models . . . . .	144
11.2	Supporting figures . . . . .	146
<b>12</b>	<b>Supplementary materials for Chapter 4</b>	<b>151</b>
12.0.1	Normal and LogNormal models of population data . . . . .	151
12.1	Materials and methods . . . . .	153
12.1.1	Zebrafish husbandry . . . . .	153
12.1.2	PCNA Immunohistochemistry . . . . .	154
12.1.3	Thymidine analogue histochemistry . . . . .	154
12.1.4	Lineage tracing immunohistochemistry . . . . .	154
12.1.5	4C4 immunohistochemistry . . . . .	155
12.1.6	Confocal micrograph acquisition and processing . . . . .	155
12.1.7	Retinal measurements . . . . .	155
12.1.8	Estimation of overall CMZ population and retinal volume . . . . .	155
12.1.9	Monte Carlo estimation of CMZ population and retinal volume rates of change .	156
12.1.10	Evidence estimation by Empirical Bayes linear regression . . . . .	156
12.1.11	Evidence estimation by Galilean Monte Carlo nested sampling . . . . .	156
12.1.12	Estimation of posterior distributions of phase model parameters . . . . .	156
12.2	Supplementary Figures . . . . .	156
12.3	Supplementary Tables . . . . .	163
<b>13</b>	<b>Supplementary material for Chapter 5</b>	<b>165</b>
13.1	Materials and methods . . . . .	165
13.1.1	Zebrafish husbandry . . . . .	165
13.1.2	Generation of transgenic <i>vsx2:eGFP rys</i> line . . . . .	165
13.1.3	Morpholino injections . . . . .	166
13.1.4	Morphological photography . . . . .	166
13.1.5	PCNA and EdU proliferative histochemistry . . . . .	166
13.1.6	BrdU pulse-chase assay . . . . .	166
13.1.7	Cumulative EdU labelling assay . . . . .	166
13.1.8	Analysis of <i>rys</i> nuclear parameters by Galilean Monte Carlo Nested Sampling .	166
13.1.9	Progenitor identity marker immunohistochemistry . . . . .	166
13.1.10	Caspase-3 immunohistochemistry . . . . .	166

13.1.11 Electron microscopy . . . . .	166
13.1.12 qPCR analysis of <i>rys</i> npat expression . . . . .	167
13.1.13 qPCR analysis of <i>rys</i> histone expression . . . . .	167
13.1.14 <i>rys</i> MNase digestions and NGS . . . . .	167
13.1.15 Nucleosome position calling . . . . .	167
13.1.16 Analyses of <i>rys</i> nucleosome position disposition . . . . .	168
13.1.16.1 Background Hidden Markov modelling of <i>D. rerio</i> genomic sequence emission . . . . .	168
13.1.17 Evidence and maximum a posteriori estimation of ICA models of <i>rys</i> nucleosome position sequence emission . . . . .	168
13.2 Supplementary figures . . . . .	169
13.3 Supplementary tables . . . . .	180
<b>14 Supplementary materials for all chapters</b>	<b>181</b>
14.1 Thesis git and HDD archives . . . . .	181
14.2 Code notes . . . . .	181
14.3 Computational cluster description and discussion . . . . .	182
<b>15 Theoretical Appendix A: Model theory and statistical methods</b>	<b>184</b>
15.1 Model Theory . . . . .	184
15.1.1 Bayesian Epistemological View on Model Comparison . . . . .	185
15.1.2 Model sampling and optimization . . . . .	187
15.1.3 Overfitting . . . . .	188
15.1.4 Simple Stochastic Models . . . . .	189
15.1.5 Systems of difference equations . . . . .	190
15.1.6 Independent Component Analysis models of sequence emission . . . . .	190
15.1.7 Position Weight Matrices . . . . .	191
15.1.8 Hidden Markov Models . . . . .	192
15.2 Statistical Methods . . . . .	192
15.2.1 Akaike Information Criterion scoring . . . . .	192
15.2.2 Monte Carlo simulations . . . . .	193
15.2.3 Simultaneous Perturbation Stochastic Approximation (SPSA) . . . . .	193
15.2.4 Bayesian treatment of Normal models of unknown mean and variance . . . . .	194
15.2.5 Empirical Bayes linear regression . . . . .	194
15.2.6 Galilean Monte Carlo . . . . .	195
15.2.7 Nested sampling . . . . .	196
15.2.8 Expectation-Maximization optimization of HMMs . . . . .	198
<b>16 Theoretical Appendix B: Metaphysical Arguments</b>	<b>199</b>
16.1 Chance is not a valid explanation for biological phenomena; Randomness is a measure of sequence compressibility and not an explanation . . . . .	199
16.1.0.1 Chance versus Randomness . . . . .	200
16.1.0.2 Chance in molecular mechanisms . . . . .	201
16.1.0.3 Can macromolecular chance be rooted in quantum indeterminacy? . . . . .	203

16.1.0.4	Randomness in RPC fate specification . . . . .	203
16.1.0.5	Summary: “Stochastic” or “variable”? . . . . .	204
16.2	Macromolecular mechanistic explanations in the Systems era . . . . .	205
16.2.1	The Feyerabendian modeller . . . . .	207
<b>17</b>	<b>Code Appendix</b>	<b>210</b>
17.1	SMME . . . . .	210
17.1.1	setup_project.py . . . . .	210
17.1.2	/apps/src/BoijeSimulator.cpp . . . . .	213
17.1.3	/apps/src/GomesSimulator.cpp . . . . .	220
17.1.4	/apps/src/HeSimulator.cpp . . . . .	226
17.1.5	/apps/src/WanSimDebug.cpp . . . . .	238
17.1.6	/apps/src/WanSimulator.cpp . . . . .	242
17.1.7	/python_fixtures/He_output_fixture.py . . . . .	249
17.1.8	/python_fixtures/Kolmogorov_fixture.py . . . . .	257
17.1.9	/python_fixtures/SPSA_fixture.py . . . . .	263
17.1.10	/python_fixtures/Wan_output_fixture.py . . . . .	280
17.1.11	/python_fixtures/diagram_utility_scripts/Gomes_He_cycle_plots.py . . . . .	284
17.1.12	/python_fixtures/diagram_utility_scripts/He_Boije_signal_plots.py . . . . .	285
17.1.13	/python_fixtures/diagram_utility_scripts/He_Det_signal_plots.py . . . . .	287
17.1.14	/python_fixtures/figure_plots/Cumulative_EdU.py . . . . .	289
17.1.15	/python_fixtures/figure_plots/He_output_plot.py . . . . .	290
17.1.16	/python_fixtures/figure_plots/Kolmogorov_plot.py . . . . .	310
17.1.17	/python_fixtures/figure_plots/Mitotic_rate_plot.py . . . . .	311
17.1.18	/python_fixtures/figure_plots/Wan_output_plot.py . . . . .	314
17.1.19	/src/BoijeCellCycleModel.cpp . . . . .	316
17.1.20	/src/BoijeCellCycleModel.hpp . . . . .	325
17.1.21	/src/BoijeRetinalNeuralFates.cpp . . . . .	330
17.1.22	/src/BoijeRetinalNeuralFates.hpp . . . . .	332
17.1.23	/src/GomesCellCycleModel.cpp . . . . .	335
17.1.24	/src/GomesCellCycleModel.hpp . . . . .	345
17.1.25	/src/GomesRetinalNeuralFates.cpp . . . . .	350
17.1.26	/src/GomesRetinalNeuralFates.hpp . . . . .	352
17.1.27	/src/HeAth5Mo.cpp . . . . .	356
17.1.28	/src/HeAth5Mo.hpp . . . . .	357
17.1.29	/src/HeCellCycleModel.cpp . . . . .	358
17.1.30	/src/HeCellCycleModel.hpp . . . . .	372
17.1.31	/src/OffLatticeSimulationPropertyStop.cpp . . . . .	378
17.1.32	/src/OffLatticeSimulationPropertyStop.hpp . . . . .	386
17.1.33	/src/WanStemCellCycleModel.cpp . . . . .	393
17.1.34	/src/WanStemCellCycleModel.hpp . . . . .	400
17.2	NGRefTools . . . . .	405
17.2.1	/src/LogNormalUtils.jl . . . . .	405
17.2.2	/src/MarginalTDist.jl . . . . .	406

17.2.3 /src/NGRef.jl . . . . .	410
17.2.4 /src/NGRefTools.jl . . . . .	411
17.2.5 /src/NIGRef.jl . . . . .	411
17.3 GMC_NS . . . . .	413
17.3.1 /src/GMC_NS.jl . . . . .	413
17.3.2 /src/GMC/galilean_trajectory.jl . . . . .	414
17.3.3 /src/ensemble/GMC_NS_Engineer.jl . . . . .	419
17.3.4 /src/ensemble/ensemble_utilities.jl . . . . .	421
17.3.5 /src/model/GMC_NS_Model.jl . . . . .	428
17.3.6 /src/model/model_utilities.jl . . . . .	429
17.3.7 /src/model/eggbox/Eggbox_Engineer.jl . . . . .	429
17.3.8 /src/model/eggbox/Eggbox_Model.jl . . . . .	432
17.3.9 /src/model/normal/LogNormal_Engineer.jl . . . . .	433
17.3.10 /src/model/normal/LogNormal_Model.jl . . . . .	436
17.3.11 /src/model/normal/Normal_Engineer.jl . . . . .	437
17.3.12 /src/model/normal/Normal_Model.jl . . . . .	440
17.3.13 /src/nested_sampler/converge_ensemble.jl . . . . .	441
17.3.14 /src/nested_sampler/nested_step.jl . . . . .	444
17.3.15 /src/nested_sampler/search_patterns.jl . . . . .	447
17.3.16 /src/utilities/coordinate_utils.jl . . . . .	450
17.3.17 /src/utilities/ellipsoid.jl . . . . .	451
17.3.18 /src/utilities/ns_progressmeter.jl . . . . .	453
17.3.19 /src/utilities/progress_displays.jl . . . . .	458
17.3.20 /src/utilities/stats.jl . . . . .	461
17.3.21 /src/utilities/t_Tuner.jl . . . . .	461
17.3.22 /test/eggbox_test.jl . . . . .	466
17.3.23 /test/ensemble_tests.jl . . . . .	466
17.3.24 /test/galilean_unit_tests.jl . . . . .	468
17.3.25 /test/normal_demo.jl . . . . .	469
17.3.26 /test/runtests.jl . . . . .	470
17.4 CMZNicheSims . . . . .	470
17.4.1 /src/CMZNicheSims.jl . . . . .	470
17.4.2 /src/CMZ_sim/CMZ_ensemble.jl . . . . .	471
17.4.3 /src/CMZ_sim/CMZ_lh.jl . . . . .	475
17.4.4 /src/CMZ_sim/CMZ_model.jl . . . . .	477
17.4.5 /src/cycle_decay_sim/decay_constructor.jl . . . . .	478
17.4.6 /src/cycle_decay_sim/decay_ensemble.jl . . . . .	478
17.4.7 /src/cycle_decay_sim/decay_lh.jl . . . . .	479
17.4.8 /src/multislice_sim/multislice_ensemble.jl . . . . .	481
17.4.9 /src/multislice_sim/multislice_model.jl . . . . .	485
17.4.10 /src/slice_sim/lens_model.jl . . . . .	487
17.4.11 /src/slice_sim/slice_ensemble.jl . . . . .	487
17.4.12 /src/slice_sim/slice_lh.jl . . . . .	490

17.4.13 /src/slice_sim/slice_model.jl . . . . .	492
17.4.14 /src/thymidine_sim/thymidine_cell.jl . . . . .	493
17.4.15 /src/thymidine_sim/thymidine_ensemble.jl . . . . .	493
17.4.16 /src/thymidine_sim/thymidine_lh.jl . . . . .	499
17.4.17 /src/thymidine_sim/thymidine_model.jl . . . . .	501
17.4.18 /src/thymidine_sim/thymidine_sim.jl . . . . .	502
17.5 BioBackgroundModels . . . . .	505
17.5.1 /src/BioBackgroundModels.jl . . . . .	506
17.5.2 /src/API/EM_master.jl . . . . .	507
17.5.3 /src/API/genome_sampling.jl . . . . .	511
17.5.4 /src/API/reports.jl . . . . .	515
17.5.5 /src/BHMM/BHMM.jl . . . . .	516
17.5.6 /src/EM/EM_converge.jl . . . . .	518
17.5.7 /src/EM/baum-welch.jl . . . . .	519
17.5.8 /src/EM/chain.jl . . . . .	523
17.5.9 /src/EM/churbanov.jl . . . . .	524
17.5.10 /src/genome_sampling/partition_masker.jl . . . . .	527
17.5.11 /src/genome_sampling/sequence_sampler.jl . . . . .	535
17.5.12 /src/likelihood_funcs/bg_lh_matrix.jl . . . . .	545
17.5.13 /src/likelihood_funcs/hmm.jl . . . . .	548
17.5.14 /src/reports/chain_report.jl . . . . .	550
17.5.15 /src/reports/partition_report.jl . . . . .	554
17.5.16 /src/reports/replicate_convergence.jl . . . . .	556
17.5.17 /src/utilities/BBG_analysis.jl . . . . .	561
17.5.18 /src/utilities/BBG_progressmeter.jl . . . . .	562
17.5.19 /src/utilities/HMM_init.jl . . . . .	565
17.5.20 /src/utilities/load_balancer.jl . . . . .	565
17.5.21 /src/utilities/log_prob_sum.jl . . . . .	567
17.5.22 /src/utilities/model_display.jl . . . . .	567
17.5.23 /src/utilities/observation_coding.jl . . . . .	568
17.5.24 /src/utilities/utilities.jl . . . . .	571
17.5.25 /test/ref_fns.jl . . . . .	571
17.5.26 /test/runtests.jl . . . . .	575
17.5.27 /test/synthetic_sequence_gen.jl . . . . .	601
17.6 BioMotifInference . . . . .	603
17.6.1 /src/BioMotifInference.jl . . . . .	604
17.6.2 /src/IPM/ICA_PWM_Model.jl . . . . .	605
17.6.3 /src/IPM/IPM_likelihood.jl . . . . .	611
17.6.4 /src/IPM/IPM_prior_utilities.jl . . . . .	617
17.6.5 /src/ensemble/IPM_Eensemle.jl . . . . .	620
17.6.6 /src/ensemble/ensemble_utilities.jl . . . . .	625
17.6.7 /src/nested_sampler/converge_ensemble.jl . . . . .	631
17.6.8 /src/nested_sampler/nested_step.jl . . . . .	634

17.6.9 /src/permuation/Permute_Tuner.jl . . . . .	638
17.6.10 /src/permuation/orthogonality_helper.jl . . . . .	641
17.6.11 /src/permuation/permute_control.jl . . . . .	644
17.6.12 /src/permuation/permute_functions.jl . . . . .	649
17.6.13 /src/permuation/permute_utilities.jl . . . . .	664
17.6.14 /src/utilities/model_display.jl . . . . .	670
17.6.15 /src/utilities/ns_progressmeter.jl . . . . .	674
17.6.16 /src/utilities/synthetic_genome.jl . . . . .	680
17.6.17 /src/utilities/worker_diagnostics.jl . . . . .	682
17.6.18 /src/utilities/worker_sequencer.jl . . . . .	683
17.6.19 /test/consolidate_unit_tests.jl . . . . .	684
17.6.20 /test/ensemble_tests.jl . . . . .	686
17.6.21 /test/lh_perf.jl . . . . .	690
17.6.22 /test/likelihood_unit_tests.jl . . . . .	690
17.6.23 /test/mix_matrix_unit_tests.jl . . . . .	696
17.6.24 /test/permute_func_tests.jl . . . . .	697
17.6.25 /test/permute_tuner_tests.jl . . . . .	703
17.6.26 /test/pwm_unit_tests.jl . . . . .	704
17.6.27 /test/runtests.jl . . . . .	707
17.6.28 /test/spike_recovery.jl . . . . .	708
17.7 AWSWrangler . . . . .	711
17.7.1 /src/AWSWrangler.jl . . . . .	711
17.8 Analyses . . . . .	714
17.8.1 /cmz/a10correlation.jl . . . . .	714
17.8.2 /cmz/a10decayslicediagnostic.jl . . . . .	717
17.8.3 /cmz/a10dvdecayslice.jl . . . . .	719
17.8.4 /cmz/a10dvratio.jl . . . . .	726
17.8.5 /cmz/a10dvslice.jl . . . . .	728
17.8.6 /cmz/a10morphology.jl . . . . .	736
17.8.7 /cmz/a10nvln.jl . . . . .	741
17.8.8 /cmz/a10periodisation.jl . . . . .	745
17.8.9 /cmz/a10popsurvey.jl . . . . .	752
17.8.10 /cmz/a19lineagetrace.jl . . . . .	758
17.8.11 /cmz/a20dvratio.jl . . . . .	771
17.8.12 /cmz/a25dvlinreg.jl . . . . .	774
17.8.13 /cmz/a25thymidinesim.jl . . . . .	776
17.8.14 /cmz/a27GMC_NS.jl . . . . .	779
17.8.15 /cmz/a27linreg.jl . . . . .	781
17.8.16 /rys/a35thymidinesim.jl . . . . .	784
17.8.17 /rys/a37.jl . . . . .	788
17.8.18 /rys/a38.jl . . . . .	792
17.8.19 /rys/a38GMC_NS.jl . . . . .	795
17.8.20 /rys/caspase.jl . . . . .	798

17.8.21/rys/em.jl . . . . .	800
17.8.22/rys/occupancy.py . . . . .	801
17.8.23/rys/qPCR.jl . . . . .	808
17.8.24/rys/rysp_GMC_NS.jl . . . . .	814
17.8.25/rys/ryspont.jl . . . . .	818
17.8.26/rys/bg_models/BBM_refinement_analysis.jl . . . . .	823
17.8.27/rys/bg_models/BBM_refinement_prep.jl . . . . .	823
17.8.28/rys/bg_models/BBM_sample_prep.jl . . . . .	824
17.8.29/rys/bg_models/BBM_survey.jl . . . . .	826
17.8.30/rys/bg_models/BBM_survey_analysis.jl . . . . .	828
17.8.31/rys/bg_models/BBM_survey_refinement.jl . . . . .	829
17.8.32/rys/nested_sampling/dif_pos_assembly.jl . . . . .	831
17.8.33/rys/nested_sampling/dif_pos_learner.jl . . . . .	834
17.8.34/rys/nested_sampling/dif_pos_sample_prep.jl . . . . .	837
17.8.35/rys/nested_sampling/local_test.jl . . . . .	840
17.8.36/rys/nested_sampling/overall_naive.jl . . . . .	841
17.8.37/rys/nested_sampling/prior_test.jl . . . . .	844
17.8.38/rys/nested_sampling/spike_recovery.jl . . . . .	848
17.8.39/rys/position_analysis/position_overlap.jl . . . . .	852
17.8.40/test/eggbox_plots.jl . . . . .	856

## Bibliography

858

# List of Tables

2.1 AIC values for stochastic and deterministic alternative models . . . . .	29
4.1 Evidence favours a 2-phase periodization of CMZ activity . . . . .	50
4.2 Maximum a posteriori parameter estimates for periodization models . . . . .	51
4.3 Evidence favours a combined slice model over separate dorsal and ventral models	54
4.4 Maximum a posteriori parameter estimates for slice models . . . . .	56
4.5 Evidence supports separate dorsal and ventral decay models . . . . .	59
4.6 Maximum a posteriori parameter estimates for decay models . . . . .	59
4.7 Evidence favours whole-CMZ linear cycle models over separate D/V models	61
4.8 Evidence supports stable layer contributions with time-varying fate commitment	64
4.9 Evidence for linear regression models supports early cohort population stability	68
4.10 GMC-NS evidence estimates confirm Empirical Bayes analysis of early cohort stability . . . . .	69
5.1 Evidence-based ranking of phenomenal contributors to <i>rys</i> phenotype . . . . .	80
5.2 Standard deviations of significance for <i>rys</i> mutant transcript > sib or WT . . .	88
5.3 Evidence favours separate emission models for the <i>rys</i> mutant and sibling differential position sets . . . . .	92
12.1 Likelihood ratio comparison between Normal and Log-Normal models of retinal population parameters . . . . .	151
12.2 Evidence favours Log-Normal models of retinal population parameters . . . .	152
12.3 Evidence favours uncorrelated linear models of CMZ-population and retinal volume over time . . . . .	153
12.4 Evidence for Normal vs. Log-Normal models of layer and lineage contribution	163
12.5 Evidence for combined vs. separate models of layer and lineage contribution across the dorso-ventral axis . . . . .	163
13.1 Evidence for separate models of retinal parameters in anti-npat morpholino-injected animals . . . . .	180
13.2 <i>rys</i> mutant differential nucleosome positions are less typical of the genome than those mapped to sibling positions . . . . .	180

# List of Figures

1.1	Section of 3dpf zebrafish eye, displaying CMZ RPCs . . . . .	7
1.2	Histogenetic birth order of retinal neurons . . . . .	9
2.1	Structure of Gomes SSM . . . . .	22
2.2	Structure of He SSM . . . . .	23
2.3	Structure of Boije SSM . . . . .	24
2.4	An alternative, deterministic mitotic mode model . . . . .	26
2.5	Model comparison: the SPSA-optimised He SSM and a deterministic alternative	28
2.6	Most zebrafish retinal neurons are contributed by the CMZ between one and three months of age . . . . .	30
2.7	Per-lineage probabilities of mitoses, He et al. observations compared to model output . . . . .	31
2.8	CMZ population of proliferating RPCs: estimates from observations and simulated Wan-type CMZs . . . . .	33
3.1	Whole-eye and slice model abstractions of CMZ RPC populations used in Chapter 4 . . . . .	41
4.1	Micrographic overview of CMZ development in <i>D.rerio</i> , 3-90dpf . . . . .	46
4.2	Population and activity of the CMZ over the first year of <i>D.rerio</i> life . . . . .	47
4.3	Maximum a posteriori output of periodization models . . . . .	49
4.4	Kernel density estimates of marginal posterior parameter distributions, 2-phase model . . . . .	52
4.5	Developmental progression of dorso-ventral population asymmetry in the CMZ .	53
4.6	Maximum a posteriori output of total, dorsal, and ventral CMZ slice models .	55
4.7	Kernel density estimates of marginal posterior parameter distributions, total slice model . . . . .	57
4.8	Maximum a posteriori output of total, dorsal, and ventral CMZ decay models	58
4.9	Kernel density estimates of marginal posterior parameter distributions, decay models . . . . .	60
4.10	MAP model output and observations for thymidine slice model of 3dpf cumulative EdU labelling . . . . .	62
4.11	Marginal posterior distributions for thymidine slice model . . . . .	63
4.12	Representative 23dpf lineage marker confocal micrographs . . . . .	66

4.13	Second-phase declines in CMZ-contributed IslPax6+ RGCs, PKC $\beta$ + bipolar neurons, and Zpr1+ double cones . . . . .	67
4.14	4C4+ microglia associate with and engulf EdU-labelled CMZ contributions in the specified neural retina . . . . .	69
5.1	<i>rys</i> mutants exhibit a small-eye phenotype . . . . .	73
5.2	Typical <i>rys</i> sibling and mutant retinal organisation and appearance . . . . .	75
5.3	<i>rys</i> CMZ populations start relatively small and quiescent, end abberantly large and proliferative . . . . .	76
5.4	<i>rys</i> CMZ RPCs fail to contribute to the neural retina . . . . .	77
5.5	Cycling RPCs in 5dpf <i>rys</i> mutant retinas have longer cell cycle times . . . . .	78
5.6	7dpf <i>rys</i> CMZ RPCs display enhanced asymmetry, increased volume, decreased sphericity, and relative but not absolute enlargement . . . . .	79
5.7	RPC nuclei of the <i>rys</i> CMZ display disorganized, loosely packed chromatin . .	81
5.8	<i>rys</i> mutant CMZs have increased caspase-3 positive nuclei . . . . .	82
5.9	Mutant <i>rys</i> RPCs display expanded expression of early progenitor markers .	83
5.10	RT-PCR analysis reveals two aberrant intron retention variants in <i>rys</i> mutant npat transcripts . . . . .	84
5.11	Functional domains of Human NPAT compared to predicted wild-type and <i>rys</i> <i>Danio</i> npat . . . . .	85
5.12	In situ hybridization reveals progressive restriction of npat expression to the CMZ . . . . .	86
5.13	<i>rys</i> overexpress total and polyadenylated core histone transcripts . . . . .	87
5.14	48hpf embryos injected with an npat ATG-targeted morpholino display a small-eye phenotype . . . . .	89
5.15	ATG morpholino perturbation of npat recapitulates decline in mean central retinal population relative to CMZ . . . . .	90
5.16	<i>rys</i> chromosomes are differentially enriched and depleted of nucleosome position density and occupancy. . . . .	93
5.17	PWM sources detected in sibling differential nucleosome positions. . . . .	94
5.18	PWM sources detected in sibling differential nucleosome positions. . . . .	95
5.19	PWM sources detected in <i>rys</i> mutant differential sources. . . . .	96
7.1	GMC-NS.jl nested sampling the eggbox toy model . . . . .	110
7.2	Example GMC-NS.jl output . . . . .	112
9.1	Example BBM.jl Partition Report . . . . .	126
9.2	Example BBM.jl Replicate Report . . . . .	127
9.3	Example BBM.jl Chain Report - Part 1 . . . . .	128
9.4	Example BBM.jl Chain Report - Part 2 . . . . .	129
9.5	Example BBM.jl Chain Report - Part 3 . . . . .	129
12.1	CMZ population and retinal volume estimates are uncorrelated at 3dpf . . .	153
12.2	Overview of <i>D. rerio</i> retinal parameters during morphogenesis . . . . .	157
12.3	Developmental progression of naso-temporal population asymmetry in the CMZ.	158

12.4 Polymodality of CMZ model MAP estimates . . . . .	159
12.5 Linear regressions performed on cumulative labelling data from dorsal, ventral, and combined CMZ sectional populations . . . . .	160
12.6 Linear regressions of temporally correlated and uncorrelated models of central retinal and 30dpf CMZ-contributed cohorts . . . . .	161
12.7 4C4+ microglia are associated with the CMZ . . . . .	162
 13.1 10dpf mutant <i>rys</i> RPCs are mitotic . . . . .	169
13.2 MAP model output and observations for <i>rys</i> sibling thymidine slice model of 5dpf cumulative EdU labelling . . . . .	170
13.3 MAP model output and observations for <i>rys</i> mutant thymidine slice model of 5dpf cumulative EdU labelling . . . . .	171
13.4 4C4-positive microglia engulf <i>rys</i> mutant RPCs . . . . .	172
13.5 Synteny Database output for the syntenic region containing <i>D. rerio</i> npat . . . . .	173
13.6 npat is overexpressed in <i>rys</i> . . . . .	174
13.7 npat morpholino injectants do not recapitulate <i>rys</i> retinal disorganisation phenotype . . . . .	175
13.8 Morpholinos directed to npat result in histone transcript overexpression similar to <i>rys</i> . . . . .	176
13.9 Novel nucleosome positions in <i>rys</i> occur in similar numbers to those lost from sibs. . . . .	177
13.10 Test likelihoods for background HMMs trained on <i>D. rerio</i> genome partitions . . . . .	178
13.11 PWM sources detected in combined sib and rys differential sources. . . . .	179
 15.1 Simple stochastic stem cell model, representing probabilities of cell division events. . . . .	189
15.2 The four directions of GMC particle movement. . . . .	196
15.3 Schematic of evidence integration by nested sampling. . . . .	197
 16.1 Cellular systems model-construction. . . . .	205

## Introductory Notes

### Thesis Guide

This document is split into three parts. Part I contains results and discussion pertaining to empirical modelling studies that will be of interest to committee members and developmental biologists. Readers wishing to confine themselves to results chapters should examine [Chapter 2](#), [Chapter 4](#), and [Chapter 5](#). Part II contains technical reports pertaining to novel software that was written in order to perform the analyses presented in Part I. Part III contains supplementary materials arising from Parts I and II. These include detailed descriptions of the methods used in Part I, less-technical explanations of relevant statistical and model theory, the source code of all software and analyses, and the bibliography. The source code has been omitted from the print document.

Synopses have been provided in italics at the top of each chapter in Part I. These consist of numbered statements, which summarise the accordingly numbered section in the text.

Readers of the .pdf document will find some text highlighted in plum throughout the thesis. Section indices highlighted in this way link to the specified section. Because empirical results in Part I are separated from relevant methods and code in Part III, links to the methods and code used to generate the figure's data and analysis are given in figure captions. Technical terms have also been highlighted when pertinent material is available in Part III to explain them for the reader who may be unfamiliar.

Readers who wish to replicate analyses, inspect datasets, etc., will find guidance in [Chapter 14](#).

### Terminology and Style

Throughout [Chapter 1](#), [Chapter 2](#), and [Chapter 3](#), I have used the appellation "Harris" when referring to the output of William Harris' research group. This is a large body of research spanning several decades, and involves many co-authors. My use of "Harris" here is a convenience, as Harris is the only common author across the period in question, and presumably the agent carrying these ideas forward from project to project. It is not intended to slight or minimize the contributions of any of the other members of Harris' group (many of whom are now senior scientists in their own right). I have used "Raff" in an identical sense in referring to Martin Raff research group's work in [Chapter 3](#).

I have preferred the terms "specification", "determination", and their derivatives, to refer to cells assuming a particular lineage fate. "Differentiation" is well-understood, but ambiguous, and often understood to relate to the mitotic event itself, which I generally do not intend. "A cell has specified" means it has assumed a stable macromolecular identity or "fate". This term is intended to correspond exactly with the appearance of stable markers of cell type.

Unless otherwise noted, formatting of quoted material is preserved, so that italic emphases appear in the original.

# **Part I**

# **Modelling Studies**

# Chapter 1

## Canonical retinal progenitor cell phenomena and their explanations

### Synopsis

(1) *The Stochastic Mitotic Mode Explanation (SMME) for retinal progenitor cell (RPC) function is the first computational model which claims to explain RPC behaviours throughout the life of a vertebrate, and so, retinal morphogenesis.* (2) *This would resolve decades of theoretical deadlock between competing explanations for RPC behaviours.* (3) *The majority of RPC phenomena are understood in terms of proliferative and fate specification outcomes of RPC lineages.* (4) *Macromolecular explanations for these outcomes are diverse, and difficult to reconcile into a harmonious explanation.* (5) *The SMME can be understood as an effort to find order by blurring out some pertinent explanatory details.* (6) *The SMME promises to clarify the macromolecular underpinnings of morphogenesis in a complex neural tissue by explaining it in terms of a stochastic molecular process.*

### 1.1 The Stochastic Mitotic Mode Explanation (SMME)

The work presented in [Chapter 2](#) examines the best-developed theory of *D. rerio* retinal progenitor cell (RPC) function. The theory in question is the work of preëminent retinal biologist William Harris' research group, referred to hereafter Stochastic Mitotic Mode Explanation (SMME). This theory purports to explain the function of zebrafish RPCs in terms of stochastic effects on mitotic mode, which specifies the mitotic state and fate propensity of offspring.

The SMME for RPC function is of fundamental theoretical and practical interest. It is a concerted effort to explain how a complex tissue like a retina can arise from a field of similar proliferating cells. In 2009, Harris coauthored a detailed review chapter, documenting the bewildering array of macromolecules and cellular processes thought to be involved in RPC function [[AH09](#)]. This enumeration contains many caveats and notes that the effects of particular macromolecules routinely differ between developmental stages, cell types, organisms, and so forth. At this time, no clear, detailed, comprehensive models of RPC function had been advanced, and the review is typified by statements like "It is difficult to reconcile

all the studies on the initiation and spread of neurogenesis in a single model.”<sup>1</sup> It is therefore remarkable that, over the next nine years, a simple stochastic model of zebrafish RPC function invoking two named macromolecules would dominate the RPC literature.

The explanation aims to advance a simple, comprehensible, “mind-sized” model of RPC function, which is a microcosm of the broader promise of “Systems Biology” to make sense of the contemporary welter of conflicting datasets. By using sophisticated mathematical methods drawn from information and complexity theories, the apparent confusion will be clarified and underlying molecular mechanisms will be revealed. Given Harris’ track record and preëminence in the field, we have good reason to take seriously the possibility that the SMME has succeeded. Examining whether this is the case is our first priority. To appreciate the scope of his theoretical maneuver, and possible alternatives to it, we begin by summarising the state of the art at the time of his 2009 review (as well as relevant subsequent additions).

## 1.2 Explanations for RPC function in 2009 and the drive to unification

Molecular biologists have sought explanations for the same remarkable features of retinal progenitor cells for decades. Animal retinas, having well-understood functions and stereotyped structures, promise tractable targets for typical molecular biological explanations. With well defined cell types present in tightly regulated proportions and topological organisations, both theoretically-inclined molecular biologists and clinically-inclined regenerative medical practitioners have found the retina to be a useful model tissue<sup>2</sup>. The regular, easily detected order in eyes seemed to suggest a similar level of order and regularity in the macromolecular processes which underlay the formation of the tissue. Older explanations commonly suggested that RPCs are more-or-less identical and go through rigidly stereotypical macromolecular processes. It is, however, the persistently observed departures from this conception that have occupied much of our attention.

The simplest explanation for the regularity of retinal development is that any given RPC is executing the same strict “developmental program” as its neighbours. If every progenitor produces a similar number of cells, and the progeny are specified in similar proportions, well-understood principles of cellular adhesion could give rise to the characteristic laminar organisation observed in animal retinas. However, by the 1980s, vertebrate lineage tracing experiments revealed a surprising degree of inter-lineage variability in many neural progenitor systems, not least of which was the retina. In their seminal 1987 paper, David Turner and Connie Cepko, using retroviral lineage labelling techniques, demonstrated that individual RPC lineages in rats had diverse proliferative and fate specification outcomes [TC87]; Harris’ group confirmed this result in *Xenopus* the subsequent year [HBEH88], suggesting this variability was a common feature of vertebrate RPC function.

Indeed, at this point, we find fairly clear accounts of what retinal biologists took their theoretical options to be in explaining this variability. As Harris’ 1988 report states:

Changes in cell character associated with cell type diversification may be controlled in an autonomous way, reflecting either a temporal program inside the cell (Temple and Raff, 1986),

---

<sup>1</sup>This was in no sense a problem with Harris’ understanding. A similarly high-level review coauthored by Pam Raymond [AR08] concluded, with regard to models of photoreceptor fate specification: “The data reviewed in the preceding sections indicate that a ‘one-size-fits-all’ model is not possible...”

<sup>2</sup>Indeed, if central neuroregenerative medicine is to become a clinical possibility, it seems likely that the theoretical and practical issues are most likely to be resolved in eyes before other areas of the CNS.

the asymmetrical segregation of cytoplasmic determinants (Strome and Wood, 1983; Sulston and Horvitz, 1977), stochastic events inside the cell (Suda et al., 1984), or some combination of these processes. Alternatively, cell type may be controlled in a nonautonomous way, as in cases in which the extracellular environment (Doupe et al., 1985) or cellular interactions (Ready et al., 1976) elicit or limit cell fate. With its multiplicity of cell types, the vertebrate nervous system would seem to require the ultimate sophistication in its means of cellular determination. [HBEH88]

It is striking, then, that Harris' review of the literature two decades later describes the situation similarly:

Once differentiation is initiated, regulatory mechanisms within the retina ensure that progenitors retain the capacity to undergo more divisions, in parallel with churning out differentiated cells, and that progenitors cease dividing at variable times. There is still debate about the extent of early programming that allows progenitors to step through a series of stereotypical divisions and the extent of regulation from within the whole retina. The production of differentiated cells alters the retinal environment with time...

Moreover, cells from the same clone do not all differentiate at the same time, suggesting three possibilities: a stochastic mechanism for the decision to differentiate, exposure of the two daughters to different environments, or asymmetric inheritance of determinants. [AH09]

In the same paper, he states that the “simple structure and accessibility of the retina make it a useful model to study cell division and differentiation, and as a result most aspects of this have been studied, from lineage tracing of progenitors, to the morphological aspects of division, to the molecular mechanisms involved.” Thus, by Harris' own account, some twenty years of additional research into almost every variety of macromolecular explanation for a huge range of RPC-related phenomena had not provided any means to narrow down the possibilities he had already laid out in 1988. We still have Raff's temporal program (“early programming … step[ping] through a series of stereotypical divisions”), asymmetric segregation of cytoplasmic inheritants during mitotic events, “stochastic” events internal to the cells, and possible “environmental” extracellular determinants. While the number of particular macromolecules functionally implicated in proliferative, specificative, and organisational RPC phenomena had greatly expanded, this had not provided any means to differentiate between these theoretical options. The incremental enumeration of macromolecules involved in neurogenesis has not given rise to a coherent model of RPC function. Harris' SMME therefore represents an example of a “Systems” biological explanation, in which biologists apply the analytical and interpretative methods of the physical and mathematical sciences in an effort to resolve the problems posed by biological complexity, without attempting to assemble all of the relevant observations of previous studies into a single explanation [Mor09].

Before proceeding to the SMME itself, let us briefly summarise the diversity of phenomena implicated in RPC function by 2009, as well as the panoply of mechanisms offered as explanations. In doing so, it will become clear what has been elided in the SMME, and what may need to be restored in any alternative modelling approach.

## 1.3 Canonical vertebrate RPC phenomena: the RPC “morphogenetic alphabet”

The bulk of our knowledge of RPC behaviour stems from histological observation employing a limited number of techniques. Simple observations of mitotic figures in a variety of animals had, by the 1950s, revealed the surprising diversity of RPC proliferative phenomena across vertebrate clades. However, it was the advent of lineage tracing techniques, particularly those marking single clonal lineages in whole retinae, and the extensive use of these techniques in the 1980s-90s, that formed most of the basic body of observations that any macromolecular explanation is now called upon to account for.

Since the majority of vertebrate retinas of biomedical interest are mammalian, and these retinae are fully formed in an early developmental period, RPC behaviour has been best-studied in an embryonic and early developmental context. Here, vertebrate RPCs are derived from the eye field population of the early neural plate and later neural tube. This population is separated into left and right eye primordia, which in turn pouch outwards toward the ectoderm, and, in conjunction with the lens placode (itself induced from the ectoderm), form the optic vesicle. The primitive eye is formed when this vesicle completes a complex morphological folding process, resulting in the cup-shaped structure of the retina [Cav18]. During this process, the cells of the neural retina are differentiated from the overlying retinal pigmented epithelium (RPE). Sometime after the formation of the retinal cup, RPCs begin to exit the cell cycle and are specified as retinal neurons. Studies of this early period revealed numerous difficult-to-explain features of RPC behaviour. Following Larsen’s observation that tissue form is attributable to only six behaviours [Lar92], which she describes as the “morphogenetic alphabet”, I have categorised RPC phenomena as relating to proliferation, fate specification, migration, growth, death, and extracellular matrix formation. Since the vast majority of reported phenomena fall under the first two categories, proliferation and specification, these are given separate sections below, while those belonging to the last four are described collectively.

### 1.3.1 Proliferative phenomena

Clonal lineage tracing experiments reliably find that vertebrate RPCs give rise to highly variable numbers of offspring over the collective “lifetime” of the lineage. The most dramatic of these findings demonstrate that rat RPC lineage sizes vary across two orders of magnitude *in vivo*, from 1 to over 200 [TSC90]. While at least some of this variability must be related to differential integration of lineage markers into “older” (giving fewer offspring) and “newer” (giving more) RPCs, vertebrate RPC lineages nevertheless differ widely in their fecundity. The physical organisation of these clones is complex; as detailed below, RPC progeny may appear in any of the 3 retinal layers in a wide variety of specified fate combinations, and may engage in short-range migrations to appropriate positions for their specified fates. Most clonal lineages are “extinguished”; that is, after some time, all of its members have become postmitotic. However, it has long been noted that not all cells produced by RPCs are strictly postmitotic neurons; specified Müller glia retain the ability to reenter the cell cycle in response to stimuli (normally, retinal damage) [DC00, FR03], and peripheral circumferential marginal zone (CMZ) RPCs remain proliferative in those vertebrates whose eyes grow beyond early development (notably in frogs and fish, while the chick retina has a CMZ of more limited output [FR00]). Therefore, some clonal lineages may be organised into clumps associated with Müller responses, while others in frogs and fish may continue to be “plated out” in a more-or-less linear manner at the retinal periphery for as long as the lineage “lives” [CHW11]. This

ongoing RPC contribution to peripheral neurogenesis has long been recognised, so that by 1954 we find the following statement introducing a study of unusual mitoses in the retina of a deepsea fish:

It is conventional<sup>3</sup> to hold that the growth of the vertebrate retina is only possible due to the presence, in this tissue, of a peripheral germinal zone. In this region, young elements actively multiply, and, by subsequent differentiation, give rise to the diverse nervous and sensory constituents of the retina. [VL54]

[author's translation from the French]

Despite this, the proliferating RPCs in this “peripheral germinal zone” (also known as the “ciliary marginal zone”, or CMZ, for its proximity to the retinal ciliary body) have not received the same level of attention as those associated with the central retina. As a consequence, these RPCs are generally treated as though they are a type of “frozen” progenitor population, recapitulating spatially, along the peripheral-central axis, the process which RPCs in the central retina undergo in a time-dependent fashion [HP98]. Figure 1.1 displays a coronal cryosection through a zebrafish eye, with the proliferating RPCs of the CMZ highlighted in magenta; these cells contribute to all three of the cellular layers of the neural retina in adulthood.

The length of the RPC cell cycle is of considerable interest, since the evolution of this parameter in time, in conjunction with the RPC population size (the number of cells specified in the eye field), determines the eventual size of differentiated retinal neural population, and therefore the retina. RPC cell cycle length is usually inferred from clonal lineage size, although it has also been assayed directly in cumulative thymidine analogue labelling experiments [AC96]. Vertebrate RPCs undergo a period of relative quiescence, in which the cell cycle lengthens, before the neural retina begins to be specified (in zebrafish, this period is 16-24 hpf). The cell cycle shortens as RPCs begin to exit the cell cycle [HH91, LHO<sup>+00</sup>]. After the central retina is specified, the RPC cell cycle again lengthens, and is presumed to continue to slow until RPCs have completed specifying<sup>4</sup>.

Finally, the orientation of the RPC division plane in mitosis is also implicated in retinal organisation. The orientation of divisions is associated with both proliferation and fate specification of RPC progeny. For instance, interfering with spindle orientation in the developing rat retina, such that more RPC divisions occur parallel to the neuroepithelial plane (rather than along the apico-basal axis) results in more proliferative and fewer postmitotic, specified progeny [ZCC<sup>+05</sup>]. That said, it seems that whatever effects are attributed to mitotic orientation are likely species-specific, as zebrafish RPCs display a different pattern of axis orientation, dividing mainly in the epithelial plane [DPCH03].

### 1.3.2 Fate specification phenomena

Offspring of vertebrate RPC lineages may enter any of the three cellular layers of the retina. Moreover, single lineages can include any possible combination of cell fates, so that RPCs cannot be readily divided into “types” on the basis of lineage fate outcomes [HBEH88, TSC90, WF88]. While some progenitors have propensities to generate similar cell types, these relations seem species-specific, and may not define separate progenitor pools [AR08]. In general, then, RPCs are taken to be multipotent with respect to the neural retina- all of the cell types<sup>5</sup> of the differentiated retina are derived from similar RPC

<sup>3</sup>Unfortunately, I am unable to locate the source of this convention, likely due to the poor preservation of many of these older reports. That this required no citation in 1954 suggests the original observations of CMZ proliferation may be in the early 20th century.

<sup>4</sup>The work presented in Section 8.4 substantially confirms this presumption, at least after 3dpf in the zebrafish.

<sup>5</sup>The “cell type” concept is unusually well-defined in the retina, as there are an abundance of distinct morphological and molecular features which differentiate numerous subtypes of the seven general types of retinal neuron.

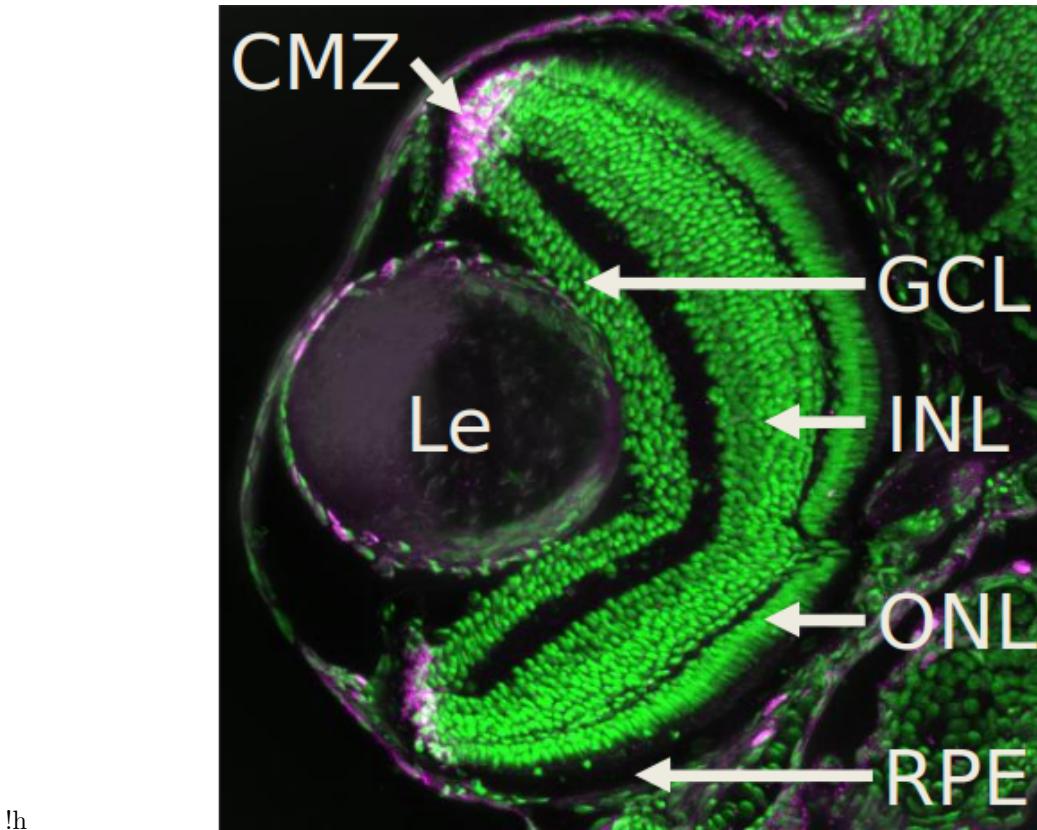


Figure 1.1: Section of 3dpf zebrafish eye, displaying CMZ RPCs

Maximum intensity projection of micro14 [metre] central coronal cryosection of 3dpf *D. rerio* eye, displaying proliferative cells labelled with anti-PCNA (magenta) against a Hoechst 33342 nuclear counterstain (green). CMZ: Circumferential marginal zone. GCL: Ganglion cell layer. INL: Inner nuclear layer. ONL: Outer nuclear layer. RPE: Retinal pigmented epithelium. In animals with a CMZ, the retina is grown by peripheral addition to the GCL, INL, and ONL by the CMZ.

lineages. Little about this picture has changed since its initial development, using a variety of lineage tracers (including retroviruses, thymidine analogues, and injectable dyes) and histochemical markers to supplement morphological identification of specified neurons. In particular, sophisticated modern live imaging experiments in zebrafish (many pioneered by Harris), have broadly confirmed the findings of the 80s and 90s in mammalian fixed specimens, explants, and live animals [BRD<sup>+</sup>15].

Of particular note are the observations of the Raff group [WR90, CBR03], who demonstrated that dissociated rat RPCs, cultured at clonal density, took on morphological and histochemical features associated with different specified neural types in similar numbers and proportions, and on a similar schedule, to same-aged RPCs cultured in retinal explants. These results dramatically suggested that both the proliferative and fate specification behaviour of RPCs depends less on intercellular contact, and the complex signalling environment of the developing retina, than on factors intrinsic to the RPCs themselves. These studies contain the essential germ of Harris' eventual commitment to SSM explanations, purporting as they do to "test the relative importance of cell-intrinsic mechanisms and extracellular signals in cell fate choice", and providing convincing evidence for the preponderant importance of the purported cell-intrinsic mechanisms.

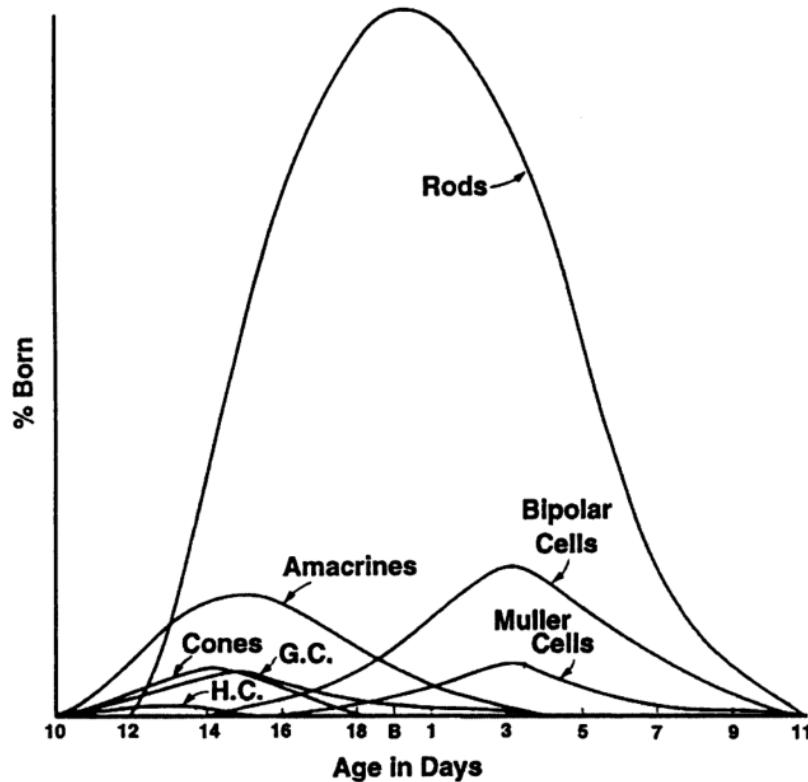


Figure 1.2: Histogenetic birth order of retinal neurons

Adapted from [You85] by [CAY<sup>+</sup>96], Copyright 1996 National Academy of Sciences. G.C., Ganglion Cells; H.C., Horizontal Cells. Data from embryonic and perinatal mouse eyes.

In spite of the apparent ability of RPCs to produce offspring specified to any of the possible neural cell fates, at any time during their lineage history<sup>6</sup>, vertebrate retinal development displays a temporal

<sup>6</sup>Even in papers arguing for a strict, linear sequence of speciative outcomes in all RPC lineages, the actual data show

ordering, such that in any particular location, retinal ganglion cells (RGCs) tend to be produced first, followed by the other cell types in what has been described as an overlapping “histogenetic order”, pictured in Figure 1.2. As noted above, RPCs also exit the cell cycle in a spatiotemporally defined order (from central to peripheral over time), and specification follows the same pattern<sup>7</sup>. This naturally gave rise to questions about the origin of the “overlap” observed in the sequential production of various cell types; conceptually, this overlap could be produced by identical RPCs executing identical rigid specification programs if they begin to execute it along the spatiotemporal gradient noted above, as originally suggested by Cepko et al. [CAY<sup>+</sup>96]. However, it is impossible to reconcile this notion with more recent results from Harris’ *in vivo* zebrafish lineage tracing studies [DPCH03, HZA<sup>+</sup>12, BRD<sup>+</sup>15], which confirm mammalian cell culture work in demonstrating that vertebrate RPC lineages do not execute similar specificative programs. More recent articulations of Cepko’s concept of linear specification programs incorporate variable sub-programs to account for this [Cep14].

### 1.3.3 Other morphogenetic phenomena

The outcomes of RPC proliferation and lineage commitment are often taken as sufficient explanation for the formation of the neural retina. Indeed, after the eye cup has been formed (prior to the specification of any neurons), RPCs are, in effect, already “in place”. Therefore, there are few documented RPC phenomena outside of the “proliferation” and “specification” categories of the morphogenetic alphabet [Lar92], which enumerates the cellular processes contributing to tissue form and structure. RPCs do not seem inclined to migrate long distances, they seem not to generate much in the way of extracellular material<sup>8</sup>, and, in non-pathological conditions, they are rarely seen to die. Still, there are a number of phenomena which appear to be important for proper retinal organisation that fall into these other “alphabet” categories.

The most notable of these is interkinetic nuclear migration (INM), in which RPC nuclei move back and forth between the apical and basal surfaces of the retina. Common in many neural tissues, INM affects both proliferation and fate specification, but is dissociable from them- both cell cycle and specification proceed (albeit in a precocious manner) when INM is disrupted [MZLF02]. The apical retinal surface provides a microenvironment which appears to be required for RPC mitosis to occur<sup>9</sup>, and INM seems to consist of a directed, probably actomyosin-mediated movement to the apical surface, followed by an undirected “random walk” away from the apical surface and hence toward the basal retina [NYLH09]. This “random walk” may arise from displacements due to the directed INM of neighbouring progenitors [AHW<sup>+</sup>20]. More committed RPCs also appear to actively migrate to positions appropriate for the specified cell type [CAR<sup>+</sup>15, IKRN16], although it is unclear if this short-range migration is caused by INM undergone by more actively proliferating progenitors. It seems likely that these short-range migrations of more specified cells are especially significant in the early retina, before the neural plexiform

---

RPCs occasionally giving rise to “late-born” photoreceptors subsequent to their first division [WR09], giving lie to the notion that the process is particularly strict.

<sup>7</sup>In many studies, cell cycle exit is conflated with specification such that evidence of the former is taken as evidence of the latter. There are, however, reasons to believe cell cycle exit and specification are not the same process, discussed below.

<sup>8</sup>The formation of a laminin-rich basement membrane seems to be necessary for optic vesicle formation [ICW13], but it is not clear that RPCs produce this. ECM function in eye formation remains under-studied.

<sup>9</sup>Nonapical divisions do occur, notably in specified, but proliferative cells [GWC<sup>+</sup>07]. The extent to which RPCs depend on the apical surface may depend on how “RPC” is defined. In general, RPCs are no longer considered as such when they acquire characters associated with differentiated neurons (cell type markers, morphological traits, etc.), although it is clear that the acquisition of these characters does not necessarily imply that the cell is postmitotic. Any complete explanation of how RPCs give rise to the structure of the eye must also consider these nonapical divisions.

layers (consisting of axons and other neural processes) have begun to divide the cellular layers into bounded compartments.

Notable gaps in the study of the RPC “morphogenetic alphabet” include the regulation of cell size and growth, and potential roles for cell death, both of which have received little attention. Cell size and growth are closely linked to proliferative behaviour in yeast [YDH<sup>+</sup>11], and studies have been conducted in Schwann cells and oligodendrocytes [CDMR01], but related effects in RPCs have not been elucidated. This may be a significant oversight, given the requirement for RPCs to continuously grow during their proliferative lifespan. Cell death does not seem to play the same “pruning” role for RPCs that it often does in other neural tissues, and observed rates of cell death in normal RPC populations are very low, so few studies have been conducted.

## 1.4 Macromolecular mechanistic explanations for RPC phenomena

Having surveyed the cellular phenomena pertaining to RPC function in retinal development, we proceed to a selection of the many macromolecular mechanistic explanations (MEx) [Fag12] that have been offered to explain them. Unsurprisingly, given the field’s focus on the early events of eye development, these MEx are mainly targeted at this period. Thus, the majority explain tissue-level phenomena like the initial “wave” of cell cycle exit and specification, without necessarily seeking a global explanation for RPC behaviour irrespective of context, so it is unknown how many of these might pertain to ongoing peripheral neurogenesis or other, adult neurogenic phenomena like those exhibited by Müller glia. That said, we now turn to examine some of the best-developed of these explanations.

### 1.4.1 Transcription factor networks

Perhaps the most notable MEx offered to explain RPC specification and development is the eye field transcription factor network, or EFTFN. The roots of this explanation are found in the Pax6 “master gene” explanation popularised in the 1990s [Geh96]. This explanation revolved around the apparently universal involvement of Pax6 gene products in eye formation in model organisms, and the promiscuous inter-species effects of Pax6 (with mouse RNA able to induce ectopic formation of eye structures in *Drosophila* imaginal discs, for instance [HCG95]), so that it appeared to be a highly conserved genetic “switch” for eye development.

Importantly, this explanation purported to resolve what Darwin regarded as a serious problem for his theory, the apparent implausibility of the gradual evolution of eyes (and other “organs of extreme perfection”) from some primitive ancestral structure [Dar88, p.143-4]. In particular, Pax6 suggested to some theorists an alternative to the surprising hypothesis of Mayr and Salvini-Plawen, that differences in eye structure and function across clades indicate the independent appearance of eyes in more than 40 clades [vM77]. Pax6 thus provided a molecular pointer to a potential common ancestor for all animal eyes [EHML09].

Subsequent investigations revealed that vertebrate Pax6 is a conserved member of a complex network of cross-activating and inhibiting transcription factors, including Pax6, Rx1, Six3, Six6, Lhx2, ET, and Tll [Zub03]. Members of this network tend to promote proliferation and suppress markers of differentiated neurons, and their loss commonly results in the failure to form the eye field at all [AH09]. The

expansion of this explanation to include other TFs in a network revealed significant differences between species [Wag07]- while the role of Pax6 is conserved between *Drosophila* and vertebrates, the roles of other members of the EFTFN are not. Moreover, the universality of Pax6 was only apparent, and not real, as there are bilaterian eyes whose development is Pax6 independent (including in *Platynereis*, *Branchiostoma*, and planarians) [Koz08]. This highlighted the great difficulty in connecting morphological characters such as those observed by Mayr with a genetic basis- it is simply not clear that Pax6 conservation points to a common ancestor for all eyes, or even all photoreceptive neurons<sup>10</sup>. Moreover, expansion of the moncausal “master gene” explanation, to include a network of TFs with broad gene regulatory effects, highlighted the problems of complexity in offering MEx for RPC function. The components of this network interact in complex, context-dependent ways. While the EFTFN as a whole is taken to promote RPC proliferation and to delay specification<sup>11</sup>, its components have also been held responsible for the specification of particular classes of differentiated neurons, and remain expressed in those postmitotic cells. Notably, Pax6 is implicated in the expression of bHLH TFs required to specify multiple classes of retinal neuron [MAA<sup>+</sup>01], and is known to directly activate Ath5, necessary for RGC specification [WSP<sup>+</sup>09]. The EFTFN has thus been offered as an explanation for the maintenance of the multipotent, proliferative RPC state, but how this network is disassembled, and its components repurposed to promote specification, remains obscure.

The EFTFN is not the only transcription factor network offered as a MEx for RPC function. Another well-developed explanation involves Chx10 (aka vsx2), a transcription factor important for normal proliferation of RPCs, its loss causing microphthalmia in the mouse [BNL<sup>+</sup>96]. Chx10 was subsequently found to repress Mitf, involved in RPE specification, and hence to promote neural retinal fates over pigmented epithelial ones [Hor04]; in the absence of Chx10 the early eye cup does not stratify properly between apical pigmented cells and the neural retina. Much like the multifunctional EFTFN components, Chx10/vsx2 has also been implicated in the specification of particular neural fates, notably bipolar neurons [BNL<sup>+</sup>96] and the regulation of Vsx1 (a parologue of Chx10), Foxn4, and Ath5, associated with specification of subpopulations of bipolar cells, horizontal and amacrine cells, and RGCs and PRs, respectively [CYV<sup>+</sup>08, VJM<sup>+</sup>09].

Clear hypotheses advocating for particular relationships between different TF-centric explanations are rarely stated. It is tempting simply to arrange them in some kind of “developmental order”, perhaps with the Chx10-Mitf network “downstream” of the EFTFN. That this would be facile is evident from the changing roles of these transcription factors, depending on developmental and cellular context. To date, no unifying framework has been applied. Obvious candidates include the “developmental gene regulatory network” concept [LD09], a type of cybernetic explanation which assembles genes into feedback networks. Given the popularity of this type of explanation, it is worth noting that no one has yet had any success in offering one for RPC function.

These transcription factor network MEx frequently incorporate extracellular signals (often as an explanation for the appearance or “set-up” of the TF network), and it is generally recognised that these signals have a profound influence on these networks, and on RPC behaviour generally. We therefore turn to explanations invoking these signalling mechanisms.

---

<sup>10</sup>Indeed, the relevant “unit” of homology for evolutionary explanations for eyes remains contested, with some arguing for the cell itself over any particular set of gene sequences [EHML09]

<sup>11</sup>This is sometimes referred to as “promoting RPC fate”, since RPCs are taken to be those cells which proliferate but do not yet display markers of specification. Since it is, by now, widely recognised that cells that appear to be well-specified may remain in cell cycle [GWC<sup>+</sup>07, ESY<sup>+</sup>17] this terminology should probably be jettisoned.

### 1.4.2 Intercellular signalling networks

Virtually every developmentally significant class of signal had been implicated in RPC function by 2009 [AH09]. These include BMP, CNTF, FGF, Glucagon, Hedgehog, IGF, Notch, TGF $\alpha$ , TGF $\beta$ , VEGF, wnt, and a host of neurotransmitters. This diversity of signalling pathways has proved to be a formidable problem for integrated explanations, since almost all of these pathways converge on the same two cellular outcomes in RPCs, that is, proliferation and specification. Thus, most signalling MEx for RPC function elide the majority of other signals which are known, or thought, to affect the same processes. That said, let us explore a few of the more detailed signalling explanations.

In developmental terms, the first phenomenon requiring explanation is the appearance of the eye field to begin with- what is it that accounts for the differentiation of RPCs from the rest of the anterior neural plate and tube? Wnt signalling MEx have been offered to explain the appearance of the *Xenopus* eye field. Fz3 signalling seems to promote expression of eye field transcription factors (see below) [RDT<sup>+</sup>01], while an unspecified non-canonical interaction between Wnt11 and Fz5 inhibits canonical  $\beta$ -catenin signalling through Wnt8b/Fz8a, which would otherwise promote prospective anterior forebrain fates [CCY<sup>+</sup>05]. Inhibition of FGFR2 signalling, and activation of ephrinB1 signalling have also been implicated in early *Xenopus* eye field cell movements [MMDM04]. Subsequent experiments determined that the xenopus ADP signalling through the P2Y1 receptor directly activates the EFTFN [MBE<sup>+</sup>07], described above. More recent experiments in zebrafish suggest that precocious acquisition of neuroepithelial apicobasal polarity, probably driven by interactions with a Laminin1 basement membrane, distinguishes the early eye field [ICW13].

Subsequent to the appearance of eye field RPCs and their rearrangement into the optic cup, the apparent central-to-peripheral “wave” of RPC exit from cell cycle and specification of early RGCs [HE99], has had detailed MEx advanced to explain it. In both zebrafish and chick retina, FGF3 and FGF8, originating from the optic stalk, initiate this early cell cycle exit and specification [MDBN<sup>+</sup>05], while inhibiting FGF signalling prevents this from occurring, and ectopic expression of FGF can cause it to occur inappropriately. The progression of this “wave” of cell cycle exit and specification has been separately explained, by Sonic Hedgehog (Shh) signalling from the newly specified RPCs inducing cell cycle exit and specification in adjacent cells [Neu00]. This process is dependent on, and downstream of, the above-mentioned FGF induction [MDBN<sup>+</sup>05]. The role of Hh signalling has been challenged on the basis that Hh inhibition in subsequent experiments did not display the same effect size [SF03], and that its effects on Ath5 expression, required for RGC specification, are ambiguous [AH09]. More recent MEx have suggested that Hh signals may decrease the length of the cell cycle, resulting in increased proliferation and earlier cell cycle exit and specification [LAA<sup>+</sup>06, ALHP07].

A well-developed “local” signalling MEx invokes the classic Notch/Delta lateral inhibition model, with small fluctuations in Notch/Delta activity giving rise to a positive feedback response that differentiates neighbouring cells. Cells which have high Delta expression tend to be specified as retinal neurons, while those with high Notch tend to remain proliferative, either as RPCs or Müller glia [DRH95, DCRH97]. More recent results implicate asymmetric inheritance of Sara-positive endosomes in the partitioning of this signal [NRN20]. Such a mechanism could regulate the activity of both early, central RPCs, as well as peripheral RPCs, and may contribute to inter-RPC variability. These differences between RPCs located in different parts of the developing retina have been of significant interest, and it is worth briefly examining patterning MEx that may also explain spatial differentiation between RPCs.

### 1.4.3 Patterning mechanisms

Among the most interesting features of RPCs is that they reliably give rise to specified neurons, in particular RGCs, that seem to “know where they are” in the retina, enabling them to wire their axons in correct retinotopic order in the superior colliculus (SC) or optic tectum (OT). The most developed MEx explaining this refer to gradients of EphA and EphB receptors expressed in RGCs, and their respective ephrin ligands expressed in the SC or OT. In the retinal RGC population, an increasing nasotemporal gradient of EphA is paired with an increasing dorsoventral gradient of EphB. A corresponding increasing rostrocaudal gradient of ephrin-A is paired with an increasing lateromedial gradient of ephrin B in the SC/OT. This allows for a two-axis encoding of an RGCs’ position in the retina [TK06]. As the RGCs’ axon pathfinding depends on repulsive effects mediated by Eph receptors, this code is sufficient to allow correct wiring of even single RGCs [GNB08]. The action of Gdf6a seems to establish this code in RPCs themselves, prior to specification [FEF<sup>+</sup>09].

Indeed, there are numerous similar observations of expression gradients that create spatial differences between RPCs themselves. Most relevant to the proliferation dynamics highlighted in this chapter is the observation that, in *Xenopus* eyes, a decreasing dorsoventral gradient of type III deiodinase renders the cells of the dorsal CMZ refractory to thyroid hormone (as the deiodinase inactivates TH) [MHR<sup>+</sup>99]. The effect of this is to set up a differential response to TH in post-metamorphic RPCs, so that the ventral population selectively expands in response to TH [BJ79].

These patterning mechanisms are of particular interest here, in large part because they clearly establish that the RPC population is heterogenous, both with respect to proliferative and specificative behaviours, and perhaps others as well. This is of critical importance for any modelling effort, as virtually all mathematical models used by stem cell biologists (and those used to justify Harris’ SMME) assume, at least initially, homogenous populations of stem or progenitor cells. Since RPCs do not meet this condition, special care is needed to use these models.

### 1.4.4 Chromatin dynamics

In recent years, the great importance of chromatin conformation in RPC proliferation and specification has become more clear. Indeed, chromatin dynamics are now widely invoked in explaining stem and progenitor cell behaviour, and suggested as a target for cell reprogramming [Kon06, TR14]. In RPCs, detailed accounts of three-dimensional chromatin dynamics have yet to appear. However, a number of studies point to the importance of chromatin state in informing the overall cellular state. In particular, histone deacetylation seems to be important for RPC specification, as the loss of histone deacetylase 1 (HDAC1) in zebrafish results in overproliferation and decreased specification, and correlated increases in Wnt and Notch activity [Yam05]. In mouse retinal explants, decreased proliferation and specification result from pharmacological inhibition of HDAC [CC07]. Additionally, the chromatin remodelling complex SWI/SNF has repeatedly been implicated in RPC function. Notably, one particular component of this complex seems to be particularly associated with vertebrate RPCs (BAF60c, an accessory subunit) [LHKR08]. A switch to other subunits seems to be necessary for specification [LWR<sup>+</sup>07]. Details regarding the subunits involved in specification and their downstream effects are complex and context dependent, much like the signalling pathways mentioned above.

## 1.5 A unified theory of RPC function? “Blurring” to order

From the foregoing discussion, we can see the theoretical conundrum. Macromolecular explanations for RPC behaviours, like those throughout the molecular biological tradition, have generally been built outwards from particular transcription factors, receptors, etc. The result is an archipelago of MEx, at best connected by tenuous speculation, and in most cases, without any known means to form an integrated model. Furthermore, the degree of complexity and context-dependence evident from the literature might seem to preclude such a model. As we have seen, multiple reviews found that the evidence did not allow for clear discrimination between logically distinct types of mechanisms for producing the observed variability in RPC lineage outcomes.

In this situation, there were two theoretical options. The first is simply to “crank the handle”- to generate more and more facts describing the difference particular molecules make to RPC outcomes in dozens of relevant contexts, piling up exceptions and idiosyncrasies, in the hope that doing so will eventually bridge the explanatory “islands” of the MEx archipelago. This has been referred as the enumerationist approach by Kaneko, who ably explains why it is doomed to failure [Kan06, pp.31-32]. The second option, the one actually chosen by Harris, is more theoretically sophisticated. As Nicholas Rescher has noted, regarding in-principle limits to scientific knowledge, the phenomenal universe has infinite descriptive complexity- one can always add more detail to a description of some phenomenon, and no such description is ever complete [Res00, p.22-9]. Moreover, “even as the introduction of greater detail can dissolve order, so the neglect of detail can generate it.” [Res00, p.62] As Rescher goes on to comment:

[W]e realize that in making the shift to greater detail we may well lose information that was, in its own way, adequate enough ... information at the grosser level may well be lost when we shift to the more sophisticated level of greater fine-grained detail. The ‘advance’ achieved in the wake of ‘superior’ knowledge can be - and often is - purchased only at a substantial cognitive loss.

...

It is tempting on first thought to accept the idea that we secure more - and indeed more useful and more reliable - information by examining matters in greater precision and detail. And this is often so. But the reality is that this is not necessarily the case. It is entirely possible that the sort of information we need or want is available at our ‘natural’ level of operation but comes to be dissolved in the wake of greater sophistication. [Res00, p.65-6]

Rescher’s greater point is that “blurring” detail, at levels below the phenomenal one under consideration (for RPCs, generally, the cell or lineage), may be necessary to produce an ordered explanation that is useful for some objective. An example of this practice in macromolecular modelling is coarse-graining, where irrelevant atomic detail is abstracted in favour of more computationally tractable approximations [ILU+14]. “Blurring” in this fashion also enhances the cognitive value of models by keeping them “mind-sized”; at our natural level of operation, as Rescher has it. Given the number of overlapping and contradictory MEx for RPC behaviour, we have the situation Rescher is describing- more sophistication, and more detail, has dissolved order, not revealed it<sup>12</sup>. The inability to assemble an unified explanation for RPC function has left us without fundamental understanding of how highly ordered neural tissues

---

<sup>12</sup>At least part of this problem is likely related to the fact that the majority of biomedical findings cannot be replicated [Ioa05]. The finding, mentioned above, that Shh effect sizes on RPC function were not as large as initially reported when subsequently investigated, is typical and symptomatic of this replication problem. “Blurring” may therefore be necessary not only because of fundamental epistemic limits, but also because it is often difficult to distinguish bona fide results and explanations from spurious ones.

like eyes are generated from composites of units with highly variable, temporally and spatially ordered outcomes like RPC lineages. As this is a common feature of vertebrate neurogenesis more generally, this leaves us without the ability to produce complete models of neurodevelopmental processes in many species. Moreover, in the absence of a clear framework for comparing the explanatory power of the diverse array of MEx so far advanced, practical contributions of clinical relevance have been scanty and tentative, with RPC transplantation, and more recently, gene therapies taking little note of complex MEx for RPC function [CAI<sup>+</sup>04, GS07, YQW<sup>+</sup>18].

In a situation of this kind, this type of “blurring” is required, and it seems that by cutting down to the simplest possible explanation, the SMME hopes to bring into view order that was previously obscured by detail. There is, of course, a significant danger here: how does one decide what is “blurred out” and what remains? We can easily understand how a practitioner’s biases could lead to a sort of relativism, where the “blurring” makes apparent a spurious order that conforms to these biases rather than to reality as such. With this in mind, let us survey the general thrust of the SMME, before proceeding to examine it in detail, in Chapter 2.

## 1.6 Explanatory Strategy and Intent of the SMME

As we have seen in Section 1.2, Harris’ long-held understanding of the explanatory options for RPC function divides them into four broad categories:

1. A linear algorithmic “program” of proliferation and specification
2. Asymmetric segregation of specificative determinants
3. “Stochastic processes” internal to the cells
4. Influences of extracellular factors

Harris’ sophisticated discussions of RPC MEx rarely treat these categories as exclusive, and concede that good explanations for RPC behaviour may involve phenomena from more than one of them. Indeed, the SMME necessarily contains elements that Harris concedes are “linear” and “deterministic” [HZA<sup>+</sup>12]. Still, his overall strategy for the SMME is, first, to substantiate the predominant influence of one of these categories of phenomena (that is, category 3, internal stochastic processes or effects), and subsequently to specify an actual macromolecular system that could plausibly be such an “internal stochastic process”. These two theoretical maneuvers, while tightly linked, serve different purposes within Harris’ overall explanatory framework, which must be examined separately.

The SMME for zebrafish RPC function has been developed across three separate papers [HZA<sup>+</sup>12, BRD<sup>+</sup>15, WAR<sup>+</sup>16]. Each builds on the earlier publications, collectively purporting to explain the behaviour of RPCs wherever, and whenever, they may be found in the zebrafish eye. The underlying model is originally derived from an earlier paper pertaining to rat RPCs [GZC<sup>+</sup>11]. He et al. [HZA<sup>+</sup>12] and Wan et al. [WAR<sup>+</sup>16] use essentially the same model and make up the substance of the first of these two maneuvers. Boije et al. [BRD<sup>+</sup>15] substantially modifies this model, specifying the activity of two known transcription factors (Ath5 and Ptf1a) as the model’s biological referents. This paper constitutes the second theoretical thrust.

The first maneuver intends to support the contention that zebrafish RPCs are equipotent progenitors with variable lineage outcomes that depend on independent “stochastic” processes within each of these

cells. This is to be provided by demonstrating that a Simple Stochastic Model (SSM) of an RPC, numerically simulated many times by Monte Carlo methods to represent a population of RPCs, produces similar outcomes to populations of RPCs *in vivo*. This explanatory strategy is common in the stem cell literature, the original example having been published in 1964 by Till, McCulloch, and Siminovitch [TMS64]. The SMME therefore represents an example of a traditional scientific logic- an explanatory pattern deployed by stem cell biologists in diverse contexts, and widely accepted because of its ongoing use in the literature, although with varying interpretations.

The success of this first maneuver thus depends on two outcomes. Firstly, the output of the SSM should accurately reflect the observed proliferative and specificative outcomes of zebrafish RPC lineages, giving weight to Harris' claim that it "provides a complete quantitative description of the generation of a CNS structure in a vertebrate *in vivo*" [HZA<sup>+</sup>12]<sup>13</sup> Secondly, the internal structure of the SSM should provide good reason to believe that one of the stochastic options is a better explanation for RPC lineage outcomes than those identified by the other three categories of theoretical options enumerated above.

The second theoretical maneuver is the specification of particular biological referents for entities in the model. This offers an opportunity to move beyond a purely conceptual argument about the kind of process that might produce variable RPC lineage outcomes, and to begin the work of explaining how the behaviour of a particular macromolecular system constitutes such a process, so that empirically verifiable hypotheses may be generated. The success of this maneuver depends on the biological plausibility of the identification between model structure and the biological function of transcription factors, *Ath5* and *Ptf1a*, that the model names. A good SSM-based explanation would point the way for further research by identifying *how* so-called "stochastic processes" in RPCs might function, and make some predictions about this. In order to address these issues, we use model comparison, which is introduced and detailed in Section 15.1. With that said, let us turn to the SMME and determine how well these manuevers have succeeded.

---

<sup>13</sup>This claim is somewhat extravagant; the SSM, by definition, includes no spatial information, so it is unclear how one could be a "complete description" of any spatially organised structure. Still, it can be complete with regard to cell population numbers.

# Chapter 2

## “Stochastic mitotic mode” models do not explain zebrafish retinal progenitor lineage outcomes

### Synopsis

(1) *The SMME is composed of Simple Stochastic Models, intended to model zebrafish RPC lineage outcomes, and purports to explain these outcomes with a “stochastic process” mechanism.* (2) *SMME models diverge from the explanatory rationale of their immediate forebears, introducing an unexplained structure of temporal phases, and become increasingly dominated by the mitotic mode model-element.* (3) *Performing an AIC-based model comparison demonstrates that an alternative model with a deterministic mitotic mode performs better than the central He et al. model.* (4) *The suggestion of Wan et al., that the He et al. model also explains postembryonic, CMZ-driven growth, is tested by simulation, and cannot be reconciled with observations.* (5) *These analyses suggest that the source of variability in RPC lineage outcomes cannot be identified by the construction of SSMs, and that older theories invoking phases of RPC behaviour should be rehabilitated.*

### 2.1 Introduction

Mechanistic explanations (MEx) derive their utility from the resemblance of the conceptual mechanism’s output to empirically observed outcomes. As maps to biological territories, biological MEx are habitually identified with the living systems they represent. Most well-developed MEx take the form of a model, whose internal structure is taken to reflect the underlying causal structure of a biological phenomenon. The nature of the causal relationship between a mechanistic model and the phenomenon it purports to explain remains a topic of active discussion in the philosophy of biology [Fag15]. Biologists, nonetheless, usually accept that a model which explains empirical observations well (usually measured by statistical or information theoretic methods), and reliably predicts the results of interventions, bears a meaningful structural resemblance to the actual causal process giving rise to the modelled phenomenon.

Biological phenomena are notable for exhibiting both complex order and unpredictable variability. A

significant challenge for MEx in multicellular systems is to explain how complex, highly ordered tissues, like those produced by neural progenitors, can arise from unpredictably variable cellular outcomes. Stem cell biologists have often used to Simple Stochastic Models (SSMs) in order to explain the observed unpredictable variability in clonal outcomes of putative stem cells [Fag13]. Because SSMs are susceptible to Monte Carlo numerical analysis as Galton-Watson branching processes, they have been convenient explanatory devices, appearing in the literature for more than half a century [TMS64]. By specifying the probability distributions of symmetric proliferative (PP), symmetric postmitotic (DD), and asymmetric proliferative/postmitotic (PD) mitotic modes, SSMs allow cell lineage outcomes to be simulated.

SSMs are “stochastic” insofar as they incorporate parametric random variables. As Jaynes has noted, “[b]elief ... that the property of being ‘stochastic’ rather than ‘deterministic’ is a real physical property of a process, that exists independently of human information, is [an] example of the mind projection fallacy: attributing one’s own ignorance to Nature instead.” [JBE03, pp.506] Despite this, macromolecular processes are often described as “stochastic” in the stem cell literature. Generally speaking, the behaviour of an SSM’s random variable is taken to represent sequences of outcomes that are produced by multiple, causally independent events. Recently, the influence of “transcriptional noise” on progenitor specification has been identified as a candidate macromolecular process that may produce unpredictable variability in cellular fate specification. Therefore, one explanatory strategy for stem and progenitor cell function compares SSM model output to observed lineage outcomes, in order to argue that “noisy”, causally independent events give rise to the proliferative and specificative outcomes of progenitor lineages.

In this report, we evaluate the best-developed of these explanations, dubbing it the “Stochastic Mitotic Mode Explanation” (SMME) for zebrafish retinal progenitor cell (RPC) function. The SMME is noteworthy because it claims: (1) to ”provid[e] a complete quantitative description of the generation of a CNS structure in a vertebrate *in vivo*” [HZA<sup>+</sup>12]; (2) to have established the functional equivalency of embryonic RPCs and their descendants in the postembryonic circumferential marginal zone (CMZ) [WAR<sup>+</sup>16]; and (3) to have established the involvement of causally independent transcription factor signals in the production of unpredictably variable RPC lineage outcomes [BRD<sup>+</sup>15]. Moreover, the SMME is taken to supply evidence for the predominance of stochastic effects over other explanations for RPC lineage outcomes, such as the classical explanation of a temporal succession of competency states [TR86]. These would be significant achievements with important consequences for both our fundamental understanding of CNS tissue morphogenesis and for retinal regenerative medicine. However, these models were not subjected to model optimization or selection procedures, as advocated by model selection theorists [BA02], and widely adopted in ecology and evolutionary biology [JO04].

In order to evaluate of the two SSMs which form the SMME’s MEx for RPC function, we have re-expressed the models as cellular agent simulations conducted using the open source C++-based CHASTE cell simulation framework [MAB<sup>+</sup>13]. We explored the structure of the SSMs, dubbed the He and Boije SSM respectively, compared to their explanatory forebear, dubbed the Gomes SSM [GZC<sup>+</sup>11]. This analysis suggested the progression of temporal “phases” in the models was largely responsible for the SMME model fits to data. In order to investigate this possibility, we built an alternative model with a deterministic mitotic mode and compared its output to the He SSM. We found that the deterministic alternative model was a better explanation for the observations, demonstrating that SMME fails when compared to alternatives. We therefore suggest that an explanatory approach based on the use of SSMs is incapable of distinguishing between theoretical alternatives for the causal structure of RPC lineage be-

haviours. Furthermore, by comparing the output of the models with novel postembryonic measurements of proliferative activity, we find that the SMME explanation cannot account for quantitative majority of retinal growth in the zebrafish, driven by CMZ activity postembryonically. Finally, we discuss the place of SSMs, the concept of “mitotic mode”, and the role of “noise” in explaining RPC behaviour, and suggest ways to avoid the modelling pitfalls exemplified by the SMME.

## 2.2 Metatheoretical analysis

The SMME for zebrafish RPC function has been advanced using two SSMs. One first appears in He et al., and again, unmodified, in Wan et al. [HZA<sup>+</sup>12, WAR<sup>+</sup>16]; it explains lineage population statistics and time-dependent rates of the three generically construed mitotic modes (that is, PP, PD, DD). We have called this the He SSM. The second appears in Boije et al. [BRD<sup>+</sup>15]; its intent is both to introduce the role of specified macromolecules into the mitotic mode process, and to explain neuronal fate specification in terms of the process. We have called this the Boije SSM. The He model is directly descended an SSM advanced to investigate causally independent fate specification in late embryonic rat RPCs, formulated in Gomes et al. [GZC<sup>+</sup>11]. The Boije SSM differs substantially from the He and Gomes models, but inherits its general structure from the He SSM.

The metascientific analysis of the development of biological explanations remains undertheorized. Perhaps most refined tool for global evaluation of biological theories (Schaffner’s ”Extended Theories” [Sch93]), treats biological explanations as hierarchically organised logical structures, similar to the method of Imre Lakatos [Lak76], and proposes the use of Bayesian logic to distinguish between them. However, biologists rarely offer explanations in this form; we rather prefer mechanisms, expressed in diagrammatic form or as mathematical model-objects.

We adopt Fagan’s view [Fag15], that MEx for the behaviour of stem and progenitor cells consist of assemblages (“mechanisms”) of explanatory components which are understood to be causally organised by virtue of their intermeshing properties. While Fagan treats SSMs separately from macromolecular MEx, and we find that the Gomes SSM was not deployed in this role, the He and Boije SSMs were used as explanations for the behaviour of RPCs. As Feyerabend famously observed, scientists operate as epistemological anarchists; the development of our explanatory logic is not bound by a set of rules, but rather arises organically from our scientific objectives, extrascientific context, and so on [Fey93].

We have therefore chosen to examine the structure of the Gomes, He, and Boije SSMs arranged in chronological order, to highlight how the explanatory logic of zebrafish SMME SSMs differs from their immediate ancestor, and from other uses of SSMs in stem cell biology. We have diagrammatically presented these SSMs as MEx, consisting of components describing the proliferative and fate specification behaviour of cellular agents. The proliferative and specificative components of the MEx are causally organised by their intermeshing property, mitotic mode. We have used abbreviations to denote important classes of model components and inputs, informed by the emphasis of Feyerabend on the persuasive role of metaphysical ingredients and auxiliary scientific material; these are as follows:

- MI - Model ingredient, making reference to some conceptual or metaphysical construct
- AS - Auxiliary scientific content
- RV - Random variable

- PM - Parameter measurement, model parameter set by measurements, independently of model output considerations
- PF - Parameter fit, model parameter set without reference to measurement, in order to produce model output agreement with observations

We draw the reader’s attention to the expansion of the “mitotic mode” intermeshing property in later SSMs; this explanatory component comes to dominate the later models, which makes its interpretation critical to the success of the mechanistic explanation in meaningfully representing a real biological process.

Numerous explanations for variability in RPC lineage outcomes have been considered. Harris has argued that these belong to three cell-autonomous categories of process, in addition to extracellular influences: (1) a linear temporal progression of competency states, (2) asymmetric segregation of determinants during mitoses, and (3) intracellular “stochastic events” [HBEH88, AH09]. All of the SSMs are used to argue for the predominant influence of the third type of process in RPC lineage outcomes, essentially by demonstrating that the output of the SSM resembles observations.

### 2.2.1 Gomes SSM: Ancestral Model of the SMME SSMs

The Gomes SSM is presented in Fig 2.1. The model’s structure is straightforward; there are three independent random variables, drawing from an empirically-derived Log-Normal probability distribution for the time each cell takes to divide, and categorical distributions on the mitotic mode of the division and the specified neural fate of any postmitotic progeny. The random mitotic mode variable functions as the “intermeshing property” linking cycle behaviour to fate specification. In this scenario, the processes governing proliferation, leading to cell cycle exit, and governing cell fate commitment are causally independent of each other and of their history. The objective of the Gomes et al. study is to compare the lineage outcomes of dozens of individual E20 rat RPCs, in clonal-density dissociated culture, with the model output, developing earlier work in this system [CBR03]. Although the model incorporates conventional proper time (clock-time), none of the RVs reference it to determine their values. The abstract “cells” represented by this model do not have any timer or any source of information about their relative lineage position. Fate specification is construed in terms of conventional histochemical markers of stable cell fates; only the neural types generated late in the retinal histogenetic order are represented, as these are the only neurons specified by E20 rat RPCs.

This SSM is explicitly built as a null hypothesis, or a model of background noise. It represents an extreme case in which all of the specificative and proliferative behaviours of an RPC are totally independent of all other RPCs and events in its clonal lineage. The stated purpose of this model is “to calibrate the data”, serving “as a benchmark” [GZC<sup>+</sup>11], a purely hypothetical apparatus to produce sequences of causally unrelated lineage outcomes. Substantial deviations of the observed data from the fully independent events of the SSM may then be interpreted as causal dependencies between RPC outcome and their relative lineage relationships, as might be observed in a developmental “program” or algorithmic process. Finding that, with some exceptions, observed proliferative and specificative outcomes generally fall within the plausible range of Gomes SSM output, Gomes et al. conclude that RPC lineage outcomes in late embryonic rat RPCs seem to be dominated by causally independent events, suggesting that causally isolated “noisy” macromolecular processes may give rise to the unpredictable variability in the fate outcomes of these cells.

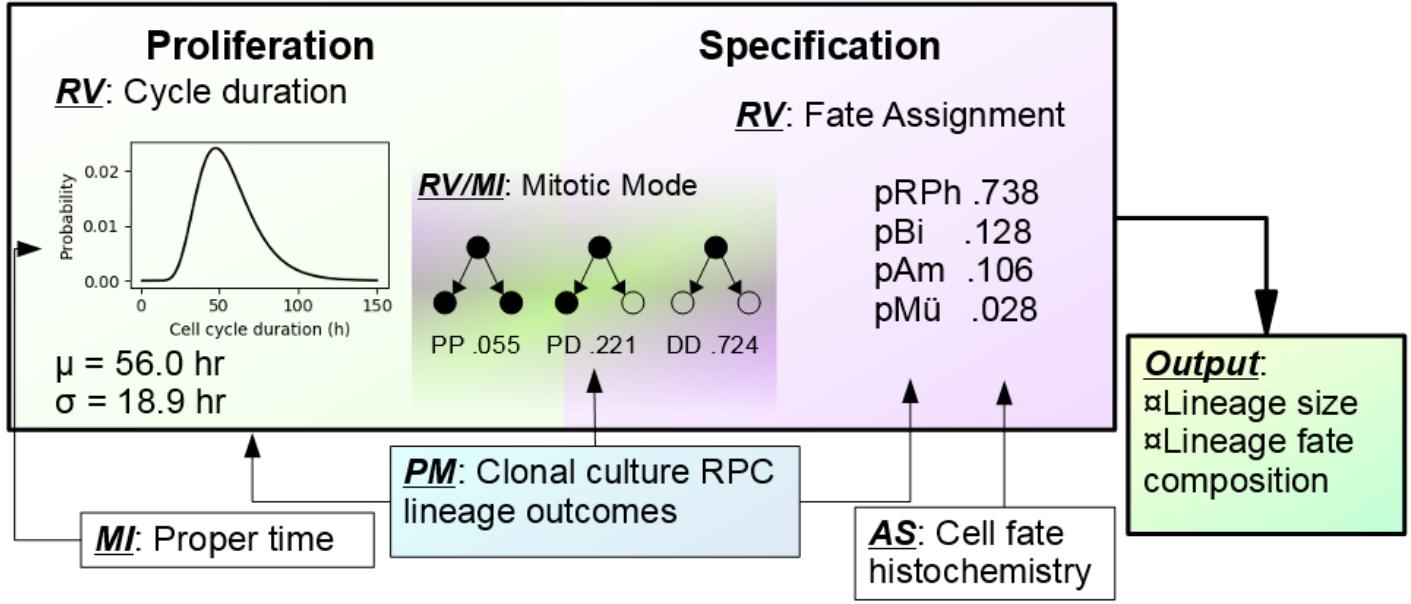


Figure 2.1: Structure of Gomes SSM

Structure of the Gomes SSM [GZC<sup>+</sup>11]. pRPh, probability of rod photoreceptor specification. pBi, probability of bipolar cell specification. pAm, probability of amacrine cell specification. pMü, probability of Müller glia specification.

### 2.2.2 He SSM: Explaining variability in zebrafish neural retina lineage size

The He SSM, shown in Fig 2.2, is deployed in a explanatory role rhetorically identical to the Gomes SSM. The extent to which model output “captures.. aspects of the data” is taken to obviate the need for explicit “causative hypothes[es]”. On this account, only residual error between model output and observations may be ascribed to non-stochastic processes like “histogene[tic ordering] of cell types or a signature of early fate specification”. That is, if the model can be fit to observations, this is taken to exclude the presence of any cell-autonomous temporal program, asymmetric segregation of fate determinants, extracellular influences, and the like.

There are fewer direct empirical inputs to the He model’s parameters than the Gomes SSM, limited to the duration of the cell cycle, which is modelled by a shifted Gamma distribution <sup>1</sup>. The close synchrony of zebrafish RPC divisions is modelled by assigning sister RPCs the same cycle length, shifted by a normal distribution with a variance of one hour, in contrast to Gomes RPCs, which are treated as fully independent. The lineage outcomes the He SSM is called on to explain differ significantly from those referred to by the Gomes SSM. Most notably, the Gomes SSM does not account for the early appearance of RGCs, which are not produced by the late E20 progenitors examined in that study. The zebrafish RPC lineages studied by He et al. produce all of the retinal neural types, including RGCs, which are typically produced by PD-type divisions. The He SSM does not model particular cell fates, supposing that mitotic mode is “decoupled” from fate specification. The mitotic mode RV linking cell cycle to fate specification in the Gomes SSM has thus subsumed the specification outcomes of RPC lineages entirely,

<sup>1</sup>The change of this distribution to Gamma in He et al. from Log-Normal in Gomes et al. is not explained, but presumably was felt to reflect the zebrafish context better.

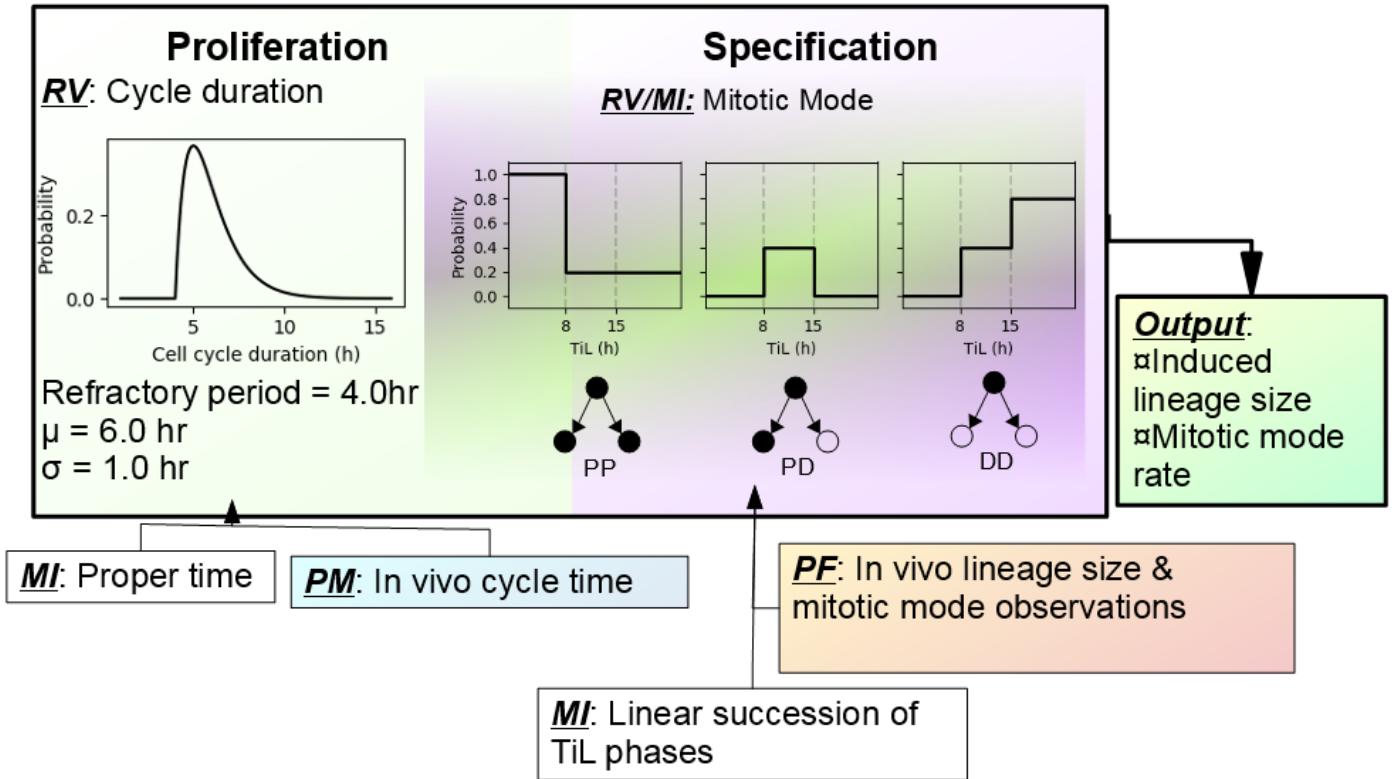


Figure 2.2: **Structure of He SSM**

Structure of the He SSM [HZA<sup>+</sup>12].  $\sigma$ :  $\text{sqrt}(\text{variance})$  of the location-shifted gamma-distributed cycle time. TiL, Time in Lineage.

and the model is concerned only to explain the sizes of lineages (marked by an inducible genetic marker at various times), and the observed progression of mitotic modes in these early RPCs.

The He model is, therefore, called on to explain the temporal structure of the proliferative behaviour of early zebrafish RPCs that dissociated late rat RPCs do not exhibit. A Gomes-type SSM, in which the RVs determining RPC behaviour are independent of any measure of time, cannot account for this temporal progression. In order to address this, He et al., assume a linear progression of three phases which cells in each lineage pass through, the timing of these phases being determined relative to the first division of the RPC lineage, called here “Time in Lineage” or TiL. The parameters of the mitotic mode RV are determined by these TiL phases. The temporal structure of the phases and their effect on the mitotic mode RV are selected to produce a model fit. While He et al. acknowledge that the model therefore represents a “combination of stochastic and programmatic decisions taken by a population of equipotent RPCs,” no test is performed to determine the relative contribution of the model’s stochastic vs. linear programmatic elements. Instead, the purportedly stochastic nature of mitotic mode determination is emphasized throughout the report.

### 2.2.3 Boije SSM: Explaining variability in zebrafish RPC fate outcomes

The second SMME model advanced to explain zebrafish RPC behaviour, the Boije SSM, is displayed in Fig 2.3. This model is primarily concerned with the lineage fate outcomes that the He SSM does not treat, while abandoning the explicit proper time of the He and Gomes SSMs in favour of abstract generation-counting. The mitotic mode model-ingredient now subsumes all RPC behaviours. Boije et al. make a laudable effort to specify the particular macromolecules ostensibly involved in determining mitotic mode, nominating the transcription factors (TFs) Atoh7, known to be involved in RGC specification, and Ptf1a, known to be involved in the specification of amacrine and horizontal cells, as primary candidates. The contribution of vsx2 is taken to determine the balance between PP and DD divisions late in the lineage, with the latter resulting in the specification of bipolar or photoreceptor cells. The binary presence or absence of these signals is determined by independent RVs structured by the phase structure present in the He SSM, translated into generational time from proper time.

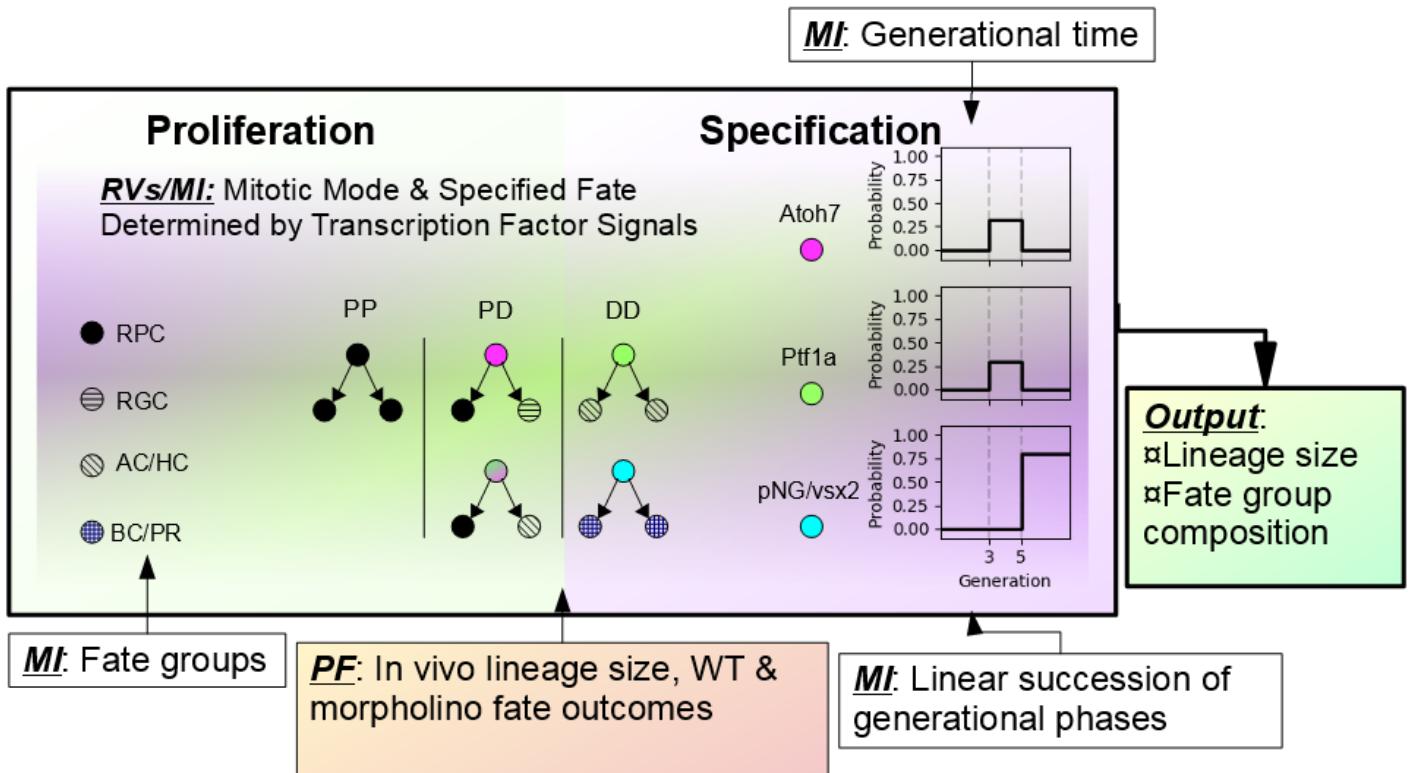


Figure 2.3: **Structure of Boije SSM**

Structure of the Boije SSM [BRD<sup>+</sup>15]. RPC, retinal progenitor cell. RGC, retinal ganglion cell. AC, amacrine cell. HC, horizontal cell. BC, bipolar cell. PR, photoreceptor. pNG- parameter representing contributions of Vsx1 and Vsx2 to specification of BC & PR fates in absence of Atoh7 or Ptf1a signals.

By this point, the SMME has become a very different type of explanation from its Gomes SSM forebear. We are no longer dealing with RVs that model causally and temporally independent processes for different aspects of RPC behaviour. There is, rather, one temporally dependent process, the determination of mitotic mode, which is explained by unpredictable subsets of each lineage generation

expressing particular TF signals. Where the Gomes SSM takes its parameters directly from empirical measurements of lineage outcomes, asking whether it is sufficient to assume that these are independently determined, the Boije SSM’s parameters are derived solely from model fit considerations. In spite of these considerable differences, the explanatory role of the SSM is effectively the same: the model’s fit to observations is taken as evidence of the predominant influence of stochastic processes determining mitotic mode on RPC behaviour. While Boije et al. acknowledge that whether some process is called “deterministic” or “stochastic” is “a matter of the level of description” [BRD<sup>+</sup>15] (i.e. is a property of the model-description and not of the physical process), the explanatory role of stochasticity for RPC lineage outcomes is emphasized throughout.

## 2.3 Model selection demonstrates the SMME is not the best available explanation for RPC lineage outcomes

It is notable that neither of the reports which use the He SSM [HZA<sup>+</sup>12, WAR<sup>+</sup>16], nor that using the Boije SSM [BRD<sup>+</sup>15] report their fitting procedures in detail, nor do any of the above report any statistical measures of model quality. Additionally, no models representing the alternative “theoretical options” are compared to those advanced as evidence for stochastic processes. Given that the He SSM and Boije SSM depart from the Gomes SSM by the addition of an unexplained temporal structure to the mitotic mode model ingredient, it is striking that the overall argument remains similar to Gomes et al.’s, despite the persuasive force of the latter deriving from the lack of such structures. While He et al. and Boije et al. acknowledge that their models involve both stochastic and linear programmatic elements, their relative influence on model output is not measured, and macromolecular explanation is only applied to the stochastic elements. Moreover, the emphasis on this mitotic mode model construct increases with each successive model, to the extent that in the Boije model there are no other elements that are used to explain RPC behaviours. All cellular behaviours are, in effect, progressively collapsed into the stochastic mitotic mode concept. Finally, the He and Boije SSMs contain more parameters, determined by fewer empirical measurements than the Gomes SSM. It is therefore important to test whether the most important ingredient in these models is the stochastic mitotic mode, against the possibility the unexplained temporal structure is the truly explanatory element.

To examine this, we compare the He SSM to an alternative model. To emulate the ad-hoc fitting procedure reported by He et al. [HZA<sup>+</sup>12], we fit the models to a training dataset, comprising the induced lineage size and mitotic mode rate data from He et al. We hold out a test dataset, consisting of the Atoh7 morpholino observations from He et al., and the CMZ lineage size data from Wan et al., which played no role in obtaining He et al.’s fitted model parameters. This allows us to test how well the models hold up under novel experimental conditions. Since the Boije SSM draws on the He SSM for its temporal phase-parameterisation and stochastic mitotic mode model ingredient, this analysis of the He SSM also bears directly on the validity of the Boije SSM.

As an alternative to the SMME He SSM, we constructed a model that has a deterministic mitotic mode with variable phase lengths. Rather than variability arising from a stochastic-process mitotic mode changing across phases of fixed length, we simply supposed that mitotic mode is deterministic in each phase (guaranteed PP mitoses in the first phase, PD in the second, and DD in the third), but the phase lengths are variable between lineages and shift slightly between sister cells. That is, we represented this linear progression of deterministic mitotic mode phases using the same type of statistical

construct the He SSM applies to model cell cycle length, to avoid introducing any novel or contentious elements into the model comparison. More specifically, each lineage has a first PP phase length drawn from a shifted gamma distribution, followed by a second phase length drawn from a standard gamma distribution. Upon mitosis, these phase lengths are shifted in sister cells by a normally distributed time period, exactly like cell cycle lengths in the He SSM. The effect of this alternative parameterisation on the mitotic mode RV is summarised in Figure 2.4.

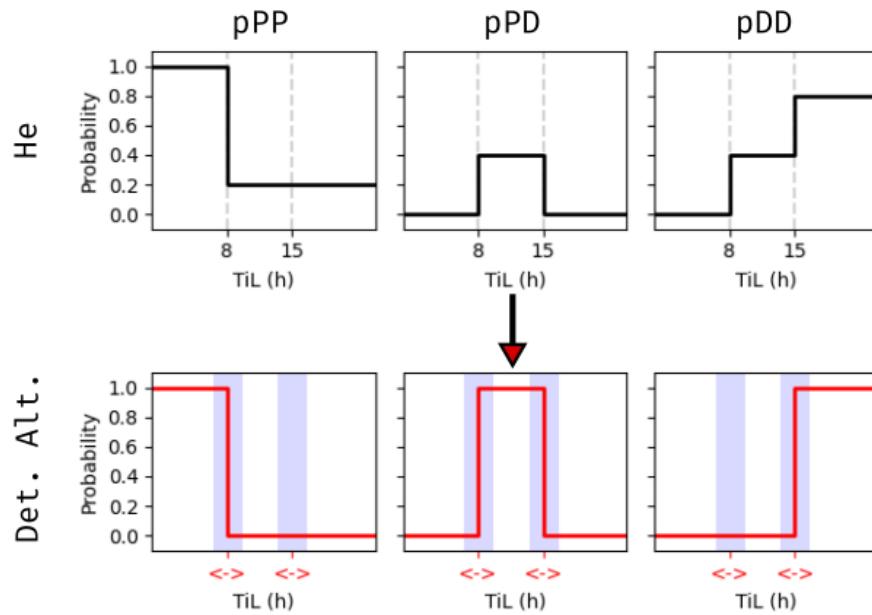


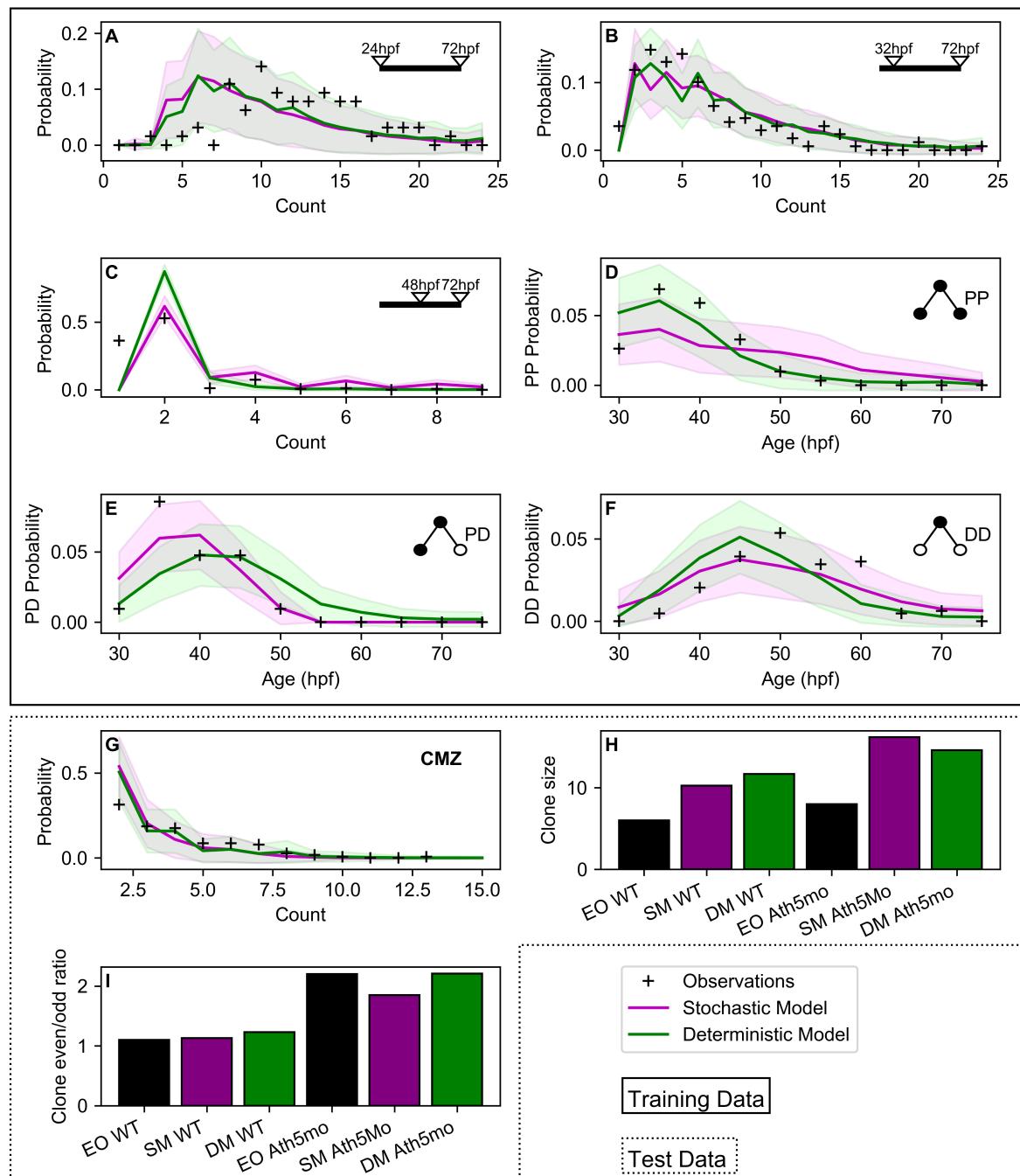
Figure 2.4: An alternative, deterministic mitotic mode model

Mitotic mode random variable in He et al.’s stochastic model (“He” panels, top) and a deterministic alternative (“Det. alt.” panels, bottom). Blue bands represent variable phase boundaries arising from gamma-distributed and sister-shifted phase boundaries.

This is a representation of the classic suggestion that RPCs step through a linear succession of competency phases. If this temporal program is governed by RPCs passing through a stereotypical series of chromatin configurations which allow for PP, then PD, then DD mitoses in turn, along with the associated competence to produce the particular cell fates associated with PD and DD mitoses, it seems plausible to suggest that RPCs differ in the lengths of time they occupy each state. Moreover, since these chromatin configurations must be broken down and rebuilt with each mitosis, the re-use of the “sister shift” model ingredient from the He SSM’s cell cycle RV is congenial, representing the same sort of cell-to-cell variability that results in the small differences between sister cells in cycle timing.

While the code used to implement the SMME models has not been published, the reports provide enough detail to reconstruct these models. We did this using the CHASTE cell-based simulation framework. Because the values selected for the He SSMs’ parameters were selected by hand, with only the value for the probability of PD-type mitoses in the second model phase being varied to produce the fit, we suspected that the fit would not be at or near the local minimum for a loss function. That is, a model fit produced in this manner is likely to be located in a region of the parameter space that is highly sensitive to small perturbations, and therefore may depend strongly on implementation-specific

idiosyncracies. He et al. report that they experienced difficulty in obtaining a good fit to their 32 hour induction data, with changes in the phase two PD probability producing large differences in fit quality. When we estimated the He SSM with the original fit parameterisation, we substantially reproduced the original fit, except for the 32 hour data, where the model output diverges substantially from that reported in He et al. (see S1 Fig). Since this is not the best fit available for the He SSM, and we wish to directly compare the best fits for the He SSM and our putative alternative model, we used the simultaneous perturbation stochastic approximation (SPSA) algorithm [Spa98] to optimise both models against observations. SPSA is convenient for complex, multi-phase models like the He SSM, because no knowledge of the relationship between the model's parameters and the loss function is required. We used Akaike's information criterion (AIC) as the loss function to be minimised, in order to provide a comparison metric between the two differently-parameterised models (the deterministic alternative has two fewer parameters than the He SSM).



**Figure 2.5: Model comparison: the SPSA-optimised He SSM and a deterministic alternative**

Empirical observations (black crosses and bars) and SPSA-optimised model output (magenta, stochastic mitotic mode model; green, deterministic mitotic mode model). Model output in panels A-G is displayed as mean  $\pm$  95% CI. Panels A-F: Training dataset, to which models were fit. Panels G-I: Test dataset, included in AIC calculation. Panels A-C: Probability of observing lineages of a particular size ("count") after inducing single RPC lineage founders at (A) 24, (B) 32, and (C) 48 hours with an indelible genetic marker, tallying their size at 72hpf. Panels D-F: Probability density of mitotic events of modes (D) PP, (E) PD, and (F) DD over the period of retinal development observed by He et al. Panel G: Probability of observing lineages of a particular size ("count") originating from the CMZ; RPCs are taken to be resident in the CMZ for 17 hours before being forced to differentiate, lineage founders are assumed to be evenly distributed in age across this 17 hour time period. Panel H: Average clonal lineage size of wild type and Ath5 morpholino-treated RPCs. Ath5mo treatment is taken to convert 80% of PD divisions, which occur in the second phase of the He model, to PP divisions, resulting in larger lineages. Panel I: Ratio of even to odd sized clones in wild type and Ath5 morpholino-treated RPCs. Methods in Section 11.1.7. Code in Section 17.1.15.

After fitting to the training dataset, we calculated AIC for the He SSM (hereafter SM, for stochastic model) and our deterministic mitotic mode alternative (hereafter DM), for both training and test datasets together. The optimized output for the two models is displayed in Fig 2.5, and presented separately in S2 Fig and S3 Fig. Remarkably, the DM closely recapitulates the output of the SM for both datasets. Moreover, the DM proves to be a better explanation of the overall dataset, as summarised in Table 2.1. Given this model selection scheme, we should choose the DM over the SM as an explanatory model of the data.

Table 2.1: **AIC values for stochastic and deterministic alternative models**

Model	AIC
Stochastic (He fit)	1564.4
Stochastic (SPSA fit)	453.6
Deterministic (SPSA fit)	<b>438.7</b>

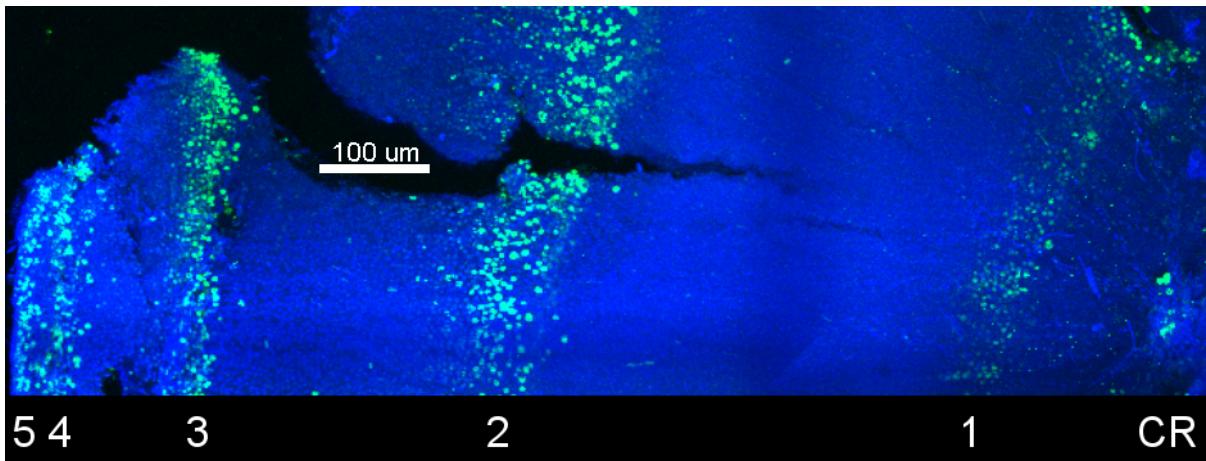
AIC: Akaike's information criterion. Lower values reflect better model explanations of the noted dataset. Lowest value noted in bold. Methods in Section 11.1.7. Code in Section 17.1.15.

We conclude that the SMME is not the best available explanation for variability in zebrafish RPC lineage outcomes. The assumed, but unexplained, linear succession of mitotic mode phases provides the overall structure of the model output. Variability in lineage outcomes may be supplied by different model ingredients without any loss of explanatory power on new datasets. To conclude that these models support a stochastic process governing mitotic mode, ruling out other types of explanation, is unjustified. It is likely that any number of different types of SSMs, representing other sorts of processes (such as asymmetric segregation of fate determinants, or differential spatial exposure to extracellular signals), can produce identical model output. Given this, we suggest SSM models of this type are inadequate for the task of locating the source of variability in RPC lineage outcomes.

## 2.4 SMME SSMs cannot explain the post-embryonic phase of CMZ-driven zebrafish retinal formation

The zebrafish retina, like other fish retinas, and unlike the mammalian retina, continues to grow long after the early developmental period, indeed, well past the organism's sexual maturity. This may be unsurprising, given that zebrafish increase in length almost ten-fold over the first year of life [PEM<sup>+</sup>09], necessitating a continuously growing retina during this period. In fact, the quantitative majority of zebrafish retinal growth occurs post-metamorphosis, outside of the early developmental period. This growth occurs due to the persistence of a population of proliferative RPCs present in an annulus at the periphery of the retina, called the ciliary or circumferential marginal zone (CMZ), which plates out the retina in annular cohorts. A typical "tree-ring" analysis from our studies, marking the newly-synthesised DNA of cohorts of cells contributed to the retina at particular times with indelible thymidine analogues, is shown in Fig 2.6. These experiments show the structure of the adult zebrafish retina is dominated by contributions from the CMZ in the period between one and three months of age. Why peripheral RPCs in zebrafish remain proliferative, while those in mammals are quiescent [Tro00], and whether and how their behaviour might differ from embryonic RPCs, remain unresolved. Answers to these questions may have significant fundamental and therapeutic implications, especially given e.g. the possibility of

harnessing endogenous, quiescent, peripheral RPCs in humans for regenerative retinal medicine.



**Figure 2.6: Most zebrafish retinal neurons are contributed by the CMZ between one and three months of age**

Maximum intensity projection derived from confocal micrographs of a zebrafish whole retina dissected at 5 months post fertilisation (mpf). The animal was treated with BrdU at 1, 2, 3, 4, and 5 mpf for 24hr. Anti-BrdU staining of the whole retina reveals the extent to which the CMZ (which is responsible for retinal growth after approximately 72hpf) contributes new neurons in tree-ring fashion, extending out from the center of the retina (CR), which is formed before 72hpf. Monthly cohorts are labelled appropriately. Scale bar, 100  $\mu$ m. Methods in Section 11.1.2.3.

The He SSM was deployed in Wan et al. [WAR<sup>+</sup>16], with the claim that the He SSM explains the behaviour of CMZ RPCs. Wan et al. argue that a slowly mitosing population of bona fide stem cells, at the utmost retinal periphery, divides asymmetrically to populate the CMZ with He-SSM-governed RPCs. In other words, the usual suggestion that CMZ RPCs undergo a somewhat different process than embryonic RPCs, perhaps recapitulating across the peripheral-central axis some progression of states or lineage phases that embryonic RPCs pass through in time [HP98], is repudiated in favour of one model which describes the behaviour of all RPCs throughout the life of the organism, with the addition of a small population of stem cells to keep the CMZ stocked with RPCs. If so, this might suggest that the problem of activating quiescent stem cells in the retina is simply that- one need only sort out how to throw the proliferative switch in these cells, since the proliferative and fate specification behaviours of the resultant RPCs will reliably be the same as those observed in development.

While we determined that the SMME is not the best available explanation for RPC lineage outcomes, we still felt that the SSMs associated with this explanation might be used to elucidate this point. In particular, if it is the case that the He SSM provides good estimates of lineage size and proliferative dynamics in the early zebrafish retina, it should be possible, using this model, and the estimates of putative stem cell proliferative behaviour provided by Wan et al., to simulate the population dynamics of the CMZ, at least through the first few weeks of the organism's life.

In pursuing this point, we noted a peculiar feature of the He SSM not documented by any of the SMME reports: the cell cycle model overstates the *per-lineage* rate of mitoses by as much as a factor of 3. That is, the mitotic mode rate data presented in He et al., recapitulated here in Fig 2.5, panels D, E, and F, and used to optimise both the SM and DM, are probability density functions that are not

standardised on a per-lineage basis. These data simply indicate the distribution of mitotic events of a particular type. When we take all of the mitotic events documented by He et al. and calculate the probability of any such event occurring per lineage, per hour, we obtain the values presented in Fig 2.7.

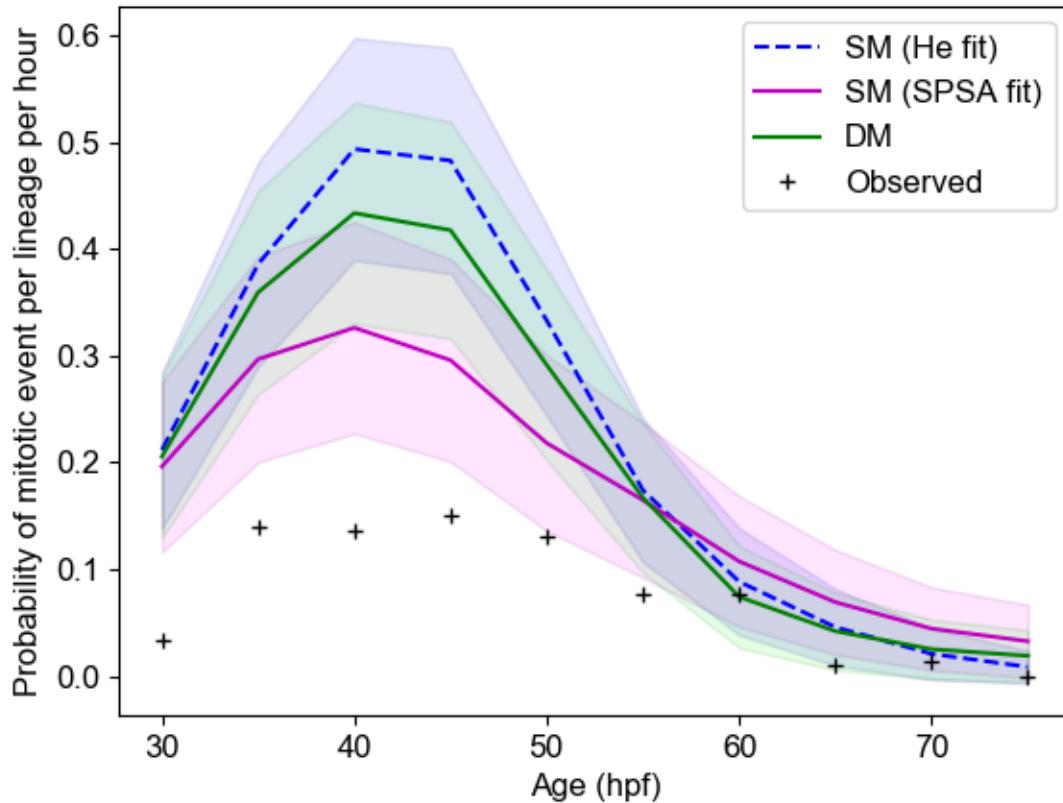


Figure 2.7: **Per-lineage probabilities of mitoses, He et al. observations compared to model output**

Probability of observing a mitotic event over the period studied by He et al., per hour, per lineage. Model output is presented as mean  $\pm$  95% CI. Methods in Section 11.1.7. Code in Section 17.1.17.

Similarly, when we performed cumulative thymidine analogue labelling of the 3dpf CMZ, (using the assumptions of Nowakowski et al. [NLM89], which treat the proliferating population as a homogenous, and dividing asymmetrically; these assumptions are flawed but adequate for a rough estimate) we obtain an average cell cycle length of approximately 15 hours, more than twice as long as the He SSM's mean cycle length. These data are displayed in 11.2. Therefore, the He SSM (in both its original and refit parameterisation, as well as the deterministic alternative, since they all rely on the same proliferative model elements) substantially overstates the proliferative potential of both embryonic and early CMZ RPCs. Since we observed a massive build-up of proliferating RPCs between two and four weeks post-fertilisation, we thought this might actually suit this later context better.

To estimate the total annular CMZ population in zebrafish retinas over time, we counted proliferating RPCs present in central coronal sections of zebrafish retinas throughout the first year of life, treating

these as samples of the annulus, and calculated the total number of cells that would be present given the diameter of the spherical lens measured at these times. Our simulated CMZ populations were constructed at 3dpf by drawing an initial population of RPCs governed by the He SSM (using the original fit parameters, which further exaggerate the proliferative potential of these lineages) from the observed distribution. We added immortal stem cells amounting to one tenth of this total. This is likely an overestimate, given that these putative stem cells are thought to be those in the very peripheral ring of cells around the lens, of which typically two to four may be observed in our central sections with an average of over one hundred proliferating RPCs. Moreover, these simulated stem cells were given a mean cycle time of 30 hours, proliferating about twice as quickly as Wan et al. suggest. Finally, to reflect the fact that these stem cells contribute to the retina in linear cohorts [CAH<sup>+</sup>14], and more of them are therefore required as the retina grows, the stem cells were permitted to divide symmetrically when necessary to maintain the same density of stem cells around the annulus of the lens. Two hundred such CMZ populations were simulated across one year of retinal growth.

The results of these simulations are displayed in Fig 2.8, overlaid over CMZ population estimates derived from observations. Given the generous parameters of the population model, consistently overestimating the proliferative potential of embryonic and early CMZ RPCs, it is surprising that the He SSM proves completely unable to keep up with the growth of the CMZ in the first two months of life. Indeed, the unrealistically active stem cell population the simulated CMZs are provided with is unable to prevent a near-term collapse in RPC numbers, only catching up after the months later as the number of stem cells increases with the growth of the lens. Thus, even given permissive model parameters, the He SSM is not able to recapitulate the quantitatively most important period of retinal growth in the zebrafish.

This analysis suggests that observations of RPCs in embryogenesis and early larval development are unlikely to provide a good quantitative model of the development of the zebrafish retina, even abstracting away spatial and extracellular factors as SSMs necessarily do. Our data point to a second, quantitatively more important phase of retinal development, between approximately one and four months of age, in which RPCs contribute more cells per lineage than in early development. It is likely that models that closely associate proliferative behaviour with fate specification, like those of the SMME, will be unable to explain this period. Recent evidence suggests that mitotic and fate specification behaviours in RPCs may be substantially uncoupled [ESY<sup>+</sup>17]. This would permit the CMZ population to scale appropriately with the growing retina. Alternatively, it is possible that RPCs have heterogenous proliferative behaviour, and that this heterogeneity is mainly apparent later in development.

## 2.5 Conclusion

Simple stochastic models are familiar tools for stem cell biologists. Introduced by Till, McCulloch, and Siminovitch in 1964 [TMS64], they proved their utility in describing variability in clonal lineage outcomes of putative stem cells, originating from macromolecular processes beyond the scope of cellular models (and beyond the reach of the molecular techniques of the time). More prosaically, their simplicity afforded computational tractability in an era when processing time was relatively scarce, allowing early access to Monte Carlo simulation techniques. That said, their abstract nature emphasizes an aspatial, lineage-centric view of tissue development, which cannot account for the generation of structurally complex tissues beyond cell numbers, and, perhaps, fate composition. This relatively loose relationship to the

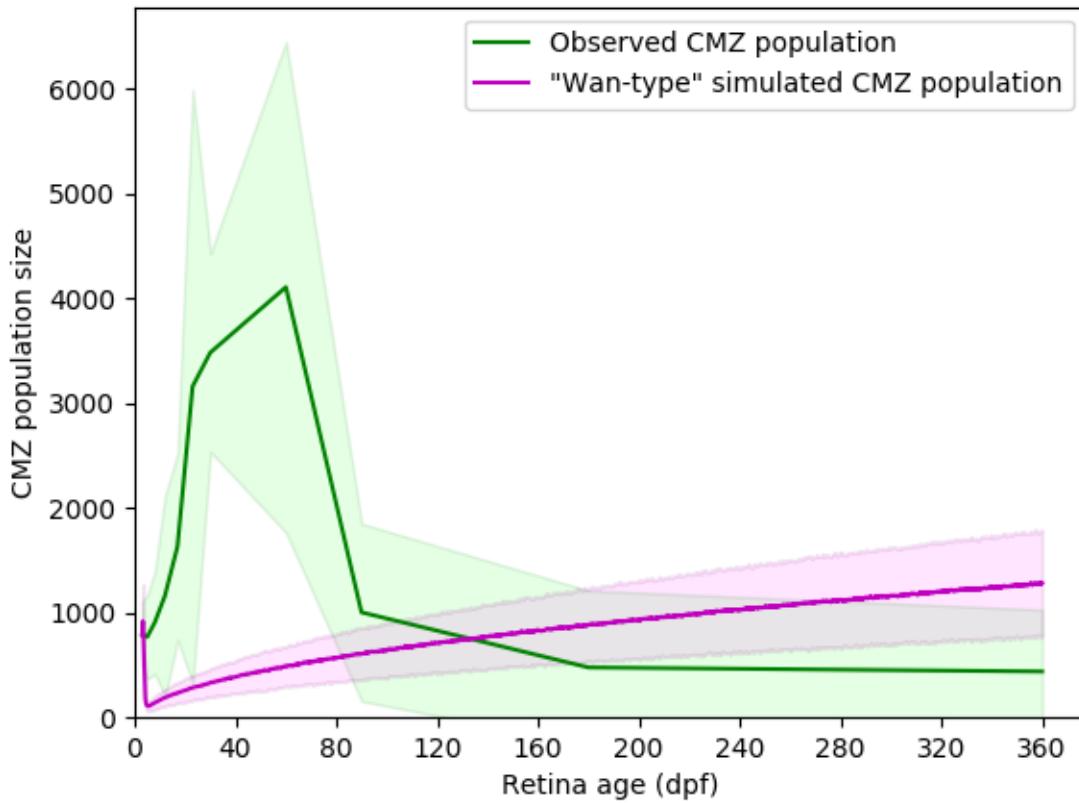


Figure 2.8: CMZ population of proliferating RPCs: estimates from observations and simulated Wan-type CMZs

Annular CMZ population was estimated from empirical observations of central coronal cryosections stained with anti-PCNA, a marker of proliferating cells, standardising by the diameter of the spherical lens observed in these animals. "Wan-type" CMZs were initialised with a number of RPCs drawn from the observed 3dpf population distribution, and given an additional 1/10<sup>th</sup> of this number of immortal, asymmetrically dividing stem cells, as described in the text, and simulated for the first year of life. All data is presented as mean  $\pm$  95% CI. Methods in [Section 11.1.2.1](#), [Section 11.1.7](#). Code in [Section 17.1.18](#).

complex morphogenetic environment provides few constraints on the model configurations that may produce similar outputs. This can result in modellers being led astray by their apparently good fits to observations, which is why the fit of one highly parameterised model cannot be taken, alone, as evidence for some theory; it is very rare that there is not a model representing an alternative theory that cannot be made to produce a reasonable model fit. Indeed, as we demonstrate here, the use of standard model selection techniques may make plain that a completely contradictory theory is a better explanation for the data.

To some extent, this can be ameliorated by careful attention to the specific macromolecular or cellular entities that model constructs represent. The "mitotic mode" construct present in all SSMs is an ambiguous one in this regard. "Mitotic mode" is not, itself, a property of a mitotic event, but is rather

a retrospective classification of the event, after an experimenter observes whether progeny resulting from the event continue to proliferate. Its original appearance in SSMs was simply to allow calculation of clonal population sizes; it was never intended to represent a particular type of process or “decision” made by cells at the time of mitosis to continue proliferating or not. Any number of pre- or post-mitotic signals and processes may result in a particular mitosis being classified as PP, PD, or DD, without anything about the mitotic event itself determining this. The use of such a retrospective classification, rather than the identification of some physical property of the mitotic event (such as the asymmetric inheritance of fate determinants), is a concession that the actual macromolecular determinants of cellular fate are outside the scope of the model.

The SMME represents an attempt to connect this abstract, retrospective model construct to observations of noisy gene transcription [Rv08]. Motivated by the observation that momentary mRNA transcript expression in RPCs is highly variable from cell-to-cell [TSC08], the suggestion is that this transcriptional noise may be responsible for the observed variability in RPC lineage outcomes. Since this noise may be “tuned” to a degree by e.g. promoter sequences [RO04], it could plausibly be under selective pressure. There are two significant problems in identifying the SMME’s stochastic mitotic mode with this type of process. The first is demonstrated by constructing an alternative model with deterministic mitotic mode but variable phase lengths: the source of variability may be located elsewhere without compromising the explanatory power of the model. It is impossible to determine what sort of process might give rise to variability in RPC lineage outcomes by using SSMs in this fashion. The second problem is that a causal explanation of the presence of the signal, for which noise is a property, is elided in favour of emphasis on “stochasticity”. This is most apparent in the Boije SSM. In this model, Atoh7 and Ptfla TFs are available to provide their noisy signal in the 4th and 5th lineage generations, and at no other time. We do not dispute that a noisy signal may contribute to variability in that signals’ effects; rather, we suggest that it is the temporal structure of such a signal (assuming this structure can be empirically demonstrated, rather than assumed) that calls for causal explanation. The SMME reports explicitly disclaim the necessity for “causative hypotheses” in the case that a stochastic model provides a good fit to observations. As Jaynes remarked in his classic text on probability theory, “[stochasticity] is always presented in verbiage that implies one is describing an objectively true property of a real physical process. To one who believes such a thing literally, there could be no motivation to investigate the causes more deeply ... and so the real processes at work might never be discovered.” [JBE03]

In identifying the model construct with the physical processes determining RPC outcomes, the SMME obscures what seems to us to be the primary lesson to be drawn from the Harris groups’ beautiful *in vivo* studies of zebrafish RPCs. That is, compared to late rat RPCs in dissociated clonal culture, RPCs in intact zebrafish retinas produce far more orderly outcomes. To the extent that these outcomes are variable, the source of variability remains unidentified. We suggest that resort to “stochasticity” as an explanatory element should not be made in the absence of model comparisons that rule out alternatives with well-defined causal structures, lest we fall into the trap Jaynes warned us about.

# Chapter 3

## Toward a computational CMZ model comparison framework

### Synopsis

(1) The SMME introduces a phased structure in order to explain early RGC generation by *Danio* RPCs, which is why it misapplies the explanatory logic of Gomes et al. [GZC<sup>+</sup>11] (2) A general-purpose statistical approach to testing many different kinds of models is desireable, and available in nested sampling. (3) Models of RPC activity in the postembryonic CMZ should prioritise simplicity, which could be achieved with “slice models”. (4) Applying nested sampling to these models allows us to compare their explanatory quality rigorously.

### 3.1 SMME Postmortem: a wrong turn at Gomes

To return to the question posed in Section 1.6: has the SMME succeeded in finding order by blurring out the chaotic welter of mechanistic explanations for RPC function in retinogenesis? Chapter 2 argues the structure of the SMME models support explanations based on the original idea of a linear, deterministic series of stages through which RPCs progress, originally put forward by Cepko et al. [CAY<sup>+</sup>96], rather than stochastic processes. The important feature of the SMME models is where they depart from the original Gomes SSM: the assumption of the linear structure of temporal phases. The succession of stages is the model ingredient that produces the structure resembling the data, not the various random variables associated with mitotic mode, which we have proven by demonstrating that a deterministic progression of mitotic mode models the observations better than its stochastic counterpart. By introducing variability into the lengths of the linear temporal program already assumed by the SMME models, we can remove variability from the mitotic mode and produce a superior model with fewer parameters. Only by the process of counterinduction [Fey93], the comparison of models with different propositional structures about reality, does this become clear. The introduction of the linear phase structure is required to model the early contribution of RGCs in zebrafish. If this is left without some biological rationale, retinogenesis remains unexplained. The SMME thus does not achieve its aim of being a complete description of lineage outcomes in retinogenesis, and this is confirmed by our observation that the Wan

model [WAR<sup>+</sup>16] has little explanatory power outside the first few days of life, which makes up a small portion of the total retinal contribution to the zebrafish retina.

The first theoretical maneuver of the SMME, explaining the data with a model with structure that supports a stochastic mitotic mode process, cannot be achieved. The underlying data used to inform these models speak better to an entirely different selection from the array of theoretical options outlined in Section 1.2, the linear progression of competencies. Because the SSM model form does not usually have spatial dimensions, we did not test models involving variability in extracellular signals, but it is a good bet that such a model could be made to fit about as well as either of the ones tested in the previous chapter. In effect, if we are to follow the SMME’s inferential logic, variability in RPC outcomes can be explained by “stochastic variability” in any model parameter which affects fate outcomes. Because of the significance of this problem for biological inferences, an explanation of the Bayesian epistemological view of probability has been provided in Section 15.1.1. Moreover, a thorough argument that “stochasticity” cannot be a property of real existents is provided in Section 16.1. It suffices here to conclude that the SMME models do not achieve the aim of supporting “stochastic processes” as explanations over the alternatives.

The second theoretical maneuver, to nominate a particular macromolecular system as the physical locus of the stochastic process, suggests *Atoh7* and *Ptf1a* as labels for abstract, Bernoulli distributed processes. It remains obscure how model variables relate to their namesake transcription factors (transcription of the TF itself? activation of other genes?), and no measurements of these factors inform parameter selection. Plainly, these factors are involved in relevant RPC behaviours, but it is unclear why they have been nominated as causally upstream of the “mitotic mode” selection. Since the Boije model inherits the assumption of a linear progression of stages, it is very likely that a model with similar explanatory power could be built using the same strategy outlined above, locating random variability outside mitotic mode, although this is outside the scope of the present work.

We conclude that the sole formally testable model of RPC function in the zebrafish retina is not adequate for our purposes. But how did we get here? As noted in Section 1.3.2, the notion that the most significant behaviours associated with RPCs are produced by mechanisms intrinsic to the cells derives much of its empirical support from the Raff group’s work. This story began with the observation that co-culturing E15 rat RPCs in dissociated pellet cultures with P1 cells did not accelerate the appearance of the first rods derived from the E15 progenitors (which occurred at a similar time as *in vivo*), suggesting a partially intrinsic commitment “schedule” for these cells<sup>1</sup> [WR90]. The scope of these observations were dramatically expanded by Raff’s subsequent work, intended to address the relative significance of intrinsic versus extrinsic processes in RPC function by comparing clonal RPC lineages in fully dissociated clonal-density cell culture to those in intact explants [CBR03]. The remarkable finding of this study was that dissociated E16-17 rat RPC lineages produce very similar numbers and types of retinal neurons as their tissue-embedded counterparts, albeit without morphological or molecular markers of mature neurons. This observation provided strong evidence for the predominance of RPC-intrinsic processes in determining both mitotic and fate outcomes for late RPC lineages. The complex, spatially organised context of intact explanted tissue seemed to only be required for the maturation of neurons, and was not required to regulate proliferation, or the initial commitment to an appropriate distribution of lineage outcomes. As correct cell numbers and types are the two most important parameters that must be

---

<sup>1</sup>Co-culturing with P1 cells, did, however, significantly increase the proportion of E15 RPCs specified as rods, resulting in Raff’s suggestion here that both intrinsic and extrinsic factors are important.

controlled for RPCs to produce a functional retina of the appropriate size, the suggestion that both are largely determined by intrinsic processes seemed to confirmation of Williams and Goldwitz's much earlier suggestion [WG92], against the prevailing view of the day, that lineage had a greater role to play than cellular microenvironment in RPC contributions. Interestingly, Raff's interpretation of their 2003 data was that RPCs were most likely stepping through a linear, programmed developmental sequence, rather than undergoing shifts in the probabilities of variable outcomes over time. It is notable that this interpretation arises not from the data collected in their study, but from considerations of a single unusual clone reported by [TSC90], and by analogy with drosophila neuroblasts. In retrospect, these arguments for linear sequences of deterministic RPC outcomes do not seem particularly strong, and it is perhaps unsurprising that these studies are remembered mainly for highlighting the importance of RPC-intrinsic processes.

The Gomes study, with its detailed study of particular rat late-embryonic RPC lineages, thus seemed to solidify the notion that unpredictable RPC-intrinsic processes dominate lineage outcomes [GZC<sup>+</sup>11]. But this study, like its forebears originating in Raff's work, is concerned with a population of RPCs that is too old to produce RGCs. This is the essential point: the SMME does not try to explain the early appearance of RGCs in terms of some macromolecular mechanism, although this is the single most significant difference between the zebrafish RPC lineages studied by Harris and the rat lineages studied by Raff and Cayouette [CBR03, GZC<sup>+</sup>11]. It is only by assuming the unexplained linear temporal structure of RPC specification that the SMME models achieve their apparently good explanatory quality.

### 3.2 Implications of the SMME study for modelling CMZ RPCs

Having taken seriously the possibility that an adequate model of zebrafish retinogenesis already exists, and concluded in the negative, we may ask how to improve on this state of affairs. Firstly, careful model estimation is needed, and to have any confidence about our interpretation of the model, we must test alternatives that make different assumptions about the causal structure of the phenomenon being explained by the models. Can we conclude that the approach used in Chapter 2 is adequate to our task? There are two elements to consider: the appropriateness of the modelling approach represented by the SSM in relation to the hypotheses we wish to test, as well as the soundness of the statistical procedures.

Taking up the SSM, its most attractive features are its computational efficiency and the ease of producing a model to fit some particular case- the SMME models include some elements like the correlation of cell cycle lengths in mitotic sisters that greatly improve the temporal modelling of RPC lineage outcomes, for instance, which we will adopt in simulations presented below. The observations arising from the SMME strongly suggest that we would like to test hypotheses about RGC specification in order to find better models of RPC function. Carefully reconsidering the hypothesis of Neumann et al. [Neu00], that Shh from nearby RGCs induces cell cycle exit and RGC specification, would seem to be a high priority, for instance. Can such a scenario be represented in an SSM? One could model the probability of being within Shh induction range of an RGC in the early part of a lineage's life, for instance. Doubtless, a model that incorporated variable RPC specification outcomes determined on this basis could be made to fit data about as well as the variable mitotic mode or variable phase length models tested above. That said, it is not very clear that a highly abstract representation of signalling would constrain the inferred RGC production rate well. Moreover, while we showed that our deterministic mitotic mode model fit test data somewhat better than the stochastic model, the modest size of the calculated AIC differences

suggests that comparing SSMs is not a particularly good way to distinguish even alternative hypotheses about the structure of processes the SSM models explicitly, like cell cycle length and mitotic mode.

It seems that we will be unable to test important models, including those involving documented macromolecular explanations of RGC specification, without the ability to represent some spatial information. Still, the computational benefits of the SSM are too appealing to leave out of the toolkit entirely, and the type of spatial information that is ultimately best used to constrain alternatives on processes like RGC fate commitment could be highly abstract. We would like to leave the door open for models of many forms, prioritising simpler ones. Taken together, this suggests that we need a very general method of testing models of potentially very different structures against one another.

Turning then to our statistical procedures, are these adequate to our goals? We can broadly characterize the approach taken as estimating the local optimum of a loss function for model output, given the dataset; specifically, minimizing [Akaike's information criterion by simultaneous perturbation stochastic approximation \(SPSA\)](#). This procedure has some advantages. AIC is a well-understood measure, well-grounded in information theory, which penalizes superfluous model complexity in a consistent and rigorous way. SPSA is a widely used, well understood algorithm that can be applied to any reasonable euclidean parameter space; cases where parameter spaces are bounded (typical of biological models where negative parameter values are usually nonsensical) are explicitly accounted for, and so on. This procedure is better than many that are available.

Still, after the experience of the SMME model comparison, caution is warranted. The AIC calculation is based on a single estimate for the locally optimal parameterisation of the model. Relative AIC rankings are strongly influenced by the "well depth" of the AIC surface at the local minimum in parameter space<sup>2</sup>. Indeed, blind interpretation of AIC rankings has lead to its use in ecology being described as a "cult" [BB20]. It is easy to imagine practitioners being reduced to "AIC hacking", in the same manner that "p hacking" occurs, in order to achieve some arbitrary value for a hypothesis.

Moreover, while SPSA is a practical algorithm its statistical guarantee is only that it will almost-surely find the local minimum of the loss function. For highly parameterised models with complex loss function surfaces, there will be many local minima for SPSA to get "stuck" in that are far from the global optimum<sup>3</sup>. If we are evaluating hypotheses in order to make decisions about potentially years-long research projects, more certainty about the reliability of the method is necessary. Finally, while SPSA is requires much less computational effort than more global Monte Carlo parameter estimation techniques like simulated annealing or Hamiltonian Monte Carlo, it requires about as much application-specific tuning.

Fortunately, a complete system of Bayesian inference, which addresses all of the problems mentioned above, has been promulgated over the last 15 years: nested sampling. Originally introduced by John Skilling [Ski06], this use of this system has became widespread in cosmology [Tro08, FHB09, HHHL19], where its generality and ability to cope with complex, high dimensional parameter spaces has been well proven. This system is discussed in detail in [Section 15.2.7](#). We have implemented two separate samplers for the exploration of parameter spaces using the overall nested sampling approach: `GMC_NS.jl`, described in [Chapter 7](#), and `BioMotifInference.jl`, described in [Chapter 10](#). `GMC_NS.jl` is a prototype general purpose Galilean Monte Carlo sampler intended to perform nested sampling of arbitrary tissue- or cell-level biological models, while `BioMotifInference.jl` is a highly specialized, well-tested, ad hoc

---

<sup>2</sup>These problems are discussed in more detail in [Section 15.2.1](#) and [Section 15.1.3](#).

<sup>3</sup>This is part of what is meant by "the curse of dimensionality", when speaking of the difficulty of sampling the loss function in high dimensional parameter spaces.

sampler for evidence calculation and MAP estimation of independent component analysis models of genetic sequence emission. `GMC_NS.jl`, while less mature, is of greater general interest, and having it in hand allows us to broadly consider the features of CMZ models we might like to test, before proceeding to test the function of the package and to determine its general utility in tissue-scale models.

### 3.3 CMZ models in a putative model comparison framework

The simulations presented in Chapter 2 consisted solely of abstract collections of lineages. The members of these model colonies have no activities beyond proliferation, and no functional attributes beyond their proliferative status<sup>4</sup>. Despite the underlying code consisting of relatively high performance C++, these simple models nonetheless occupied the local component of the cluster used in this work for more than a week. Because the introduction of explicit three-dimensional spatial simulation implies much more computational expense, this could strongly limit the ability to test inferences with local resources. This constraint dominates other considerations in building models of the CMZ. Since funding constraints limit the cloud-export of computational burden from the confines of the research institution, it is reasonable to proceed upwards in model computational expense, seeking the minimum model complexity required to compare hypotheses of interest to us.

#### 3.3.1 Spatial dimension of the models: the "slice model"

Although decomposing the RPC population of the CMZ into a collection of unordered, independently-proliferating SSMs prevents us from assessing many interesting hypotheses, a computational model of the entire CMZ or retina may not be required. Because observations suggest that RPC lineages contribute to the retina in linear cohorts of neurons, any particular centrally-oriented slice of the CMZ annulus will be responsible for the generation of the neurons central to it. The retina can be represented by a series of these "slice units", lined up radially, like slices of pie. A complete "slice unit" would include the central-most larval remnant, contributed by embryonic retinogenesis, surrounded by the CMZ's more ordered neural contribution from the postembryonic period, and, peripherally, the CMZ itself. Depending on the hypotheses to be tested, the differentiated central retina may be mostly irrelevant, so the modelled slice may consist only of the CMZ and its interface with these central neurons. It is important to note that these slices are conceptually different from the linear cohorts generated from particular lineages, the so-called 'ArCCoS', which justify the slices conceptually [CHW11]. It is not necessarily the case that a slice model would consider only one RPC lineage; the slices are better thought of as spatial boxes that sample the CMZ, the conceptual equivalent of the histological section through the eye<sup>5</sup>.

Slice models are especially attractive because the proliferative status, position, specified fate, etc. of simulated cells can be directly compared to measurements of fixed sections of retinal tissue. This is especially important for studying the postembryonic CMZ, which rapidly becomes optically inaccessible due to the increasing thickness and pigmentation of overlying tissue, so that live imaging often cannot be used<sup>6</sup>.

---

<sup>4</sup>I.e. the specified identity of post-proliferative cells in these models has no function within the model.

<sup>5</sup>The case of only one simulated lineage may still arise given thin enough sample boxes or old enough animals, but it is a limiting case and probably would not be the norm.

<sup>6</sup>It is worth noting that the spatial parameters of such models would pertain to fixed and not live retinas; it is probable that some scaling relation compensating for fixative shrinkage could allow the mixture of live imaging data in evidence calculations.

If the retina can be thought of as a series of slice models, and the activity of the CMZ is basically homogenous around its circumference, the activity of the CMZ may be usefully represented with a single such model. Moreover, the slice need only include one portion of the retinal periphery, the orientation of which is irrelevant (i.e. the model parameterisation for the dorsal portion of a coronal slice could be identical to the ventral). It may be erroneous to abstract such a slice from its context, because the modelled cells are in contact with adjacent slices. In this thesis, in order to model the interaction between the simulated slice and the rest of the eye, we use a model of lens growth to produce an estimate of the number of additional fractional CMZ slices the modelled slice would have had to produce in order to maintain the CMZ annulus. The appropriate number of RPCs are subtracted from the slice's population, which models the lateral, slice-adjacent interface of the CMZ in addition to niche exit from the CMZ into the neural retina central to the CMZ slice. The various models used in Chapter 4 are summarised in an inferential pipeline schematic in Figure 3.1.

The zebrafish retina is a manifestly asymmetrical structure, with an optic nerve positioned ventro-temporally relative to the center of the optic cup. The CMZ itself is generally understood to be an asymmetric structure, with a larger dorsal than ventral population. This calls into question the assumption of CMZ homogeneity outlined above. It is nevertheless plausible that the appearance and maintenance of these structural features of the zebrafish retina could be explained with a set of slice models, for example, one for each of the dorsal, ventral, nasal, and temporal extrema<sup>7</sup>. Ultimately, these considerations are only relevant if our objective is to compare model explanations for retinogenesis as a whole. If we restrict ourselves to the more modest goal ranking causal influences on the lineage outcomes of RPCs in the dorsal extremity of the CMZ, a slice model could still be valuable even if it fails to explain aspects of tissue-level zebrafish retinogenesis.

---

<sup>7</sup>Such models are implemented by `CMZNicheSims` as `MultiSlice_Models`.

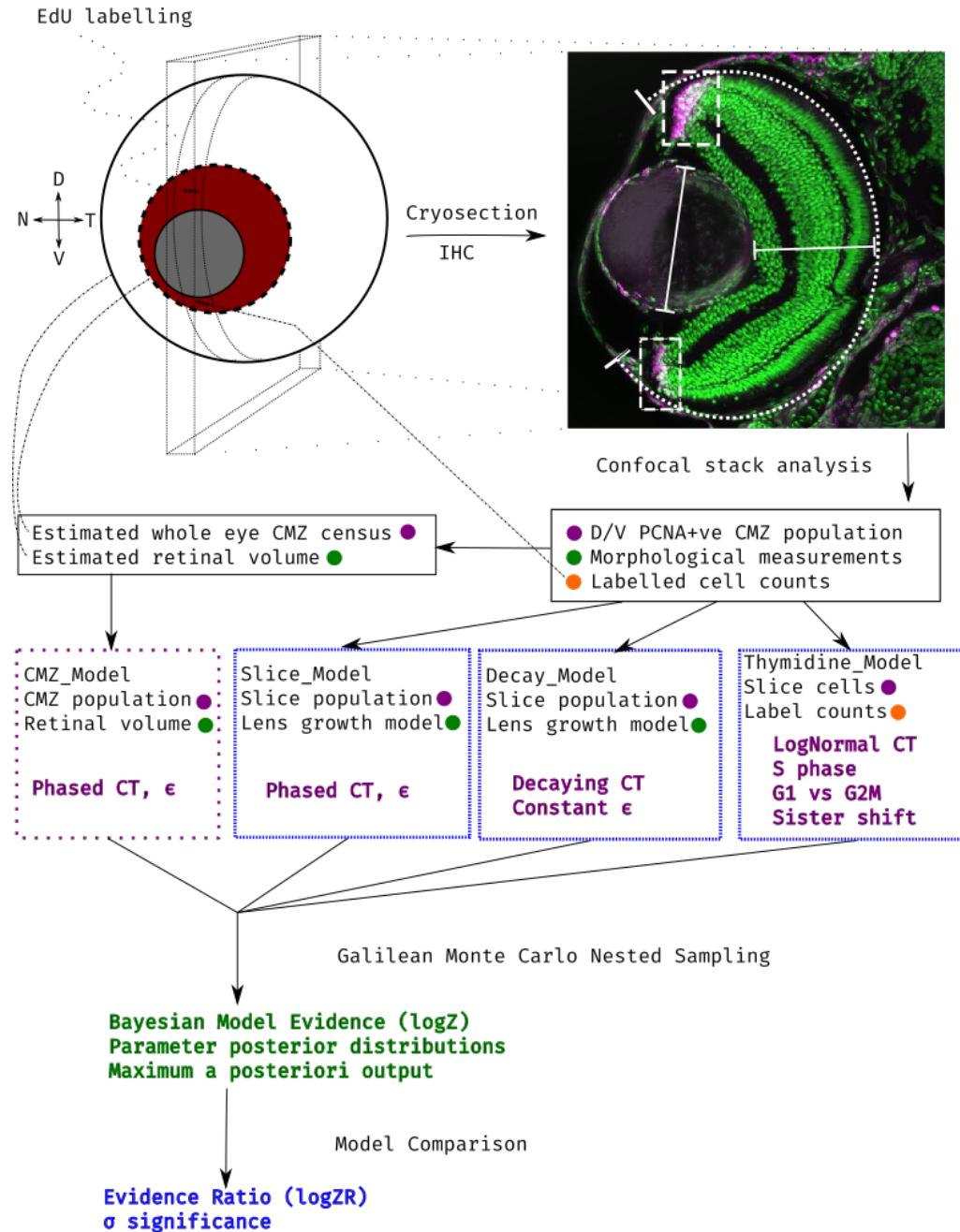


Figure 3.1: Whole-eye and slice model abstractions of CMZ RPC populations used in Chapter 4

D/V: Dorsoventral axis. N/T: Nasotemporal axis. Red torus: CMZ. Grey circle: Lens. Colored dots denote the data sources used in each model. Magenta text gives the parameters of the model. CT: Cycle time.  $\epsilon$ : exit rate. See Chapter 8 for more.

Data to estimate models were acquired from immunohistochemical analyses of cryosections through *Danio* eyes. Confocal stacks were segmented and analysed to count PCNA+ve and (if applicable) EdU-labelled CMZ cells in the slice as well as to measure the lens and neural retina. These slice data were used to estimate whole-eye measurements for a fully abstract **CMZ\_Model**, representing the entire peripheral RPC population, as well as directly compared to the output of three different types of **Slice\_Model**. Quality estimates from these models were sampled by GMC-NS, producing model evidence, posteriors, and MAP output. Model evidence was compared to produce estimates of the evidence ratio and significance.

### 3.3.2 Temporal resolution of the models

Another important consideration for any CMZ model comparison framework is the time-scale of any phenomena which are to be admitted as possible causal contributors to retinogenesis. While it is reasonable to think that proliferative events may be well-described with a model that operates on a scale of days and fractions thereof, and that such a model would be well suited to describing the full sweep of CMZ activity across the life of the organism, very few of the relevant macromolecular processes are likely to be well-described with a resolution more coarse than seconds or minutes. The implied difference in the number of calculations required to simulate any given time period is thus several orders of magnitude. A simplifying assumption of the temporal homogeneity of CMZ activity would allow us to abstract long developmental time frames; an explanation that pertains to a few hours can perhaps be extended without recalculation.

If an assumption of temporal homogeneity proves inadequate, the functional structure of the simulation will be determined by this consideration. To illustrate this, consider a case where we wish to incorporate measurements of eye pressure, or membrane tension across the retina, as inputs into the proliferative activity of the CMZ, by way of progenitor cortical tension [Win15]. This would permit assessing whether modelling tissue-mechanical inputs to cellular activities allows us to extract additional information from our observations. If such physical model elements are to be included, the functions which relate physical parametric data to the proliferative behaviour of CMZ progenitors must not operate on a time scale too short to cover the period of interest. Suppose we are mainly interested in explaining the assembly of functional units of the mature, specified retina by the CMZ, from the first division of the presumed distal stem cell responsible for the unit to the determination of the last neuron of its functional column. If this process takes days, finite element approximation of cortical tension with a resolution small enough to capture important details of relevant cellular movements (e.g. interkinetic nuclear migration) will probably be too expensive to allow for effective application of Monte Carlo techniques, requiring thousands or millions of simulations per call of the model likelihood function.

The models in this thesis mostly are of very coarse resolution, because they model the CMZ over significant portions of the organism's life; the exception is the `Thymidine_Model`, which is a cell-based intended to model thymidine analogue labelling of cells passing through S-phase, and is not intended to model RPCs leaving the niche at all. The `Thymidine_Model` is also by far the slowest of the `CMZNicheSims` to estimate.

## 3.4 Structuring models under uncertainty

The extent to which model simplifications are justified, and the types of phenomena that could be explained with these models, depend heavily on considerations that can only be informed by observations. As has been demonstrated in Chapter 2, the growth of the CMZ population cannot be explained by SMME models fitted to embryonic RPC activity. However, it remains unclear what sort of alternative structure is justified by observations, given our uncertainty about inferred parameters of the populations being measured. Because we have, in `GMC_NS.jl` a general method of comparing differently parameterised models against our observations, we can seek to select models from among alternatives on the basis of their total explanatory power for the observations, independent of the particular model structure or parameter values. Because this process also produces samples from the posterior distribution of the model's parameters, we are also able to account for our uncertainty on these parameters. This allows

us to broadly answer the question of what sorts of hypotheses may be addressed by the conventional immunohistological techniques we have used thus far, and what changes to methodology or analysis may be necessary to produce data that adequately constrains models of interest.

Our general approach is to estimate model evidence over broad prior distributions on model parameters. We assume as little as possible about the processes underlying our results. The values used in these comparisons are the logarithm of the model evidence,  $\log Z$ , also understood as the logarithm of the marginal probability of the model. Likelihoods and probabilities are often compared in ratios; evidence is the same, and since the logarithm of a ratio is the logarithm of the numerator minus that of the denominator, this log-ratio,  $\log ZR$ , may be calculated by subtracting the evidence of one model from another. This is discussed in more detail in [Section 15.2.7](#). This general model-comparison logic is iterated to address most of the important hypotheses we test in the following chapters.

## Chapter 4

# Bayesian characterisation of postembryonic CMZ activity

### Synopsis

(1) Variability in CMZ RPC population sizes and whole retinal volume estimates is better modelled Log-Normally than Normally. (2) The dynamics of CMZ RPC population measurements over the first year of life suggest that CMZ RPCs may be passing through a succession of phases with different cell cycle and niche exit rates. (3) Whole-eye modelling of CMZ population and retinal volume estimates suggests that only two such phases are necessary to explain these dynamics. (4) Slice modelling of the CMZ population more successfully recapitulates data, and suggest there is no timing difference in proliferative schedule across the CMZ's dorso-ventral axis. (5) Modelling RPC dynamics with a single phase of decaying cycle rate and constant exit rate is a superior description to the 2-phase models; decay models also suggest that there is no dorso-ventral gradient of cell cycle and niche exit parameters. (6) Nowakowski-type modelling of cumulative thymidine analogue labelling gives overstated cycle rates, comparable to the defective whole-eye models, but confirm no dorso-ventral differences in these parameters. More accurate estimates can be obtained using cell-based slice modelling. (7) Cohorts of neurons contributed to the retina by the CMZ are divided among layers in fractions that remain stable over time; however, the neural fates specified vary over time. (8) Retinal neurons contributed by the CMZ are turned over by microglia at a rate too low to be detected in labelled cohorts. (9) Explaining postembryonic CMZ dynamics requires different models from proliferatively-restricted embryonic lineage simulation. Slice models allow abstraction of spatial complexity. (10) Many opportunities for improved slice modelling exist.

### 4.1 Independent Log-Normal modelling of CMZ parameters

To take up the question of how CMZ RPC activity evolves over time, we measured CMZ and retinal parameters in histological studies of cryosections of zebrafish eyes harvested over the first year of the animal's life. The initial approach is to geometrically impute measurements whole-eye CMZ and retinal volume from sample central cryosections, calculated as described in Section 11.1.4. These derived measurements form the first dataset we consider.

While it remains common to assume that population census data are Normally distributed, it is been known [Hea67] that Log-Normal distributions are usually better models of the outcomes produced by additive processes with small, variable steps (like population sizes or income distributions). Since all of our models require modelling distributions of population data, we start by selecting the most explanatory population model.

As described in more detail in Section 12.0.1, we conclude that population variability in CMZ census and retinal volume estimates are best described by independent Log-Normal distributions. Because Log-Normal distributions are simply transformed Normal Gaussian distributions, we may model our uncertainty about their parameters with Normal-Gamma distributions over the mean and variance of the underlying Normal distribution of the Log-Normal population model [Mur07]. This is explained in more detail in Section 15.2.4. A useful analytic feature of the Normal-Gamma prior is that the marginal posterior distribution of the mean, assuming an uninformative (ignorance) prior, is a location-scaled T distribution. Most of the descriptive statistics in the next section, therefore, calculate the 95% credible interval for the posterior mean of the underlying by T distributions (with the correct change of variables by exponential transformation to produce the features of the correct Log-Normal distribution). Unfortunately, differences of T distributions are not, themselves, necessarily T-distributed, so we have relied on Monte Carlo estimation of rates of change of the these posterior means over time.

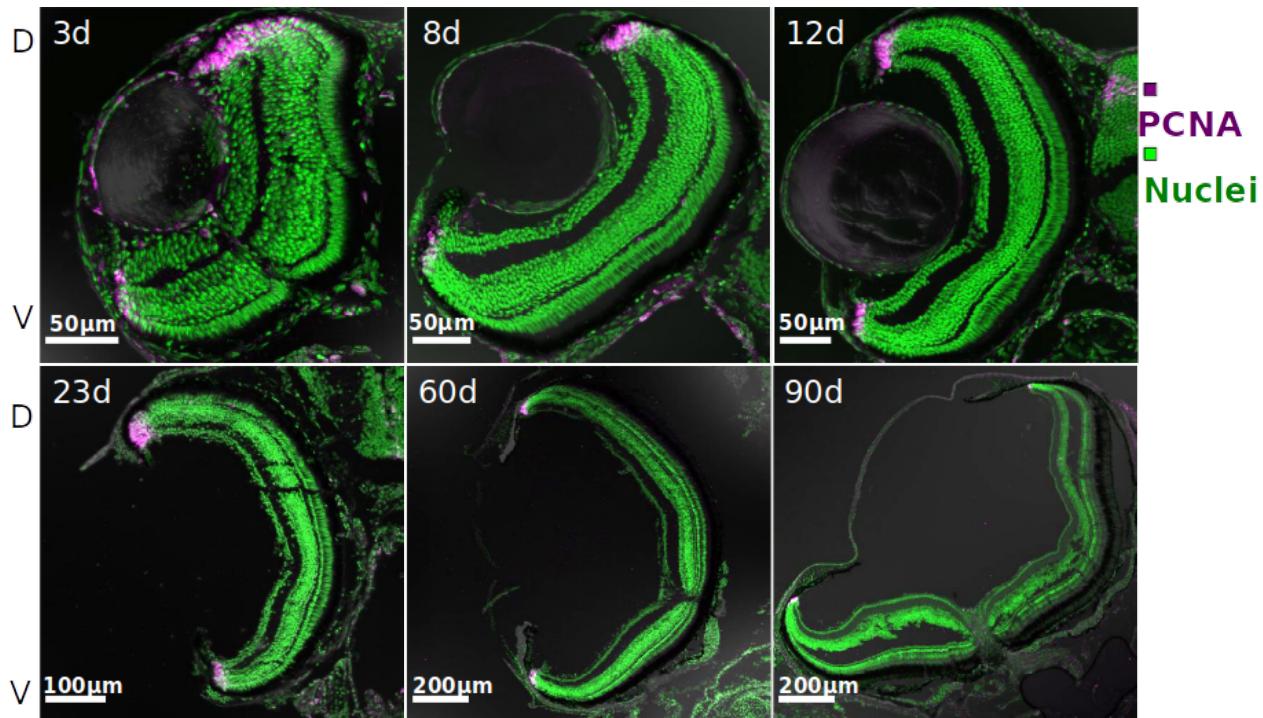
## 4.2 Survey of CMZ population and gross retinal contribution

If it is true that the majority of zebrafish retinogenesis occurs postembryonically, and that models trained on embryonic data do not describe this period well, as seen in Chapter 2, what characterises this CMZ-driven phase of retinogenesis? A schematic overview of the appearance of the CMZ during the first three months of post-embryonic life is presented in Figure 4.1, displaying the progressive expansion of the neural retina and the relative size of the CMZ over this time period.

We first estimate mean CMZ annulus population and retinal volume over the first year of life, in Figure 4.2, panels A and C. The lack of growth observed in the first week of life suggests an initial quiescent period in the population history of the CMZ. Observations in Figure 5.3, showing CMZ RPCs are proliferating too slowly to be labelled by a 12 hour pulse of a thymidine analogue at 10dpf in the siblings of npat mutants, also suggest this quiescence. On the other hand, we have good confidence that retinal volume continues to grow over this time; 99.56% of the marginal posterior mass of the mean 5dpf estimate is above the 3dpf mean, which may indicate any proliferative pause is too short to appear in the retinal volume data.

The first two to three weeks of life (magnified in lens insets in panels A and C) see a slow build in both estimated CMZ annulus population and retinal volume compared to the large increase which follows. While zebrafish are better staged by size than age [PEM<sup>+</sup>09], the onset of this growth seems to come earlier than the metamorphic transition from larval to juvenile stages at 45dpf [SH14]. This raises the question of how the CMZ contributes to the retina during the critical period of exponential growth of the organism, between about 45 and 90dpf. It is plausible, for instance, that the growth of the CMZ population mainly reflects the increased cell cycle rate, with RPCs specifying at some steady rate, or that the CMZ builds itself up for a wave of increased niche exit and specification somewhat later, similarly to the sequence of events in the embryonic central retina.

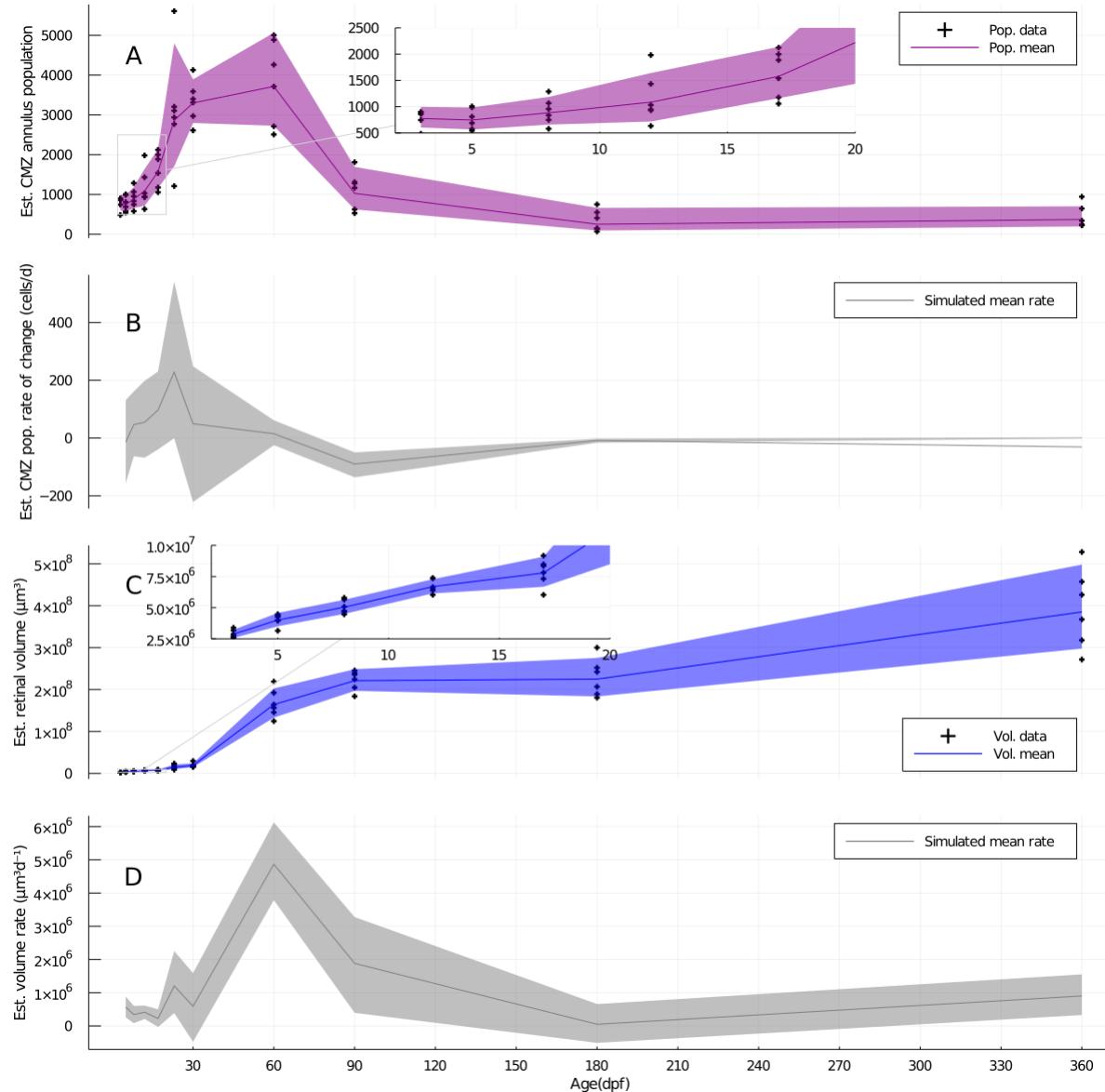
To ascertain this timing, we simulated the mean daily rate of change of the CMZ annulus population



**Figure 4.1: Micrographic overview of CMZ development in *D. rerio*, 3-90dpf**  
 Representative confocal micrograph maximum intensity projections (MIPs) of 14 µm coronal eye cryosections, processed via IHC, of zebrafish of noted ages. Magenta – anti-PCNA staining. Green – Hoechst 33342 nuclear staining. Note changing scale bars. Methods in [Section 11.1.2.1](#).

and retinal volume, by performing Monte Carlo difference operations between samples from the marginal posterior means of subsequent timepoints. These simulated mean rates and their associated, empirically determined confidence intervals are plotted in [Figure 4.2](#) panels B and D. These show the time-structure of the phenomenon; the increase in simulated daily CMZ population growth rate occurs before the increase in retinal volume, but the CMZ population estimate does not begin to drop off until 60dpf, by which time the majority of volumetric growth is complete. This seems to substantiate some combination of the scenarios outlined above: there is an early buildup of CMZ population around 18-30dpf, before CMZ RPCs begin to make their primary contribution to the retina between 30-60dpf, which seems to be characterised by much slower population growth and more rapid volumetric growth, implying steady and elevated specification and niche exit of RPCs.

Subsequent to the bulk of the CMZ buildup and contribution to the retina, the population of the CMZ declines at about 39% the rate of its peak ascent, with an estimated  $-90.1 \frac{\text{cells}}{\text{d}}$  by 90dpf. This thins out the CMZ population to below its initial value, spread out over a much larger peripheral annulus, for a much less dense mature CMZ. Interestingly, the 95% CI on the marginal posterior mean of estimated retinal volume at 180 dpf (ranging from  $1.84\text{e}8 \mu\text{m}^3$  to  $2.75\text{e}8 \mu\text{m}^3$ ) completely encompasses the 95% CI on volume at 90dpf ( $1.96\text{e}8 \mu\text{m}^3$ - $2.48\text{e}8 \mu\text{m}^3$ ), while the 360dpf mean ( $3.85\text{e}8 \mu\text{m}^3$ ) is greater than six standard deviations from the 180dpf mean ( $2.25\text{e}8 \mu\text{m}^3$ ). This suggests a possible second period of quiescence from approximately 90-180dpf, followed by steady contribution to the retina without a buildup in the CMZ population subsequently.



**Figure 4.2: Population and activity of the CMZ over the first year of *D. rerio* life**  
 Panel A: Marginal posterior mean CMZ annulus population. Panel B: Marginal posterior mean retinal volume estimate. Insets in Panels A & B display data from 3-17 dpf. Panel C: Marginal posterior mean of the proliferative index of the CMZ annulus, assayed by specified retinal neurons with incorporated thymidine from an 8hr pulse at the indicated ages. Panel D: Mean daily rate of volumetric increase of the neural retina, calculated as the difference in volumes between two ages over the number of elapsed days. All means are displayed in a band representing the ±95% credible interval for the marginal posterior distribution of the mean. Methods in Section 12.1.8, Section 12.1.9. Code in Section 17.8.9.

Given this description, the ontogeny of the CMZ as a stem cell niche could be well-described by different phases of activity, with different rates of proliferation and specification. It is not immediately clear what sort of periodization is justified by the data. It seems plausible that as few as two phases could explain the data well enough: an initial phase of logarithmic growth, with short cell cycle time

and lower exit rate of RPCs from the CMZ into the specified neural retina, followed by a second phase of decay with longer cycle time and higher exit rate. On the other hand, perhaps some of the data features noted above justify a more complex model that captures, for instance, the initial quiescent period, or the post-180dpf growth of the retina. Because this is a question of how much model structure is justified by our data, we may address it as a model selection problem.

### 4.3 Two-phase periodization of postembryonic CMZ activity by phased difference equation modelling

To perform our model selection task, we simulate the time-evolution of CMZ population and retinal volume. This requires initial values for the size of the simulated CMZ populations and the volumes of the simulated retinae they are associated with. Given the findings presented in Figure 12.1, we are justified in initializing CMZ population and retinal volume by independent samples from the Log-Normal models of their interindividual variability at 3dpf, the end of embryogenesis and the beginning of the period of CMZ-driven retinal growth. To produce new, simulated values of CMZ population and retinal volume, we apply a system of difference equations as follows, where  $pop_n$  is the population at  $n$  dpf,  $CT$  is the mean cell cycle time of the population in hours, and  $\epsilon$  is the proportion of the population at time  $n - 1$  that exits cycle and contributes to the volume of the specified neural retina, and  $\mu_{cv}$  is the mean volume per cell contributed to the retina in  $\mu\text{m}^3$ :

$$p_n = p_{n-1} \cdot 2^{\frac{24}{CT}} - p_{n-1} \cdot \epsilon \quad (4.1)$$

$$v_n = v_{n-1} + p_{n-1} \cdot \epsilon \cdot \mu_{cv} \quad (4.2)$$

A model "phase" can then be defined by the  $CT$  and  $\epsilon$  parameters it applies to update the population and volume, over the appropriate number of days. The full parameterisation of a model with  $p$  phases is given by  $p$  pairs of  $CT$  and  $\epsilon$  values, and  $p - 1$  phase transition dates.  $\mu_{cv}$ , which is taken to apply equally to all phases, is estimated from 3 dpf nuclear measurements, as described in Section 12.1.8. This model is described further in Section 8.2.

Given an initial population and volume sampled from the Log-Normal models of their interindividual distributions, Equation 4.1 and Equation 4.2 are applied to these values to produce simulated sample measurements. Many such samples obtained by Monte Carlo can be used to estimate Log-Normal distributions for simulated CMZ populations and retinal volume, which can be used to score the model against observations. By defining prior distributions over the model parameters, we may sample from the prior to initialize a model ensemble. The ensemble can then be compressed by nested sampling, moving each model-particle over the parameter space by Galilean Monte Carlo, as described in Section 15.2.6. Using typical procedures applied in nested sampling [Ski06], we estimated the Bayesian evidence, maximum a posteriori model output, and marginal posterior distributions on parameters for 2 and 3-phase models.

The maximum a posteriori model output shows the major problem with this simple model; the model relationship between changes in CMZ population and changes in volume breaks down rapidly. That is, the later volume estimates are too large for the CMZ to produce, given the calculated cellular volume ( $\mu_{cv}$ ) at 3dpf. The models fit the early population and volume data better, but the population peak is dragged upward to produce more-likely volume output at later ages. While it is possible that the later

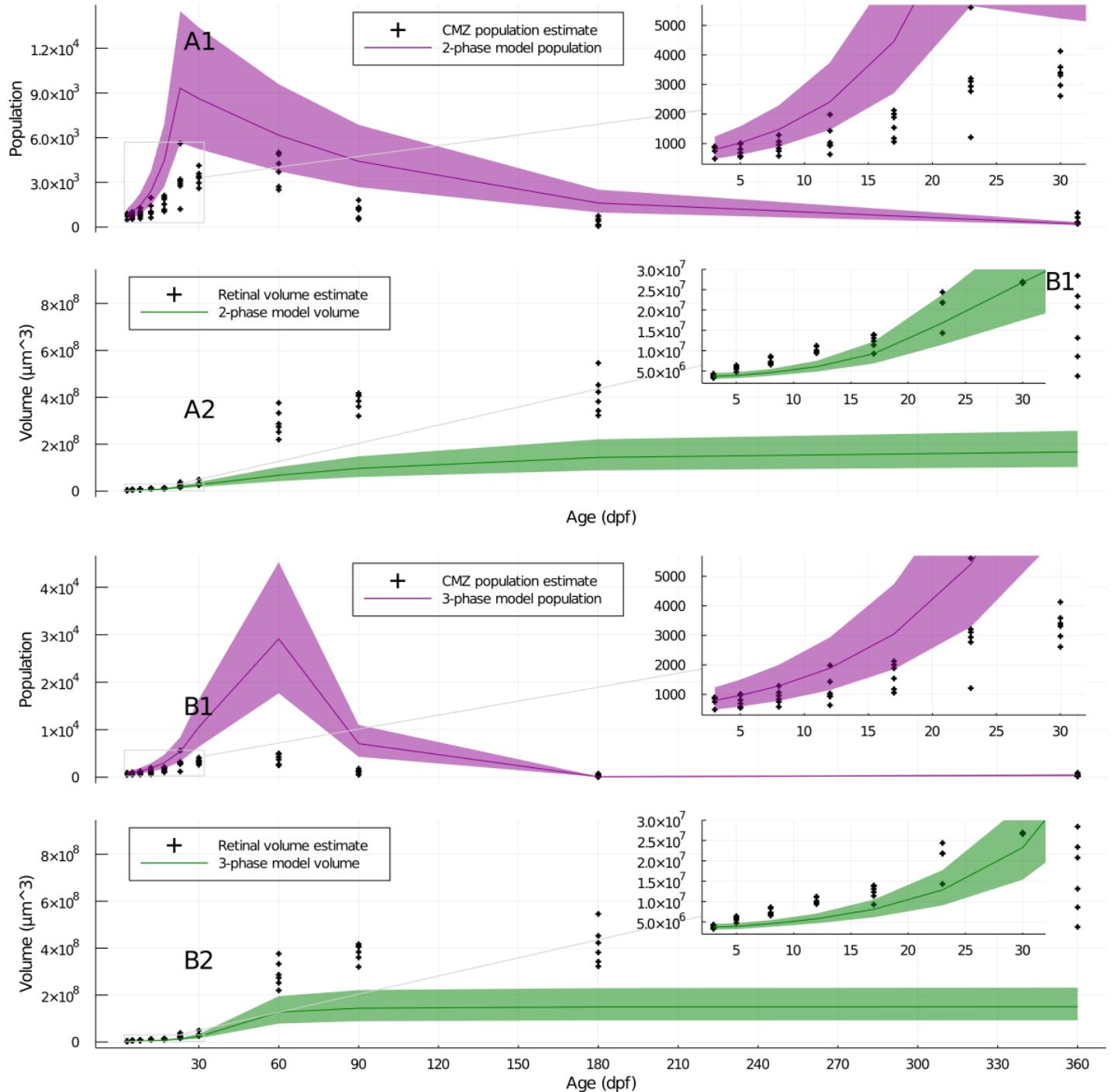


Figure 4.3: Maximum a posteriori output of periodization models

Population and volume estimates from observations (crosses) plotted with mean  $\pm$  95% probability density model output, for the 2-phase model (A panels) and 3-phase model (B panels). A1,B1: population estimates. A2,B2: volume estimates. Insets provide magnified views of data from the first 30dpf. Methods in Section 12.1.11. Code in Section 17.8.8.

CMZ contributes more volume per neuron to the cellular retina, it seems more likely that the volume approximation applies better to more-nearly spherical eyes at younger ages, than to the flattened eyes of later ages. When we tried floating  $\mu_{cv}$  as a variable within the model, the MAP results were similar (data not shown), suggesting that a constant value for  $\mu_{cv}$  across ages is the problem in achieving good model fits, not the particular value chosen. This reinforces the idea that the problem relates to the breakdown of the retinal volume estimate at later ages.

While this limitation prevents either model from explaining the combined estimate datasets very well, they are in this sense under the same constraint, and so some inference about the number of phases justified by the data is still possible. Evidence estimates for the 2-phase and 3-phase models, given these data, are presented in Table 4.1. There are greater than 3500 orders of magnitude more evidence for the 2-phase model; this result has over 200 standard deviations of significance. This reflects the expanded parameter space in the 3-phase model, which has 8 parameters, compared to the 2-phase models' 5. The additional flexibility afforded by the 3rd phase in fitting the later volume data is unable to overcome the evidentiary penalty associated with the larger parameter space. We conclude that the 2-phase model is a better-justified description of these data.

Table 4.1: Evidence favours a 2-phase periodization of CMZ activity

2-phase logZ	3-phase logZ	logZR	$\sigma$ Significance
<b>-7147.3 ± 9.4</b>	-10743.0 ± 13.0	3596.0 ± 16.0	227.624

logZ: logarithm of  $p(D)$ , the marginal likelihood of the data, or model evidence. logZR: evidence ratio; positive ratios in favour of the 2-phase model. Largest evidence value bolded. Methods in Section 12.1.11. Code in Section 17.8.8.

While the parameter estimates associated with these models are clearly unreliable, they are useful to inspect in for comparison to the simulations to follow. We present the parameterization of the MAP model output displayed above in Table 4.2. Unsurprisingly, the selected 2-phase model begins with a first phase of rapid proliferation, with a  $CT$  of 14.5 hr, followed by a second, slower phase of 17.5 hr. The imputed exit rate  $\epsilon$  is greater than 200% of the day's starting population in the first phase, suggesting that new cells exit the CMZ after about 12 hours, around one cycle, while the second phase exit rate is lower, with 160% of the day's starting population exiting the CMZ, again suggesting a residency time of about one cycle. The imputed phase transition age is about 23dpf. Due to the volume estimate problem noted above, it is reasonable to believe that the  $CT$  estimates are likely too short, the  $\epsilon$  estimates too high, all favoured in order to produce higher volume estimates at later ages.

Because nested sampling produces weighted samples from the posterior, we can estimate posterior distributions on parameters. We did so by kernel density estimation, to investigate whether the marginal posterior distributions for the selected model are polymodal. This can reveal how the evidence supports multiple hypotheses about the parameters of the two imputed phases. Kernel density estimates (KDEs) for marginal posterior distributions on the 2-phase models' parameters are presented in Figure 4.4.

These estimates reveal monomodal posterior distributions over the parameters.  $CT$  posteriors are well constrained to a small part of the sampling space (left panels, X axes, and top right panel), with the MAP parameter estimates given in Table 4.2 falling within the most dense regions. However, it is evident that the data do not constrain phase exit rates  $\epsilon$  (left panels, Y axes) very well. That is, the marginal posterior probability distributions for these parameters are very widely spread. The marginal posterior distribution on the age at which the transition between phases occurs is plotted in the bottom

Table 4.2: Maximum a posteriori parameter estimates for periodization models

Parameter	2-phase MAP	3-phase MAP
Phase 1 $CT$ (h)	14.5	14.9
Phase 1 $\epsilon$	2.02	1.95
Phase 2 $CT$ (h)	17.5	75.6
Phase 2 $\epsilon$	1.6	0.29
Phase 3 $CT$ (h)	NA	68.4
Phase 3 $\epsilon$	NA	0.27
Transition 1 age	23.2	47.9
Transition 2 age	NA	179.4

$CT$ : cycle time.  $\epsilon$ : niche exit rate. Methods in Section 12.1.11. Code in Section 17.8.8.

right of Figure 4.4; the KDE suggests that the age of phase transition is likely around 23 dpf.

The lower exit rate associated with MAP second phase highlights the fact that the relative cycle time or exit rate between phases is irrelevant to the overall behaviour of the system. What matters is whether the cycle time and exit rate within a phase produce a growing or shrinking CMZ. We conclude that, while our global model of CMZ population and volumetric retinal contribution is too flawed to make good parameter estimates, a 2-phase model of this activity is better substantiated by the evidence than a 3-phase model. We proceed on this basis, tentatively accepting the 2-phase model, and taking up the idea of the "slice model" introduced in Section 3.3.1, to investigate modelling the CMZ RPC population more concretely, directly from sectional observations, rather than from the calculated population and retinal volume estimates presented above.

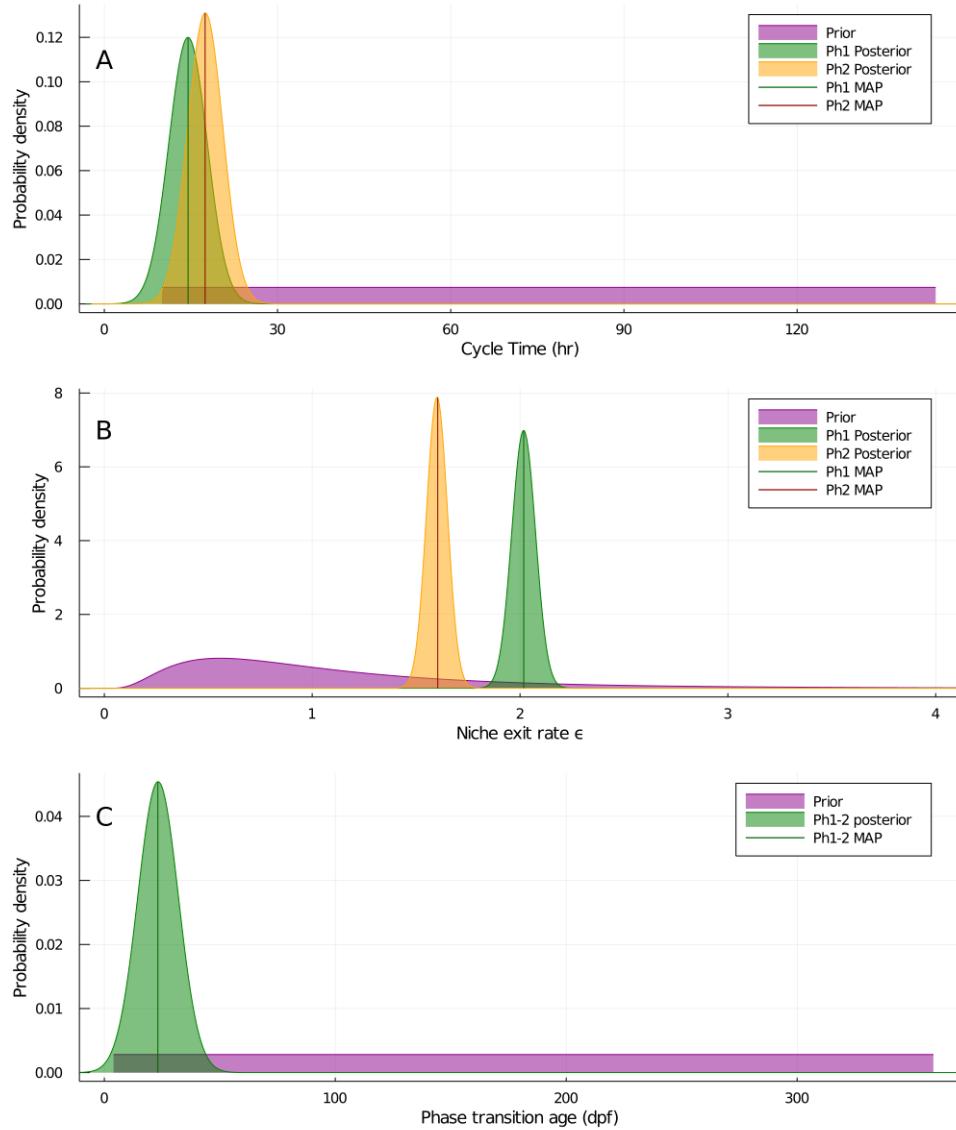


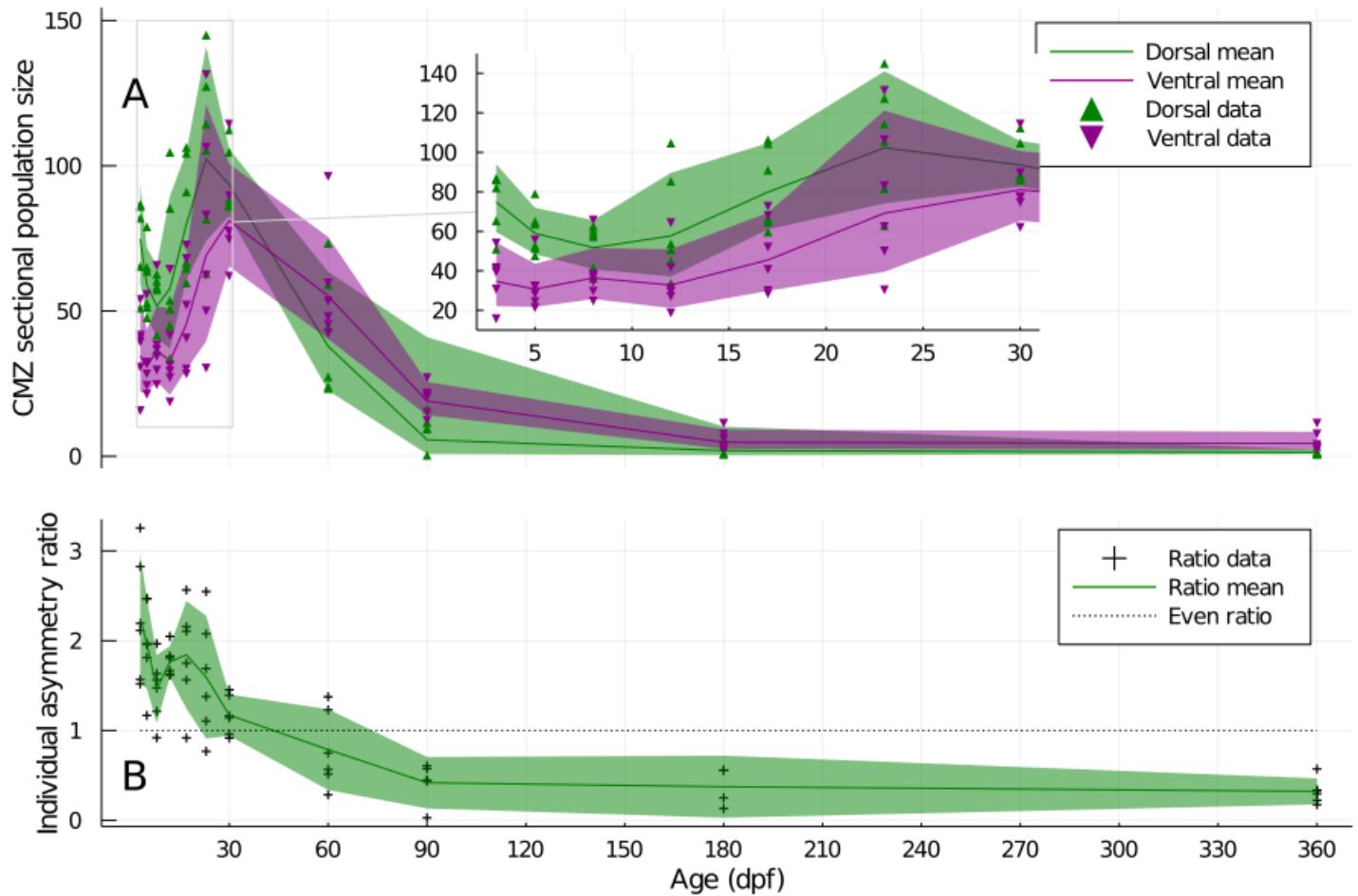
Figure 4.4: **Kernel density estimates of marginal posterior parameter distributions, 2-phase model**

Prior distributions (magenta) apply to parameters of both phases. Maximum a posteriori model parameterisation indicated by vertical lines. Panel A: Marginal posterior distributions on Phase 1 and 2  $CT$  cycle time parameters. Panel B: Marginal posterior distributions on Phase 1 and 2  $\epsilon$  niche exit rate parameters. Panel C: Marginal posterior distribution on Phase 1 to 2 transition age parameter. Methods in [Section 12.1.12](#). Code in [Section 17.8.8](#).

## 4.4 Phased slice-models of CMZ population dynamics suggest subtle shifts across stable-population contour

In the course of the preceding investigations, it became apparent that the CMZ population asymmetry mentioned in [Chapter 3](#) was not a static phenomenon, with the dorsal lobe of the CMZ annulus being consistently more populous than the ventral lobe, as generally implied by the sources covered in [Chap-](#)

ter 1. Rather, both the extent and orientation of asymmetry change over time. Sectional population totals for the dorsal and ventral CMZ are presented in Figure 4.5, Panel A, alongside the related intra-individual asymmetry ratio in Panel B. The initially pronounced dorsal population and reduced ventral population both seem to go through the overall boom-bust progression of CMZ population, but their relative proportion within individuals reverses itself over the period from 17-90dpf. We also observed a similar phenomenon occurring across the naso-temporal axis over the same time period (Figure 12.3).



**Figure 4.5: Developmental progression of dorso-ventral population asymmetry in the CMZ.** Marginal posterior distribution of mean dorsal (D) and ventral (V) population size in  $14\mu\text{m}$  coronal cryosections (panel A) or intra-individual D/V count asymmetry ratio (panel B),  $\pm 95\%$  credible interval,  $n=5$  animals per age. Data points represent mean counts from three central sections of an experimental animal's eye. Methods in Section 11.1.2.1. Code in Section 17.8.4.

Inspected closely, these data provide a possible rationale for the reversal of asymmetry in the proliferative dynamics of the niche itself: the sectional (or “slice”) population of the dorsal CMZ is increasing beyond its postembryonic minimum by 12dpf, while the ventral CMZ takes until 17dpf to exhibit a noticeable increase in size; moreover, the peak dorsal population is achieved by 23dpf, whilst ventrally the peak is only achieved at 30dpf. This suggests that the dorsal and ventral CMZ populations may undergo similar, time-shifted processes of proliferation from different starting populations. If this is so,

an explanation for this time-shifted phenomenon could have fundamental relevance to predicting and controlling the proliferative behaviour of peripheral RPCs and stem cells.

To test this hypothesis, we used a “slice model” of the CMZ, where the thickness of the slice is taken to be the same as the observed cryosection thickness (14 µm). The population of the CMZ is modelled with a difference equation, as above, but with an additional exit term representing lateral, circumferential contributions of the CMZ to the generation of new, adjacent “slices”. The value of this term is calculated from the difference in CMZ annulus diameter over the calculated time period, as implied by a power-law model of lens growth fitted to observations, discussed in [Section 11.1.4](#). The resultant difference equation is [Equation 4.3](#). Terms are as defined above, except that  $p_n$  is the sectional population at  $n$  dpf, and not the total CMZ annulus population; additionally,  $\eta$  is defined as the daily circumferential exit rate implied by the power-law model. The model is detailed in [Section 8.3](#).

$$p_n = p_{n-1} \cdot 2^{\frac{24}{CT}} - p_{n-1} \cdot \epsilon - \eta \quad (4.3)$$

We reasoned that, if the phase transition occurs earlier in the dorsal CMZ than in the ventral CMZ, there should be some informational gain in separating these observations vs. both slices being governed by one model. We therefore estimated the evidence, MAP, and posterior marginals for 2-phase models given the sectional dorsal and ventral populations, both combined and as separately-parameterised slices. To focus on the most informative subset of the data for our hypothesis, we restricted this analysis to the population data within the first three months of life.

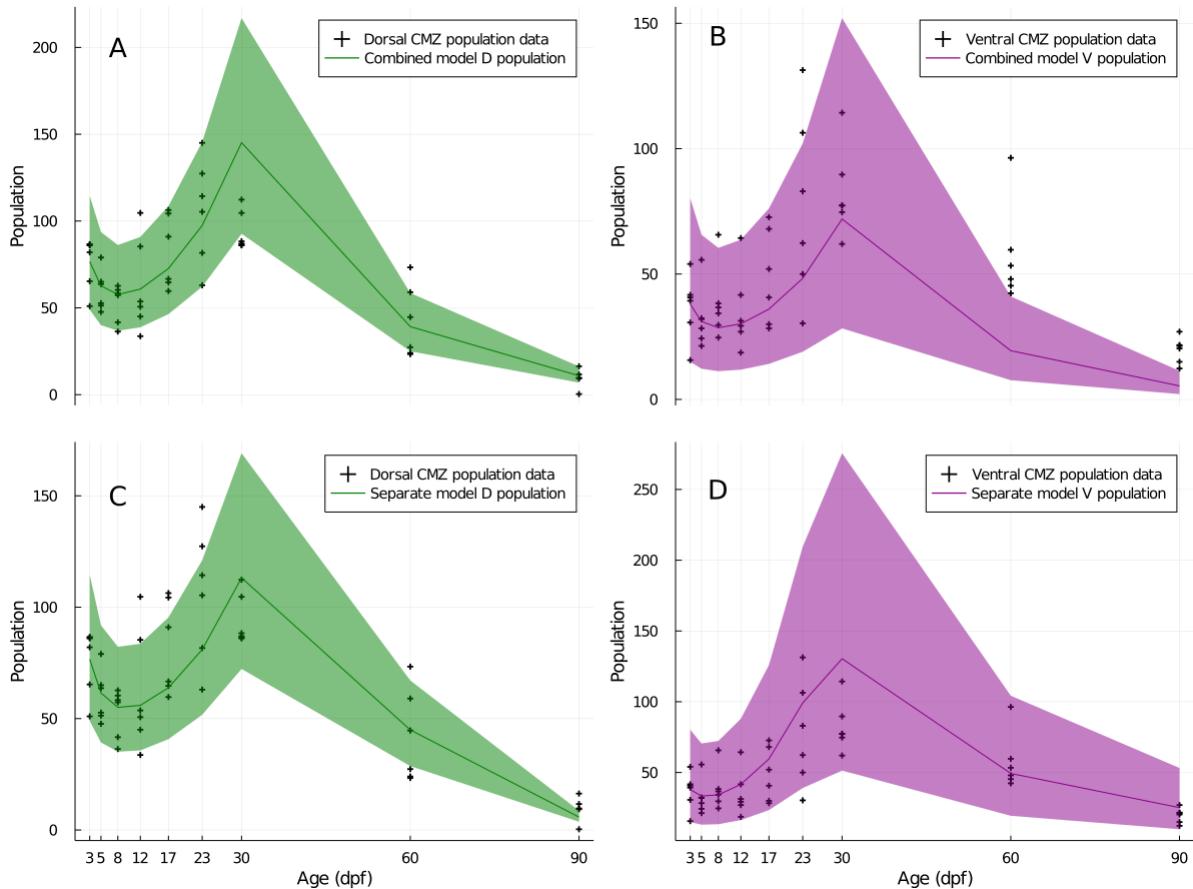
The slice model proves to have greater success at explaining sectional counts than the whole-eye model does at explaining the annulus population estimates; maximum a posteriori model output is presented in [Figure 4.6](#). In particular, all of the models adequately represent the early decline in sectional populations, arising from rapid early growth of the eye that exceeds the CMZs’ proliferative capacity, without further ado; there is no justification for a separate early quiescent period, as suggested above.

The hypothesis that there is a time-shift in the phase transition across the dorso-ventral axis is not supported by these models. First, the evidence estimates demonstrate that we are not justified in separating the dorsal and ventral populations. The total-population slice model receives greater than 500 orders of magnitude more evidence than the joint evidence for the separate dorsal and ventral models, with greater than 20 standard deviations of significance, as displayed in [Table 4.3](#). It is interesting to note that the evidence for the dorsal model ( $-579.9 \pm 0.9$ ) is substantially less than for the ventral model ( $-328.9 \pm 0.5$ ). This may indicate a causal influence on the dorsal population that is neither in the model nor acting on the ventral population, or it may be an uninformative sampling effect. A second indication that this hypothesis is unsupported are the MAP model parameter values, summarized in [Table 4.4](#). The dorsal MAP transition is actually later than the ventral date, which further suggests the original model-idea of a time-shifted late ventral phase change is unsupported by these data.

**Table 4.3: Evidence favours a combined slice model over separate dorsal and ventral models**

Combined D/V model logZ	Separate D/V model logZ	logZR	$\sigma$ Significance
<b><math>-874.3 \pm 1.2</math></b>	$-908.8 \pm 1.0$	$34.5 \pm 1.6$	21.9

logZ: logarithm of  $p(D)$ , the marginal likelihood of the data, or model evidence. logZR: evidence ratio; positive ratio in favour of the combined model. Largest evidence value bolded. Methods in [Section 12.1.11](#). Code in [Section 17.8.5](#).



**Figure 4.6: Maximum a posteriori output of total, dorsal, and ventral CMZ slice models**  
 Population and volume estimates from observations (crosses) plotted with mean  $\pm$  95% probability density model output, for the total CMZ population model (panels A & B), dorsal CMZ population model (panel C), and ventral CMZ population model (panel D). Methods in Section 12.1.11. Code in Section 17.8.5.

The MAP parameters for the combined slice model suggest that those found for the 2-phase MAP in Figure 4.4 overstate cell cycle speed and exit rate. The combined slice model gives  $CT$ s of 34.4 and 23.3 hours for the first and second phases, compared to 14.5 and 17.5 estimated for the whole-eye model. Moreover, the niche exit rates  $\epsilon$  are estimated at .54 and 1.07 for the combined slice model, compared to 2.02 and 1.6 for the whole-eye model. The values of the slice model seem more realistic in general; it seems unlikely that over one day, fully 200% of the day's starting CMZ population has exited the niche and entered the specified neural retina, for instance, as implied by the first phase exit rate of the whole-eye model. As well, given that  $CT$  represents an average cycle time over a heterogenous population of progenitors, many of whom are becoming postmitotic,  $CT$  values over 24 hr seem much more plausible than the 15 hr parameter required for the whole-eye model to produce its unrealistically exaggerated population.

The marginal posterior distributions of the total slice model, presented in Figure 4.7, are substantially more polymodal than those in Figure 4.4. This illustrates how data can support multiple hypotheses about model parameters to different degrees. When the phase marginal  $CT$  and  $\epsilon$  are plotted together,

Table 4.4: Maximum a posteriori parameter estimates for slice models

Parameter	Combined MAP	Dorsal MAP	Ventral MAP
Phase 1 $CT$ (h)	34.4	33.6	17.8
Phase 1 $\epsilon$	0.54	0.57	1.43
Phase 2 $CT$ (h)	23.3	35.0	11.2
Phase 2 $\epsilon$	1.07	0.67	3.42
Transition age	32.2	41.0	27.8

$CT$ : cycle time.  $\epsilon$ : niche exit rate. Methods in Section 12.1.11. Code in Section 17.8.5.

they display the characteristic density curves displayed in Panels A and B of Figure 4.7. This supplies the most interesting information available by estimating this slice model. Firstly, the posterior marginal distributions of the first and second phase are very close to one another. Importantly, when the  $CT$ - $\epsilon$  contour which gives a stable population is plotted (dotted lines), it becomes obvious that the bulk of the first phase posterior lies just under this curve, while the bulk of the second phase lies just above it. This strongly suggests that the population dynamics of the CMZ may be characterised by a relatively subtle shift across the stable population contour, rather than distinct phases.

It should be noted that the MAP estimate does not occupy the most credible KDE kernel. An important property of nested sampling is that the accuracy of evidence calculations is traded off against the accuracy of estimating the posterior distribution [Spe19], as noted in Section 15.2.6. Since we have here prioritized evidence estimation, this is one cause of this discrepancy; the MAP models themselves may have relatively little weight in the KDE estimates. The oversampling of the prior periphery noted in Chapter 7, as well as the very significant polymodality of the model posteriors displayed in Supplementary Figure 12.4 also contribute to this problem<sup>1</sup>.

On the basis of the marginal posterior analysis of this model, we suspected that separately parameterised phases is not the best available description of the data. In particular, the relatively small shifts across the balanced population contour implied by the posterior distributions argues strongly for a model with one phase with a slowly decaying cycle rate.

---

<sup>1</sup>A more conventional approach to reporting posterior estimates is to provide the mean and variance of the distribution, rather than the position of the most likely model in the MAP ensemble. We have eschewed this because of the tendency of GMC\_NS to oversample the prior volume.

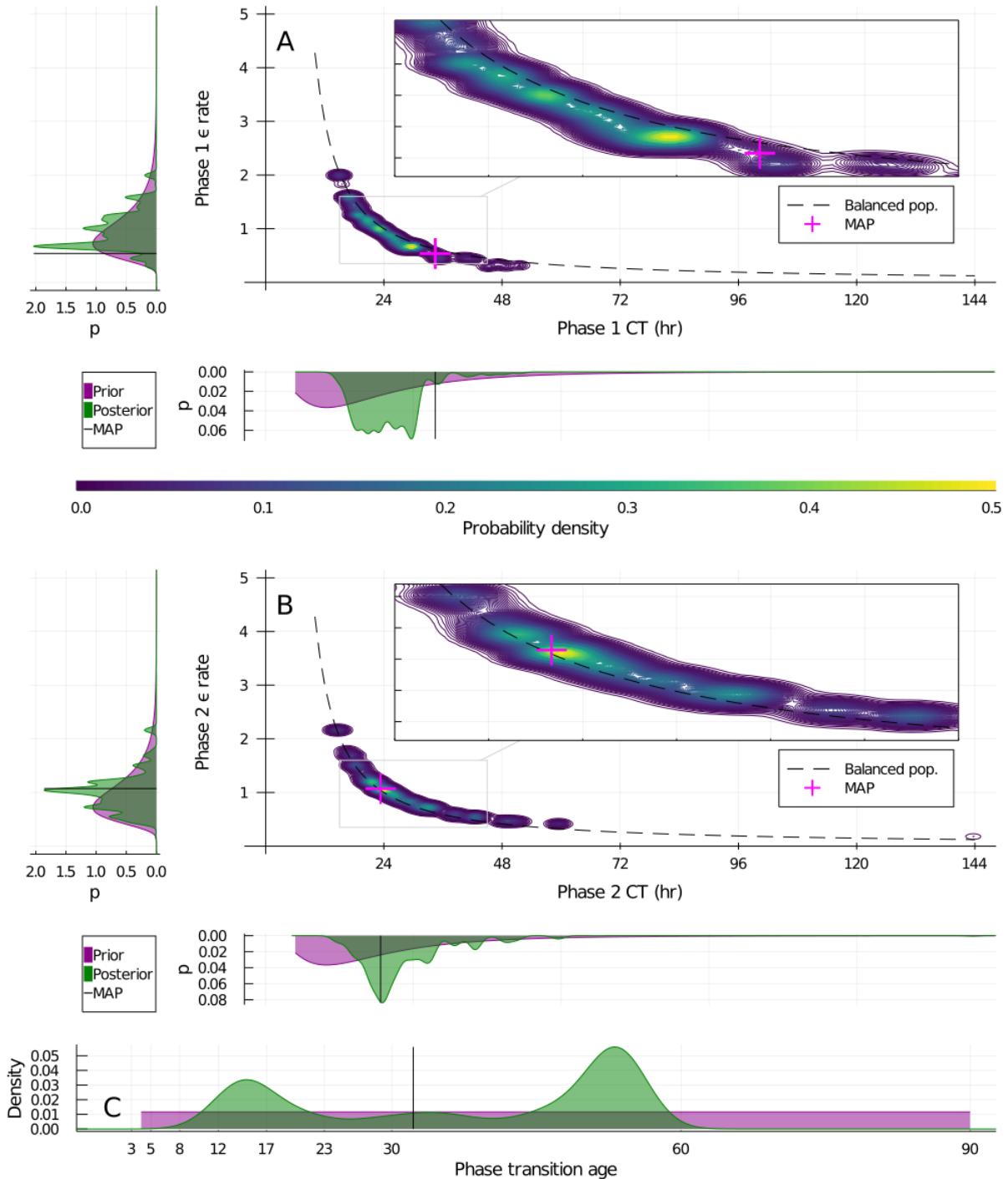


Figure 4.7: Kernel density estimates of marginal posterior parameter distributions, total slice model

Panel A: Bivariate KDE of Phase 1 posterior  $CT$  vs.  $\epsilon$  density. Univariate marginals, with priors, are plotted to the left and below of the bivariate panel.

Panel B: Bivariate KDE of Phase 2 posterior  $CT$  vs.  $\epsilon$  density. Univariate marginals, with priors, are plotted to the left and below of the bivariate panel.

Panel C: Univariate KDE of Phase 1-2 transition age.

Line height (univariate panels) or color scale (bivariate panels) indicates the estimated marginal posterior density present at the indicated parameter value. MAP parameters are given with magenta crosses (bivariate panels). Methods in Section 12.1.12. Code in Section 17.8.5.

## 4.5 Slice models of decaying cell cycle support anatomical homogeneity of RPC behaviour

To model the CMZ as a population of cells with a continuously lengthening cell cycle time, we modify the slice model described above by removing the phases and introducing an exponential decay equation for cycle time. This results in a slice model with three parameters: the initial cycle time ( $CT_i$ ), a constant exit rate ( $\epsilon$ ), and a decay constant ( $\kappa$ ). Population values are updated daily as per Equation 4.3, including the use of the lens model, except that the cycle time  $t$  days after initialising is determined by  $CT_t = CT_i \cdot e^{\kappa \cdot t}$ . More detail is provided in Section 8.4

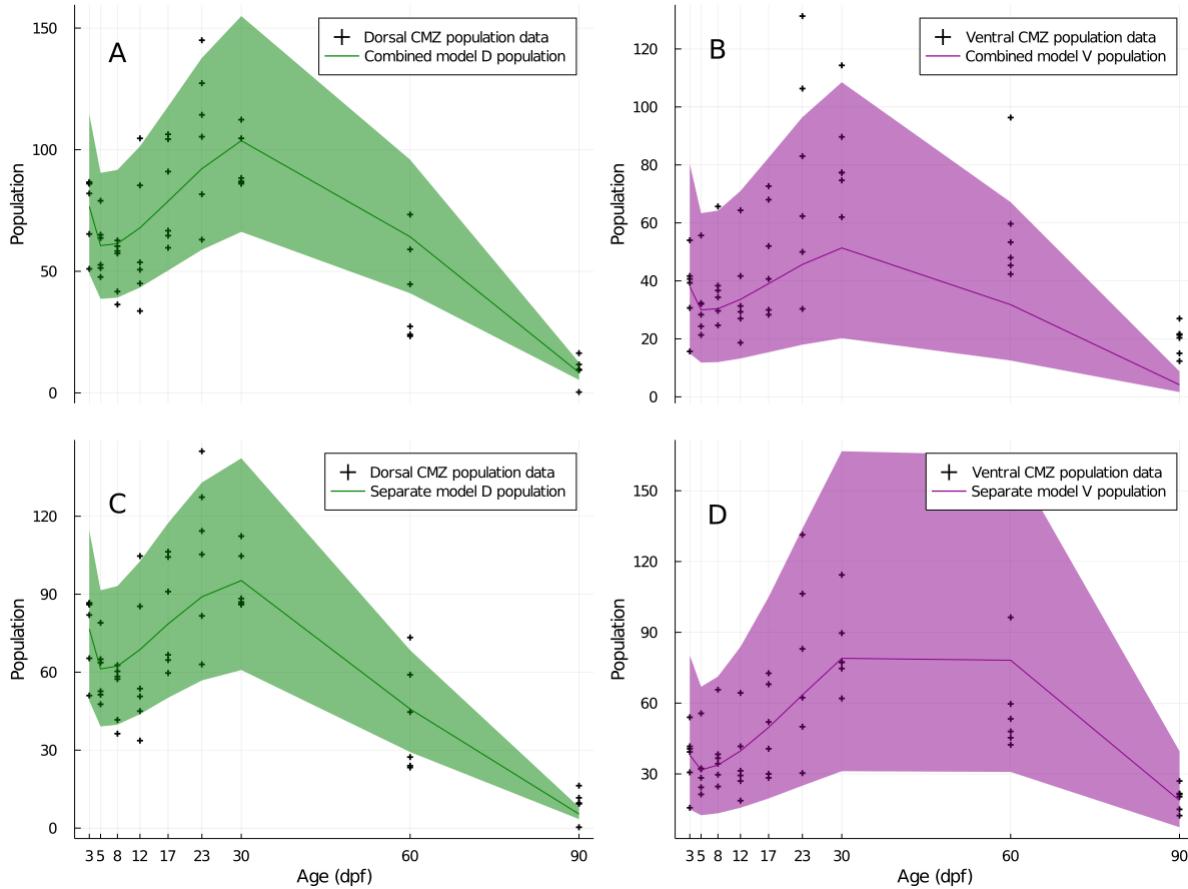


Figure 4.8: **Maximum a posteriori output of total, dorsal, and ventral CMZ decay models**  
Population and volume estimates from observations (crosses) plotted with mean  $\pm$  95% probability density model output, for the total CMZ population model (panel A & B), dorsal CMZ population model (panel C), and ventral CMZ population model (panel D). Methods in Section 12.1.11. Code in Section 17.8.3.

We performed the same sampling routine on the same 3–90dpf dataset used for the earlier slice models, to enable direct comparison between them. The evidence calculations are presented in Table 4.5. The combined decay model substantially outperforms the combined slice model selected above, as well as separate decay models. Therefore, the model evidence supports the use of the decay model for describing RPC behaviour in general; however, there is insufficient evidence to distinguish between dorsal and

ventral populations using this model.

Table 4.5: Evidence supports separate dorsal and ventral decay models

Combined slice logZ	Combined decay logZ	D/V decay logZ	decay logZR	$\sigma$ Sig.
-874.3 $\pm$ 1.2	<b>-762.6 <math>\pm</math> 0.7</b>	-794.6 $\pm$ 0.8	32.0 $\pm$ 1.1	29.3

logZ: logarithm of p(D), the marginal likelihood of the data, or model evidence. logZR: evidence ratio between separate D/V decay models and a combined decay model; positive ratio in favour of the combined model. Combined slice model from Table 4.3 included in left column for comparison. Largest evidence value bolded. Methods in Section 12.1.11. Code in Section 17.8.3.

MAP model output for the combined and separate decay models is presented in Figure 4.8, while the MAP parameter estimates are summarised in Table 4.6. Generally speaking, the values estimated for the parameters of the decay model suggest a much gentler proliferative schedule for RPCs, with initial cycle times measured in days, and daily niche exit rates of between 15-30% of the RPC population. Cumulative thymidine analogue labelling results, presented below, suggest these values are unrealistically long. Interestingly, MAP parameterisations for the split Dorsal and Ventral models do not imply that dorsal RPCs are more active than those found ventrally. Probably, if consistent regional biases in CMZ RPC behaviour exist, better data would be required to constrain niche exit rate. Indeed, it seems likely that the lack of effective constraint on the model's niche exit rate  $\epsilon$  allows for the unrealistically long  $CT_i$  suggested by both combined and separate decay models. We conclude that explanations invoking differences in CMZ RPC behaviour across morphological axes cannot be supported with our models and datasets, indicating that morphological homogeneity of these parameters is descriptively adequate at this level.

Table 4.6: Maximum a posteriori parameter estimates for decay models

Parameter	Combined MAP	Dorsal MAP	Ventral MAP
$CT_i$ (h)	52.0	69.7	62.1
$\kappa$ decay	0.0067	0.0124	0.0091
$\epsilon$ exit	0.276	0.164	0.189

$CT_i$ : Initial cycle time.  $\kappa$ : Exponential cycle time decay constant.  $\epsilon$ : niche exit rate. Methods in Section 12.1.11. Code in Section 17.8.3.

Marginal distributions on these parameters are summarised in Figure 4.9. We again note that the data are informative for parameters of cell cycle ( $CT_i$  and  $\kappa$ ), but give little information regarding niche exit rate. The MAP models found by GMC are well outside the kernel density estimates of the posterior distributions supplied by nested sampling this model. It is not clear why this should consistently be the case. The decay model may be characterised by small, high credibility modes that are outside the bulk of the less credible posterior mass; possibly, these are underweighted if they come late in the nested sampling process. In any case, the posterior marginals seem to confirm that there is little reason to separately distinguish dorsal and ventral subpopulations of CMZ RPCs.

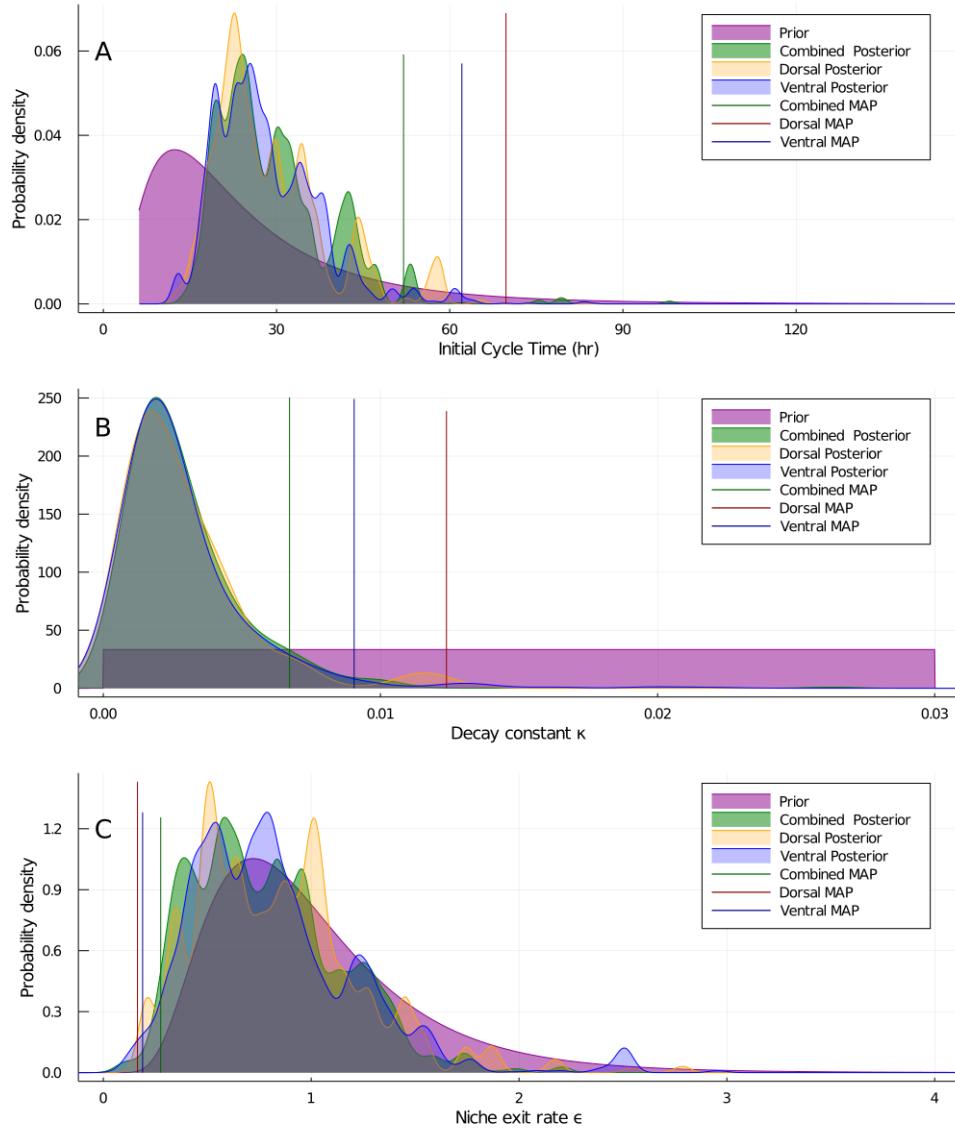


Figure 4.9: Kernel density estimates of marginal posterior parameter distributions, decay models

Panel A: Marginal prior and posterior distributions on  $CT_i$ . Panel B: Marginal prior and posterior distributions on  $\kappa$ . Panel C: Marginal prior and posterior distributions on  $\epsilon$ .

MAP model parameters are given by vertical lines. Methods in Section 12.1.12. Code in Section 17.8.3.

## 4.6 Bayesian inference of cycle parameters from cumulative thymidine labelling experiments supports slice model cycle time estimates

We sought to confirm the plausibility of the  $CT$  estimates obtained from modelling CMZ population measurements with an independent line of evidence. We examined 3dpf CMZ RPCs cumulatively labelled with a 10.5 hour pulse of the thymidine analogue EdU [CMD<sup>+</sup>09]. We used the Empirical Bayes approach

to estimate the evidence for separate Nowakowski-style [NLM89] linear models for the dorsal and ventral CMZ, against a model for both subpopulations combined. While this model is inadequate for reasons outlined in Chapter 2, it can serve to substantiate the first phase cycle time parameter. These results are summarized in Table 4.7, with the relevant linear regressions displayed in Supplementary Figure 12.5.

Table 4.7: Evidence favours whole-CMZ linear cycle models over separate D/V models

Model	Implied $T_c$ (hr)	Implied $T_s$ (hr)	$\log Z$
Dorsal	$14.7 \pm 1.6$	$1.38 \pm 0.76$	7.778
Ventral	$14.0 \pm 1.2$	$0.8 \pm 0.58$	15.202
Combined	$14.6 \pm 1.1$	$1.25 \pm 0.53$	<b>26.165</b>

$T_c$ : calculated cell cycle time.  $T_s$ : calculated S-phase length.  $\log Z$ : logarithm of  $p(D)$ , the marginal likelihood of the data, or model evidence. Largest evidence value bolded. Methods in Section 12.1.10. Code in Section 17.8.12.

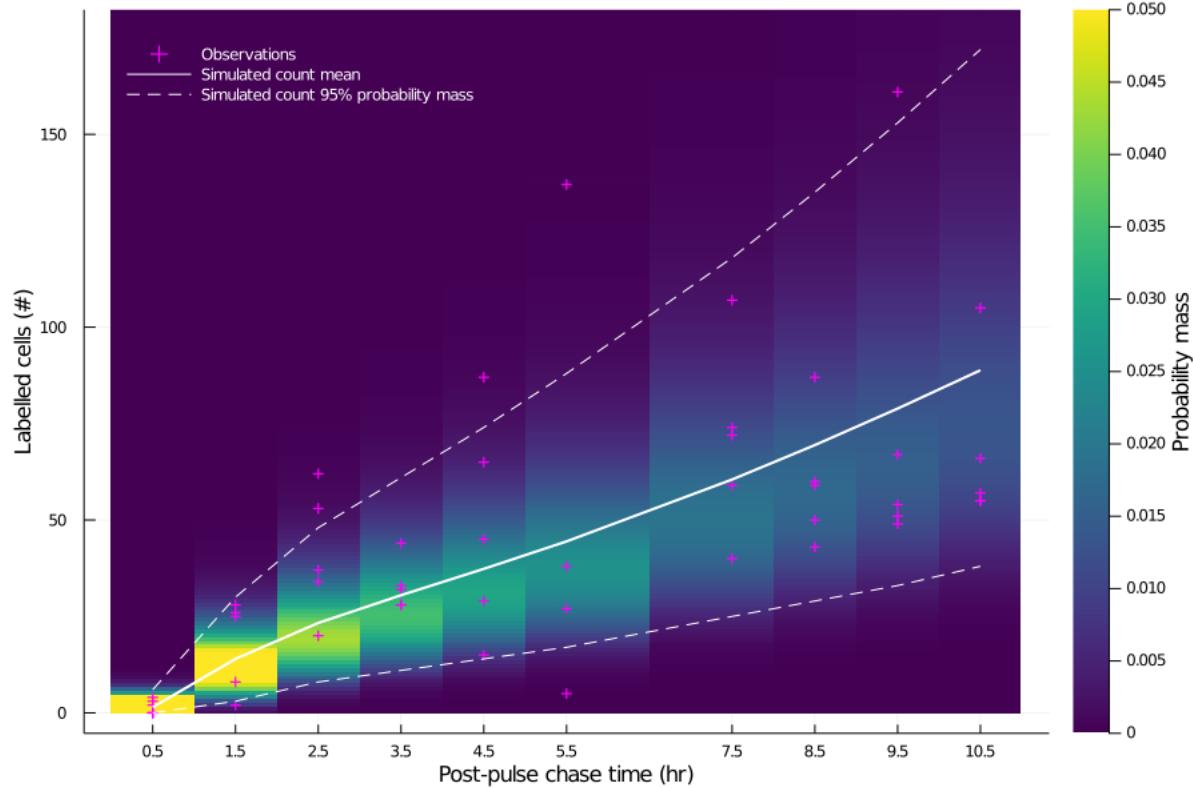
Firstly, there are approximately 3 orders of magnitude of evidence in favour of the combined model vs. the joint evidence for separate models (ie. 26.165 vs. 22.980). In general, this suggests that there is little traction to be gained by separating these observations along the D/V axis. The Nowakowski-calculated cell cycle time  $T_c$  for the combined model,  $14.6 \pm 1.1$  hr, includes the whole-eye MAP first phase  $TC$ , 14.5 hr, within one standard deviation. This suggests the Nowakowski model has similar problems to the whole-eye models presented in Figure 4.3, substantially overstating cycle rate. Calculated S phase lengths are also unrealistically short. Finally, the data seem to diverge from a linear trend toward the end of the pulse (see Figure 12.5), showing the limitations of this model.

Because the Nowakowski model is clearly inadequate for estimating RPC cycle parameters in the CMZ, we implemented a cell-based cycle simulator, similar to those in Chapter 2, to model discrete cumulative thymidine labelling counts, in lineages with explicitly simulated cycle events. This model is described further in Section 8.6. The MAP model output for this model is presented in Figure 4.10; it does a better job of describing the observations than the Nowakowski method, particularly at the earliest timepoints, due to simulation of cycle time correlation between sibling cells, as well as detector sensitivity.

KDE estimation of marginal distributions over the model parameters are presented in Figure 4.11. These estimates give us a clear picture of what can be learnt from these data; mean cycle time and the length of S phase as a fraction of the cycle time are significantly compressed relative to the prior, but little can be gleaned about the variance of the cycle time distribution, and almost no information is available about the G1 or sister shift fraction. That is, for the parameters which the data offer little information, the posterior distribution correctly collapses back onto the prior, which indicates that this approach adequately copes with unconstrained parameters<sup>2</sup>.

The MAP parameters for the LogNormal distribution of cycle times produces a mean cycle time of 24.0 hr. This seems to be a more realistic estimate of RPC cell cycle time, and tends to support the longer values estimated in the slice models, although it does suggest the 50+ hour  $CT_i$  estimates for the decay models may be excessively long. Furthermore, this result supports our observation in Figure 2.7 that the He model examined in Chapter 2 substantially overstates the per-lineage proliferative rate of

<sup>2</sup>The model is overparameterised relative to the available dataset. This was done deliberately to allow the model to be used with richer data than were available here; however, it does have the effect of making sampling against these observations less efficient than is achievable.



**Figure 4.10: MAP model output and observations for thymidine slice model of 3dpf cumulative EdU labelling**

Count of EdU-positive cells observed in 14 $\mu$ m coronal cryosections through *Danio* CMZs, at indicated chase times, during a 10 mM EdU pulse (magenta crosses), overlaid with MAP model output. Probability mass distribution of the discrete non-parametric output is shown by color scale; yellow values include counts with  $>0.05$  mass. Output mean and 95% mass are indicated by solid and dashed white lines. Methods in Section 13.1.7, Section 12.1.11. Code in Section 17.8.13.

RPCs, at least in the CMZ; the He model's mean cycle time of 6 hours is much too fast to correctly model this population. It is likely that most reports of similar, very short cycle times for embryonic RPCs are underestimates of the actual distribution of cycle times.

Because the thymidine simulator's output resembles observations more closely than the Nowakowski model, produces more realistic parameter estimates than it, and clearly conveys how much information is available in the dataset, we suggest that this explicit simulation approach should be preferred to Nowakowski's method wherever the computational cost can be justified.

## 4.7 The CMZ contributes stably to each cellular layer with time-variable lineage composition

By labelling CMZ RPCs with the thymidine analogue EdU in a pulse at 3, 23, and 90 dpf, followed by histochemical analysis for known zebrafish retinal neural lineage markers after a 7 day chase, we investigated the possibility that RPC lineage outcomes change over the life of the organism. This

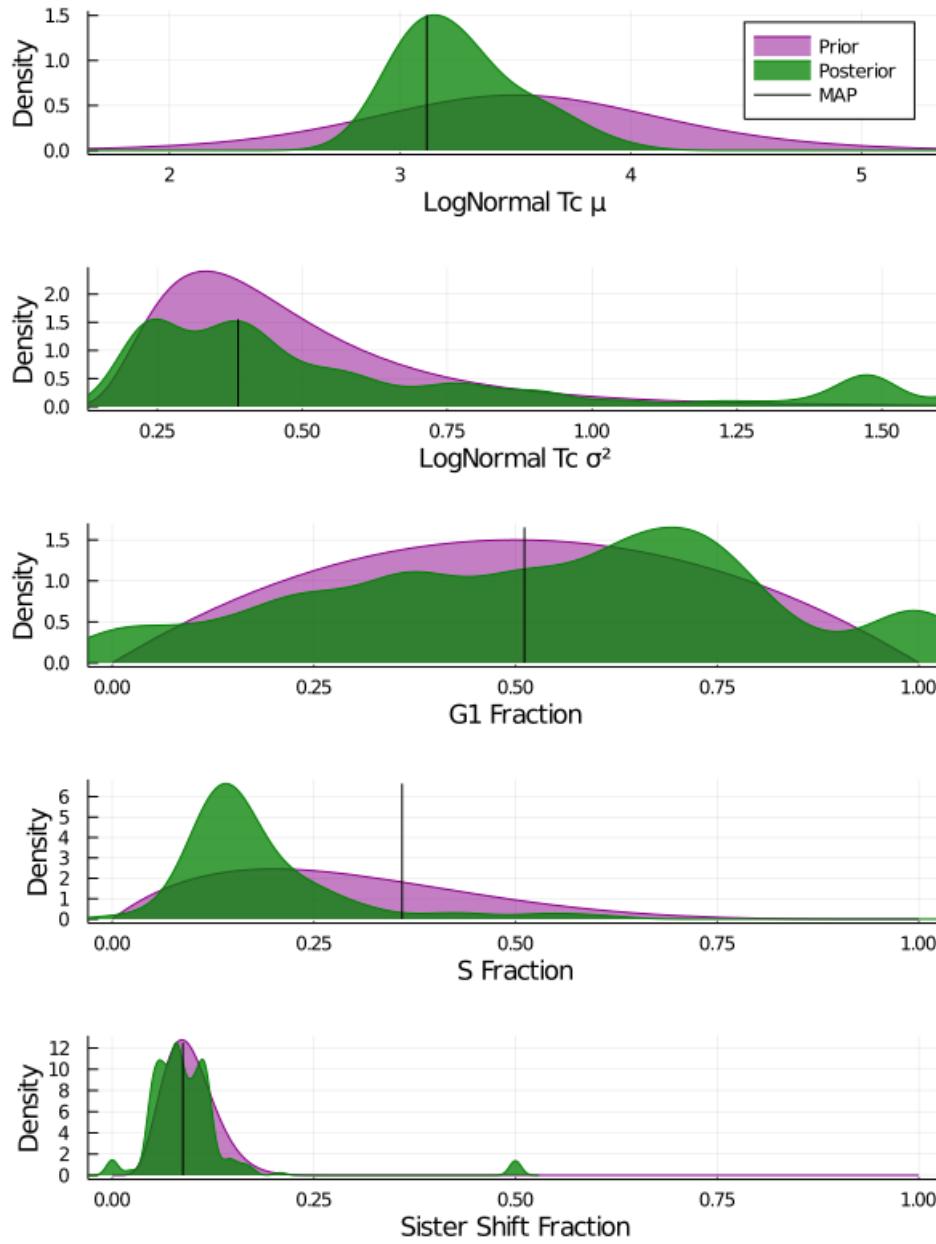


Figure 4.11: **Marginal posterior distributions for thymidine slice model**

Marginal prior (magenta) and posterior (green) distributions on thymidine simulator cycle parameters, noted on the x axis. MAP model parameters are displayed with vertical black lines.

Methods in Section 13.1.7, Section 12.1.12. Code in Section 17.8.13

hypothesis is of particular interest, as differences in the mosaic organisation of embryonically-contributed central retina and CMZ-contributed peripheral retina remain unexplained [ABS<sup>+</sup>10]. It may, moreover, have clinical significance, were quiescent peripheral stem cells to be entrained for regenerative medical purposes, as their lineage outcomes may be different than embryonic RPCs. We used antibodies raised against Pax6 and Isl2b to mark retinal ganglion cells (RGCs) of the ganglion cell layer and amacrine cells

of the inner nuclear layer. Anti-glutamine synthetase (GS) and anti-PKC $\beta$  were used to mark Müller glia (MG) and bipolar cell (BPC) populations of the INL. The unique flattened nuclear morphology of horizontal neurons was used to identify them. Lastly, the antibody Zpr1, directed against an unknown antigen present in photoreceptors with double cone morphology, was used to mark these cells.

Observations were collected in "staining groups", which combined histological markers; representative confocal micrographs from this study in animals pulsed at 23dpf are displayed in Figure 4.12, while data from all ages are plotted in Figure 4.13.

These data take the form of fractions of the thymidine-labelled cohort entering each of the three cellular layers (panel A of Figure 4.13), or the subfraction of the cohort within a given layer expressing a particular cellular marker (Panels B-I). While some variability is apparent in all of the measurements, it is unclear whether it is well-described as time-dependent in most cases. In order to address the question of whether CMZ layer or lineage outcomes differ over time, we needed to assess the joint evidence for separate models of each measurement at each age, against the evidence for a single model of the measurement for all of the assessed ages.

Because it is not obvious that these fractional measurements are better described Log-Normally, as the underlying population counts are, we first assessed the joint evidence for Normal and Log-Normal models of the data, summarized in Supplementary Table 12.4. This supports Log-Normal modelling of all measurements aside from the INL and ONL overall fractions, as well as GCL-resident Pax6- and ONL-resident Zpr1-positive cells. We noted that the evidence estimates differs from the simple likelihood ratio tests in some cases, as summarized in Supplementary Table 12.3. Without a clear fundamental justification for uniformly preferring one model or the other, we selected the best-supported model for each measurement. We estimated the evidence for an age-marginalized model, representing stable contribution to the layer or lineage over time, and compared this to the joint evidence for separate models at each age, representing a time-varying contribution model. These estimates are presented in Table 4.8.

Table 4.8: Evidence supports stable layer contributions with time-varying fate commitment

Layer	Marker	Cell type	Stable logZ	Time-vary logZ	logZR	$\sigma$ sign.
GCL	Cohort	All GCL cells	<b>-0.84 ± 0.71</b>	-30.96 ± 1.0	30.1 ± 1.2	24.7
GCL	Isl2b	RGC	-48.68 ± 0.26	<b>-23.76 ± 0.59</b>	-24.92 ± 0.64	39.0
GCL	Pax6	Displaced am.	<b>-23.85 ± 0.44</b>	-40.08 ± 0.71	16.24 ± 0.84	19.3
GCL	Isl2b/Pax6	RGC subtype	-32.97 ± 0.28	<b>-16.82 ± 0.83</b>	-16.14 ± 0.87	18.5
INL	Cohort	All INL cells	<b>-41.15 ± 0.49</b>	-52.65 ± 0.66	11.5 ± 0.82	14.0
INL	Pax6	Amacrine cell	-15.37 ± 0.57	<b>-9.14 ± 0.46</b>	-6.23 ± 0.73	8.5
INL	PKC $\beta$	Bipolar cell	-5.99 ± 0.3	<b>2.81 ± 0.54</b>	-8.79 ± 0.62	14.2
INL	GS	Müller glia	14.34 ± 0.35	<b>10.18 ± 0.6</b>	4.17 ± 0.7	6.0
INL	HM	Horizontal cell	<b>12.0 ± 0.31</b>	3.02 ± 0.6	8.98 ± 0.68	13.3
ONL	Cohort	All ONL cells	<b>-47.42 ± 0.45</b>	-63.19 ± 0.7	15.77 ± 0.83	19.1
ONL	Zpr1	Double cones	<b>-18.79 ± 0.37</b>	-31.63 ± 0.6	12.84 ± 0.7	18.3

logZ: logarithm of p(D), the marginal likelihood of the data, or model evidence. Largest evidence values bolded. logZR: evidence ratio; positive values in favour of stable model. Methods in Section 12.1.4, Section 12.1.11. Code in Section 17.8.10.

These calculations demonstrate that there is evidence for time-varying contributions to the retina, in

particular, to RGC, amacrine, bipolar, and Müller glial fates. All of these findings have between 6 and 39 standard deviations of significance. In the INL, the trend of the posterior mean for PKC $\beta$ -positive bipolar neurons (panel F) and GS-positive Müller glia of these lineages declines from the early postembryonic period (3dpf) to the late juvenile period (90dpf). Most of the time-varying fates show a downward trend in the fraction of the cohort expressing the relevant marker. These time-varying, declining lineages may be functionally related. However, we have no specific evidence that would implicate Isl2b/Pax6 double-positive RGCs in circuits with bipolar or amacrine cells, for instance. This nonetheless provides an explanation for the observation of a more-ordered retinal mosaic in later retinal contributions relative to the embryonic central remnant: since one or more sublineages in each layer are depleted in older fish, a different overall mosaic pattern as the neurons associate and pack together is expected. This may occur by changing the pattern of crystalline defects in this packing order, recently suggested as a mode of mosaic organisation [NNM<sup>+</sup>19]. Based on the timing of these changes, we tentatively ascribe this differential lineage production to the later phase of CMZ contribution, after the balance between proliferation and niche exit has shifted to the latter.

Lastly, we investigated the possibility that there might be detectable differences in layer or lineage contributions across the dorso-ventral axis; we tested this by measuring the evidence for age-marginalized combined models of fractional contribution against age-marginalized models split along the dorso-ventral axis. We found no evidence to support this hypothesis. These results are presented in Summary Table 12.5.

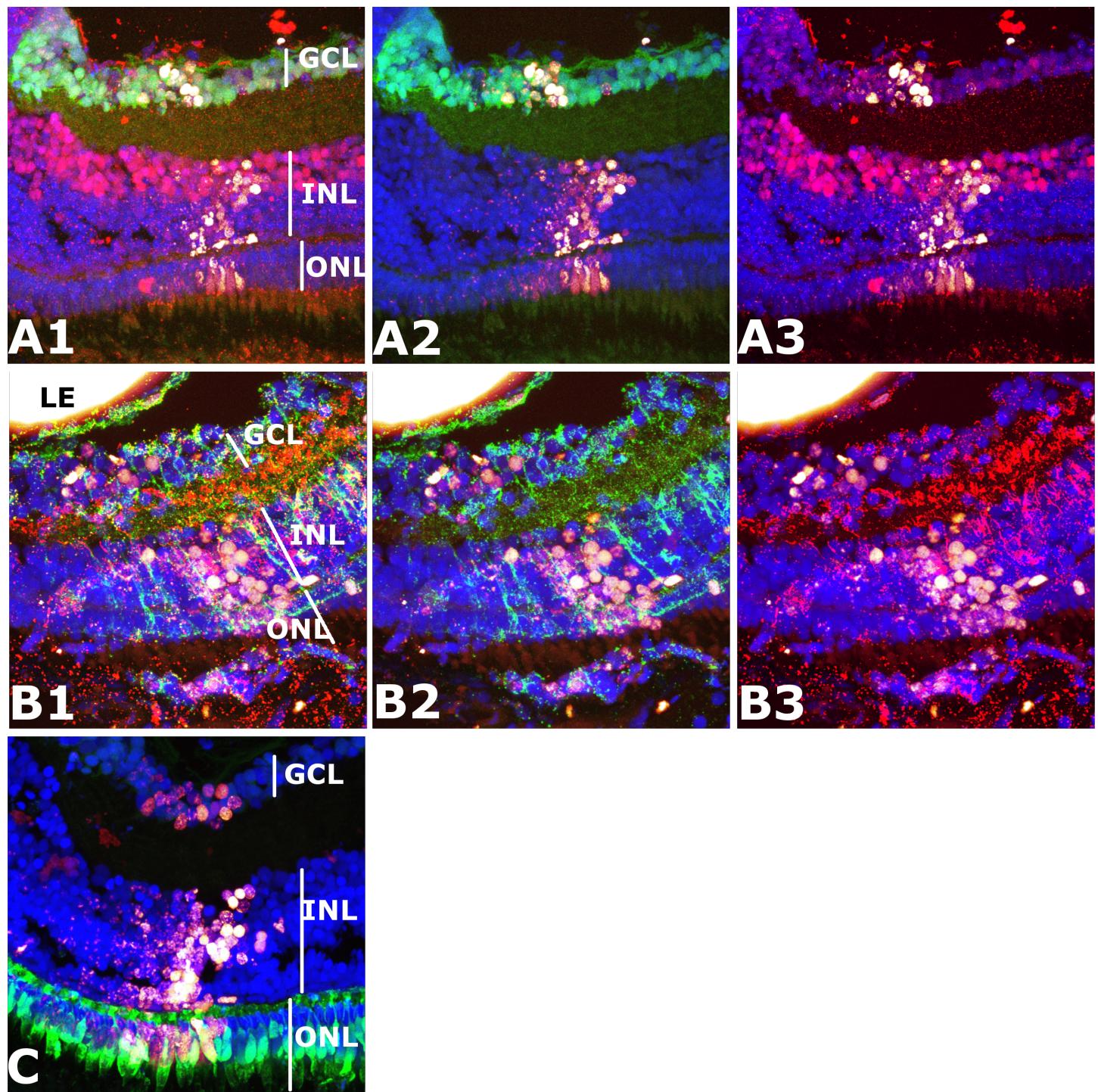


Figure 4.12: Representative 23dpf lineage marker confocal micrographs

Panel A1: RGC/Amacrine staining group. A2: Isl2b channel. A3: Pax6 channel.

Panel B1: MG/BPC staining group. B2: PKC $\beta$  channel. B3: GS channel.

Panel C: Double cone staining group. Zpr1 channel.

GCL: Ganglion cell layer. INL: Inner nuclear layer. ONL: Outer nuclear layer. Methods in Section 12.1.4.

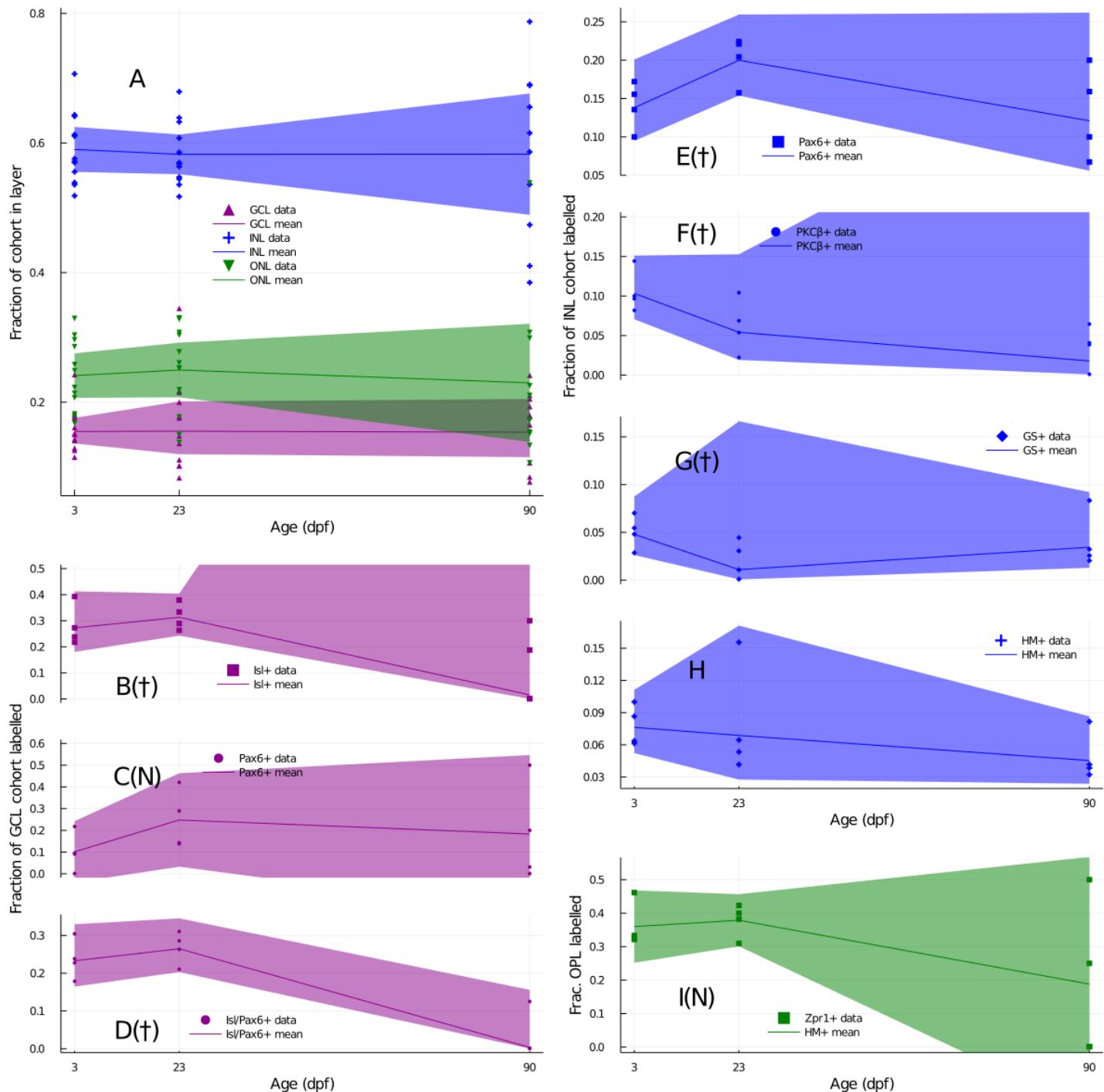


Figure 4.13: Second-phase declines in CMZ-contributed IslPax6+ RGCs, PKC $\beta$ + bipolar neurons, and Zpr1+ double cones

Overall fraction of CMZ-contributed cohort entering cellular layers (Panel A), or fraction of the layer subcohort expressing the noted immunohistochemical marker of lineage. All mean values are presented as marginal posterior means  $\pm$  95%CI.

$\lambda$ : Fractional lineage contribution is modelled Log-Normally

$\dagger$ : Evidence supports time-varying model of fractional lineage contribution

Magenta: GCL measurements; Blue: INL measurements; Green: ONL measurements

Methods in Section 12.1.4, Section 12.1.11. Code in Section 17.8.10.

## 4.8 Early retinal cohorts of the *D. rerio* retina are turned over at a low rate by 4C4-positive microglia

Recently, extensive neural death has been reported in older zebrafish retinae [VGV<sup>+</sup>18]. This was described as a “neurodegenerative pathology” and suggested as a model of age-related neurodegeneration. However, during our thymidine analogue pulse-chase studies, CMZ-contributed cohorts often appeared less numerous only a month or two after their entry into the neural retina, even in juveniles of 30–90 days of age. Moreover, we observed numerous 4C4-positive microglia associating with the CMZ, as displayed in Supplementary Figure 12.7, which suggested that microglia may prune CMZ contributions. If neural retinal turnover occurs throughout the life of the organism, this would belie view that it should be treated as pathological, rather than constitutive. However, the thinning phenomenon could be explained by the changing morphology and geometry of the neural retina during this period. In particular, the neural retina thickens noticeably over this time, as displayed in Supplementary Figure 12.2. Although this increase is due, in large part, to the lengthening of photoreceptor outer segments, the inner nuclear layer is also significantly thickened. It is plausible that this process involves a compaction of the neurons along the coronal axis typically sampled, thus appearing to lose cells over time without this actually occurring.

To investigate, we administered 24 hr pulses of EdU to 1dpf embryos, and followed with 24 hr pulses of BrdU at 23dpf to mark a CMZ-contributed cohort near the height of its activity. By taking both coronal and transverse sections through the eyes of animals at 30, 60, and 90 dpf, we sampled these cohorts from both morphological axes of the retina, and counted labelled sectional totals. Data were totalled across morphological axes to account for the possibility of the cohorts being compacted on the coronal plane and extended on the transverse one. In order to test the hypothesis of early turnover, we estimated the evidence for linearly correlated and uncorrelated models of cohort population over time, using the Empirical Bayes regression method. If the addition of a time-dependent term in the correlated regression model is justified by the evidence at these early ages, this supports the notion that *D. rerio* retinal turnover is a lifelong phenomenon; if not, the apparent contraction of the cohorts is more likely a function of the alternative explanations noted above, or is occurring at too low a rate to be detected by these means. The results are summarized in Table 4.9, with the regressions plotted in Supplementary Figure 12.6.

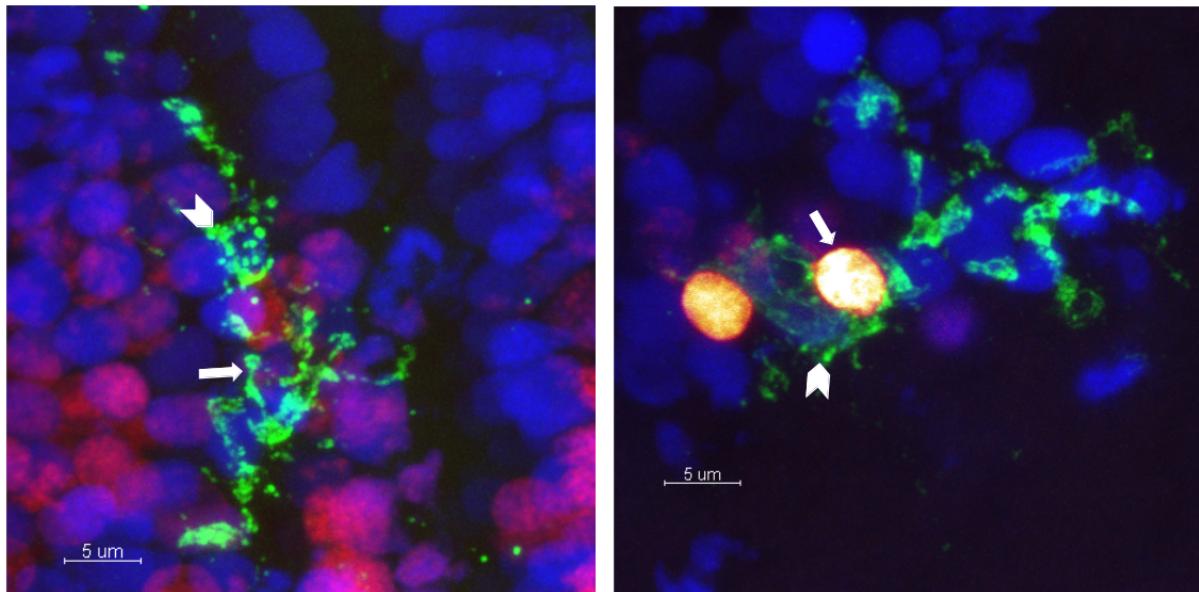
Table 4.9: Evidence for linear regression models supports early cohort population stability

Measurement	Stable logZ	Declining logZ	logZR
1dpf Central Remnant	<b>-212.953</b>	-214.167	1.213
30dpf Cohort	<b>-107.567</b>	-110.827	3.261

logZ: logarithm of p(D), the marginal likelihood of the data, or model evidence. Largest evidence values bolded. logZR: evidence ratio; positive values in favour of stable model. Methods in Section 12.1.3, Section 12.1.10. Code in Section 17.8.15.

There is more evidence for a model of cohort population that is stable over time than one correlated with time, indicating that the additional model complexity implied by allowing turnover is unjustified, for this early period. Despite this, we did find isolated instances where members of these cohorts were visibly being engulfed by 4C4-positive microglia; one such event is depicted in Figure 4.14. Moreover,

we also observed TUNEL-positive nuclei in the central retina of *r3s* siblings in the early postembryonic period (Figure 5.8). This indicates that some level of turnover is occurring during this earlier period. The evidence ratio in favour of the uncorrelated models is not overwhelming, which suggests that the cohorts are not turned over at a high enough rate to be detectable in the early period, although the process does occur even at this early age. In order to confirm this finding, we estimated the evidence for age-marginalized Log-Normal models of the population counts against age-differentiated models, representing time-constant and time-varying population models. This investigation resolves any ambiguity: no time-varying model is justified by the available evidence, as summarized in Table 4.10.



**Figure 4.14: 4C4+ microglia associate with and engulf EdU-labelled CMZ contributions in the specified neural retina**

Representative maximum intensity projections from confocal micrographs of 14 $\mu$ m coronal cryosections through 30dpf zebrafish eyes labelled at 23dpf with EdU.

Blue: Hoechst 33258 nuclear counterstain. Green: Microglia labelled with 4C4 antibody. Red-orange-yellow: Intensity scale of EdU staining, indicating cellular origin in the 23dpf CMZ.

Chevrons: microglial nuclei. Arrows: EdU-positive nuclei found within 4C4-labelled extent of microglial cell body and appendages.

Methods in Section 12.1.5.

**Table 4.10: GMC-NS evidence estimates confirm Empirical Bayes analysis of early cohort stability**

Cohort	Time-constant logZ	Time-varying logZ	logZR	$\sigma$ Significance
Embryonic central remnant	<b>-857.08 ± 0.29</b>	-868.02 ± 0.6	10.94 ± 0.67	16.4
1 month CMZ	<b>-402.38 ± 0.31</b>	-405.76 ± 0.43	3.39 ± 0.53	6.4

logZ: logarithm of p(D), the marginal likelihood of the data, or model evidence. logZR: evidence ratio; positive ratios in favour of the 2-phase model. Largest evidence value bolded. Methods in Section 12.1.3, Section 12.1.11. Code in Section 17.8.14.

## 4.9 Summary: Modelling the postembryonic CMZ

CMZ population dynamics, as encapsulated in our datasets, are best explained by a model of slowly decaying cell cycle rate and constant niche exit rate. This parameterisation is superior to phased models of CMZ activity, although both imply an initial period characterised by CMZ population growth, followed by a second phase of during which the CMZ is depleted. These dynamics can be achieved with a wide range of cycle times and niche exit rates, which we show by establishing posterior distributions on these parameters. These demonstrate that our estimates of cell cycle lengths are better constrained by our data than estimates of exit rates. The population of the CMZ displays an asymmetric structure which reverses over time; we have shown this is not due to time-shifting of the two proliferative phases across anatomical axes, nor can it be explained by different initial cell cycle times and decay rates. The proportional layer composition of CMZ-driven contributions is stable over this time, and does not display variability over the dorso-ventral axis. However, by estimating the evidence for time-stable and time-varying models of lineage contribution, we identified a decline in particular retinal lineage subtypes, particularly those of the INL, but including the GCL. This a potential explanation for the teleost postembryonic change in retinal mosaic pattern. Finally, we have shown that microglia-mediated turnover of retinal neurons is occurring at a rate too low to have a measurable effect on cohort sizes during this time.

Thus, the niche history of the CMZ is not adequately explained by a “frozen” population of RPC progenitors, homeostatically recapitulating a program of development found in embryonic progenitors, as has been suggested previously [HP98, WAR<sup>+</sup>16], for the proliferative potential implied by these models is too low. Instead, a model that does not limit the size of lineages, but has a slowly decaying cell cycle rate, is sufficient to produce the observed niche population history. This suggests that the postembryonic CMZ is under a separate regulatory regime from the embryonic RPCs, which contribute the central, larval remnant of the neural retina. This period of RPC activity benefits from separate modelling and investigation. Moreover, the decaying cycle rate model suggests an explanation for the difference between animals that have sustained CMZ RPC activity in adulthood, like teleosts, versus those that do not, such as mammals: the rate of decay in mammals may simply be much higher.

In addition to discovering unique features of postembryonic CMZ activity, we have proved out a general framework for Bayesian evaluation and selection of arbitrary biological models, which can inform both future experiments as well as modelling and theoretical choices. For instance, slice models more successfully combine population and morphological information, to answer questions about CMZ population structure, than do abstract whole-population models of the CMZ. From the foregoing analyses, we conclude that treating the CMZ population as a combined unit with shared proliferative parameter evolution is likely well justified, suggesting that much of the apparent complexity of the niche’s population history can be usefully abstracted away.

## 4.10 Future directions

These studies show that the simple datasets we have collected are sufficient to suggest a general model of the postembryonic CMZ, which does not explicitly limit the proliferative potential of RPCs by a measure of time in lineage, like the SMME models. However, they also reveal that these datasets provide little constraint on many important parameters of interest. This is an important insight, as future experiments can be designed with the nested sampling model comparison in mind. For instance,

our models consistently exhibit poorly constrained niche exit rates. The kinetics of niche exit could be examined by thymidine pulse-chase assays, timed to catch cohorts of RPC progeny as they cross the boundary from the CMZ into the neural retina, and this would likely significantly improve our estimates. Cell-based models of cycle parameters can glean some information from cumulative thymidine analogue labelling alone, but these would be greatly improved with costaining for markers of other cycle phases, for instance using phosphorylated histone H3 as a marker of M phase [RTL<sup>+</sup>18]. Experiments using multiple different thymidine analogue labels [HZP18] are another avenue for refining these estimates.

In general, the model comparison approach worked out here seems promising for testing competing explanations for observed phenomena. It would be worthwhile to improve the accuracy of `GMC_NS.jl`, and to continue developing models that make reference to the widest available array of observations and interventions. Given some adjustments to the experimental approach, and a refined sampler, this approach could replace obsolete models of cell cycle like the one described by Nowakowski et al. [NLM89], while giving a more realistic picture of the information actually provided by particular datasets.

As a final remark, it should be emphasized that the conclusions of this chapter always refer to the quality distribution jointly implied by the model and the data. Despite our failure to distinguish between RPC behaviours in different morphological regions of the CMZ, these rejected hypotheses should be returned to and re-tested with refined models and datasets. The same is true of phased and single-regime models of RPC cell cycle regulation. By proceeding in this manner, as Feyerabend suggested, we can “maximise the empirical content of [our] views” by “try[ing] to improve rather than discard the views that have failed in the competition.” [Fey93, p.20]

## Chapter 5

# Mutant npat results in nucleosome positioning defects in *D. rerio* CMZ progenitors, blocking fate specification but not proliferation

Data and images presented in Figure 5.1, Figure 5.8, Figure 5.10, Figure 5.12, and Supplementary Figure 13.6 were generated by Monica Dixon, who also collected npat ISH and pax6 IHC observations in Figure 5.9 and prepared samples for Figure 5.7. Maria Augusta Sartori prepared Figure 5.11, collected the data presented in Figure 5.13, Figure 5.14, Supplementary Figure 13.8, and performed morpholino injections underlying datasets in Figure 5.15 and Supplementary Figure 13.7.

## Synopsis

(1) The *Danio* mutant rys has a small eye with an apparently enlarged CMZ. (2) Mutant rys RPCs remain mitotic, but cycle slowly and fail to contribute to the neural retina. (3) The rys phenotype is primarily characterised by expanded nuclei with disorganised chromatin, expanded progenitor identity, and marginal cell death. (4) The causative mutation of rys lies in the npat gene, which produces elevated levels of H2A and H2B histone transcripts. (5) rys nucleosomes are displaced relative to siblings, and there is a subpopulation of position present in mutants but not siblings, and another in siblings but not mutants (the “differential sets”). (6) Estimation of Bayesian evidence for ICA models of nucleosome sequence preference suggest that the differential sets are produced by separate causal processes. (7) Maximum a posteriori estimates of these models suggest that the rys differential set positions are much more strongly influenced by nucleosome-DNA interaction than the sibling differential set. (8) This suggests a mechanism for npat’s effects on specification through the disruption of normal chromatin organisation, possibly by altering the available pool of histones.

## 5.1 Introduction

The zebrafish (*D. rerio*) circumferential marginal zone (CMZ), located in the retinal periphery, contains the retinal stem cells and progenitors responsible for the lifelong retinal neurogenesis observed in this cyprinid. Analogous to CMZs in other model organisms, such as *X. laevis* [PKVH98], it has been of particular interest to us since the discovery of quiescent stem cells at the mammalian retinal periphery [Tro00], as an understanding of the molecular regulation of this proliferative zone may shed light on whether these mammalian cells might be harnessed for the purpose of regenerative retinal medicine. While significant progress has been made in this direction [RBBP06], molecular lesions in a plethora of zebrafish mutants displaying defects in CMZ development and activity remain largely uncharacterised. We examine here one such microphthalmic line identified in an ENU screen, *rys* [WSM<sup>+</sup>05], characterised by Wehman et al. as a Class IIA CMZ mutant, with a small eyes (see Figure 5.1) and apparently paradoxically enlarged CMZ.

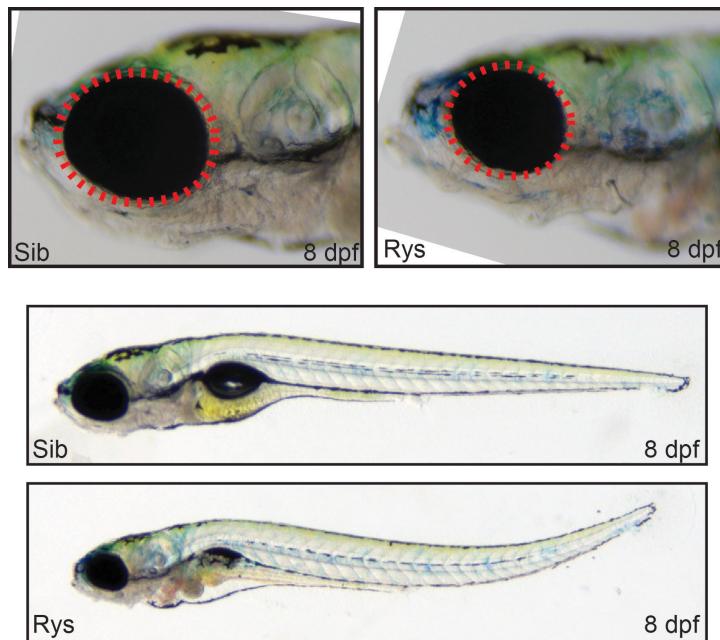


Figure 5.1: *rys* mutants exhibit a small-eye phenotype

8 dpf *rds* sibling (Sib) and mutant (Rys), head area (top panels), whole body (bottom panels). Red dashes highlight the overall reduced eye size in homozygous *rds* mutant animals.

Mapping revealed the causative *rds* mutation in the zebrafish npat gene, the nuclear protein associated with the ataxia-telangiectasia locus in mammals [IYS<sup>+</sup>96]. Although npat is heretofore uncharacterised in zebrafish, its mammalian homologues, human NPAT and mouse Npat, have been extensively examined. These studies demonstrated that NPAT plays a critical role in coordinating events associated with the G1/S phase transition in proliferating cells [YWNH03]. S-phase entry requires tight co-ordination between the onset of genomic DNA synthesis and histone production, in order to achieve normal chromatin packaging and assembly. NPAT, found in the nucleus [STN<sup>+</sup>02] and localised, in a cell-cycle dependent manner, to histone locus bodies [GDL<sup>+</sup>09], induces S-phase entry [ZDI<sup>+</sup>98] and activates replication-dependent histone gene transcription by direct interaction with histone gene clus-

ters [ZKL<sup>+00</sup>] in association with histone nuclear factor P (HiNF-P) [MXM<sup>+03</sup>]. The protein's effects on S-phase entry and histone transcription are associated with distinct domains at the C-terminus and N-terminus, respectively [WJH03]. NPAT is also known to associate with the histone acetyltransferase CBP/p300 [WIEO04] and directs histone acetylation by this enzyme [HYSL11].

NPAT is involved in E2F transcriptional regulation [GBB<sup>+03</sup>] and is substrate of cyclin E/CDK2 [ZDI<sup>+98</sup>], although E2F-independent activation by cyclin D2/CDK4 in human ES cells has also been described [BGL<sup>+10</sup>]. The expression of NPAT protein peaks at the G1/S boundary [ZDI<sup>+98</sup>], as does its phosphorylation, which promotes its transcriptional activation of replication-dependent H2B [Ma00] and H4 [MGv<sup>+09</sup>] genes, while its effect on low, basal levels of H4 transcription is phosphorylation-independent [YWNH03]. Of particular interest, NPAT has recently been found to be required for CDK9 recruitment to replication-dependent histone genes [PJ10]; CDK9 and monoubiquitinated H2B are essential for proper 3' end processing of stem-loop histone transcripts [PSS<sup>+09</sup>]. NPAT and HiNF-P have also been found associated with the U7 snRNP complexes that perform this function [GDL<sup>+09</sup>]. The replication-dependent activities of NPAT are thought to be terminated by WEE1 phosphorylation of H2B, which excludes NPAT from histone clusters [MFKM12].

All of these studies were performed in tissue culture, perhaps due to the challenges associated with studying this critical protein in vivo; mouse embryos with provirally inactivated Npat arrest at the 8-cell stage, for instance [DFWV<sup>+97</sup>]. The availability of *rds*, a zebrafish npat mutant which develops well into the larval stage, is therefore of considerable interest, as it allows for the study of npat's function within complete tissues. We demonstrate here that npat is critical for fate specification of CMZ neural progenitors. We find substantial evidence for the role of *D. rerio* npat involvement with histone transcription, nucleosome positioning, and scheduling of mitotic activity; we suggest that the *rds* phenotype is brought about by a failure to coordinate the genomic states required to specify, independently of proliferative capacity, in postembryonic zebrafish RPCs of the CMZ.

## 5.2 Results

### 5.2.1 The *rds* CMZ phenotype is characterised failure of RPCs to specify, altered nuclear morphology, aberrant proliferation and expanded early progenitor identity

*rds* reportedly has an enlarged CMZ [WSM<sup>+05</sup>], but whether this is a consequence of more RPCs, larger RPCs, or other causes, is unknown. The overall appearance of *rds* retinal sections is displayed in Figure 5.2. Two *rds* mutant eyes are displayed below a typical sibling eye. The bottom-most *rds* retina displays the extensive disorganisation which is the typical retinal endpoint of the *rds* pathology.

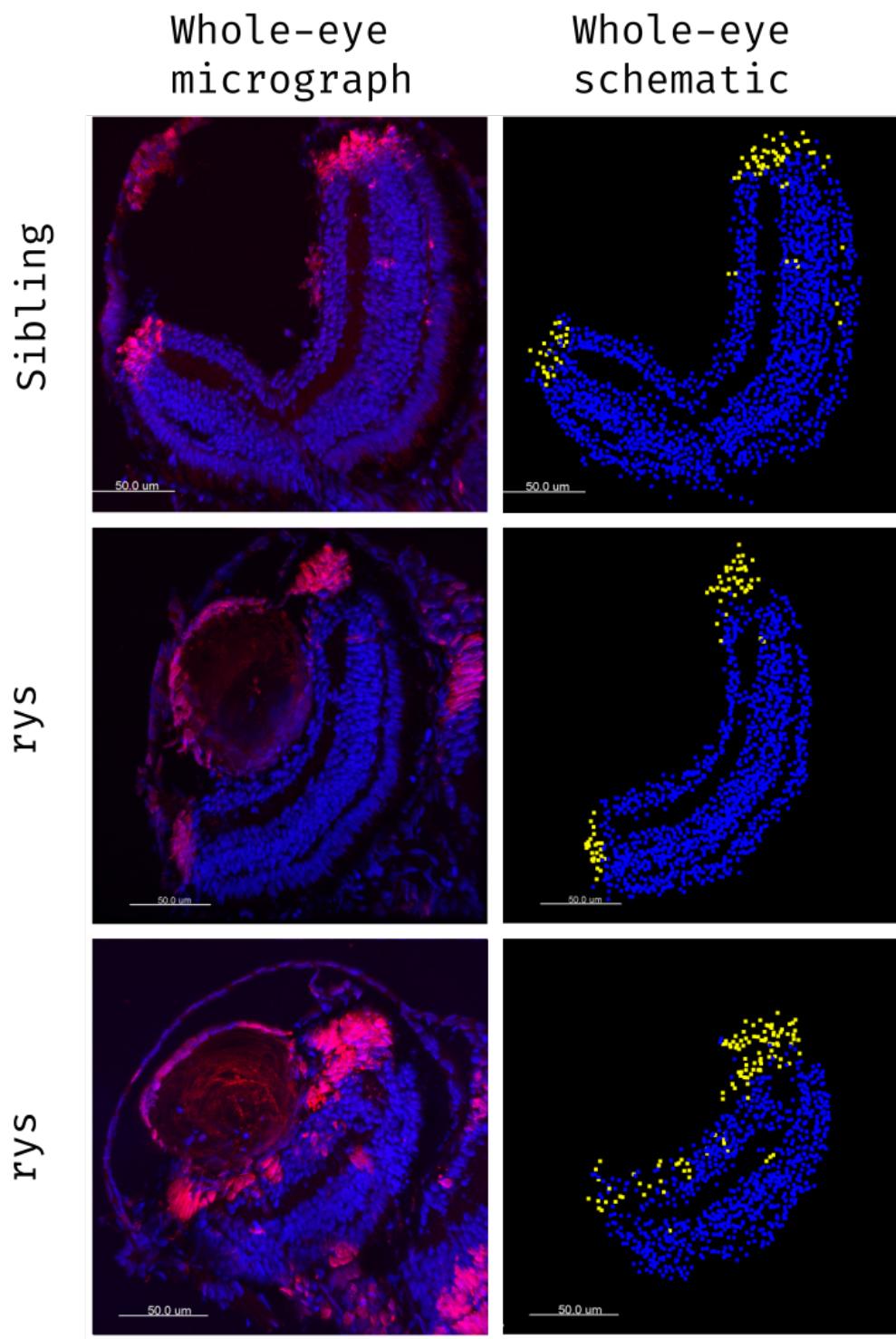


Figure 5.2: Typical *r*ys sibling and mutant retinal organisation and appearance

Left panels display maximum intensity projections of confocal micrographs taken of 14  $\mu\text{m}$  central coronal cryosections through a representative 5dpf sibling eye (top panels) and two *r*ys eyes, displaying less (middle panels) and greater (bottom panels) retinal disorganisation. Red staining of PCNA+ve nuclei against a blue Hoechst 33342 counterstain. Right panels display schematics of sectional nuclear organisation, with nuclear positions given by blue dots and PCNA+ve RPC nuclei highlighted with yellow. Methods in Section 13.1.5.

We examined niche ontogeny during the early life of *ryst* animals using two histochemical markers of cycle activity: Proliferating Cell Nuclear Antigen (PCNA), which in *D. rerio* is expressed throughout the cell cycle; and the genomic incorporation of the thymidine analogue EdU over a 12 hour pulse, marking passage through S-phase during this time. These data are displayed in Figure 5.3. During this period, the PCNA positive population of the sibling CMZ declines (Panel A), with a corresponding decrease in proliferative activity as measured by the incorporation of EdU (Panel B). Whether or not the mutant *ryst* CMZ population is enlarged by comparison depends on the age at which it is sampled. There is a 99.6% probability that the posterior mean RPC population in mutants is less than the sib mean at 4dpf, while there is a 99.8% probability of the 10dpf *ryst* mean being greater than that of sibs. Therefore, the *ryst* CMZ population is better described as achieving its peak periembryonic size later than siblings; its population is only numerically larger than sibs at later ages<sup>1</sup>.

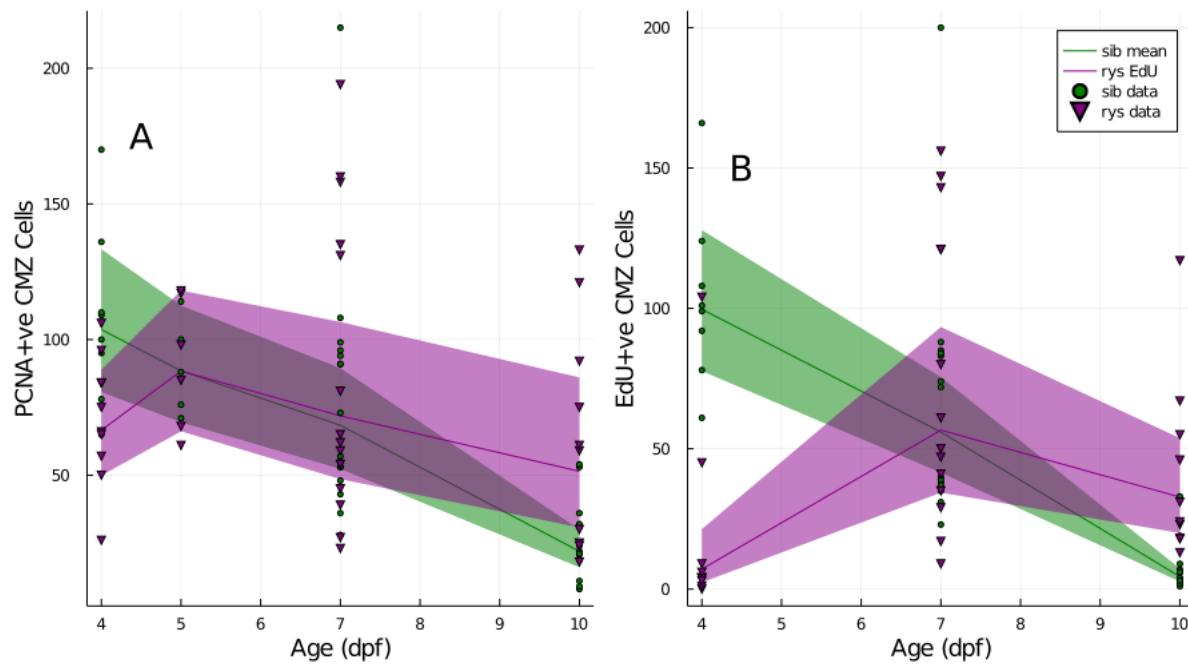


Figure 5.3: *ryst* CMZ populations start relatively small and quiescent, end abberantly large and proliferative

Panel A: Counts and estimated mean  $\pm 95\%$  credible intervals of PCNA-positive cells in the peripheral CMZ of *ryst* (magenta) and their siblings (green). Counts obtained from one 14  $\mu\text{m}$  central coronal cryosection per individual. Panel B: As above, but for counts of double PCNA-,EdU-positive cells in the CMZ after a 12 hour pulse of EdU. Methods in Section 13.1.5. Code in Section 17.8.25.

Few *ryst* RPCs are actively cycling at 4dpf, by comparison to the rapidly proliferating sibling cells; a mean estimate of  $96.2 \pm 3.4\%$  of sib RPCs are labelled during the 12 hr pulse at 4dpf, while only  $24.1 \pm 37.2\%$  of *ryst* RPCs are. The situation is broadly reversed at 10dpf, with only  $24.4 \pm 14.9\%$  of sib RPCs labelled, compared to  $65.8 \pm 16.2\%$  of *ryst* RPCs. Confirming this late-stage increase in *ryst* thymidine analogue labelling represents bona fide mitotic activity, we found mitotic figures in labelled mutant RPCs at 10dpf, shown in Supplementary Figure 13.1. In *ryst* animals which have actively proliferating RPCs at early ages, the CMZ fails to contribute to the central neural retina. Confocal

<sup>1</sup>*ryst* animals universally die by approximately 3 weeks of age.

micrographs displaying *rys* CMZ cohorts labelled with BrdU at 3dpf that have failed to enter the neural retina after 7 days of chase time are displayed in Figure 5.4.

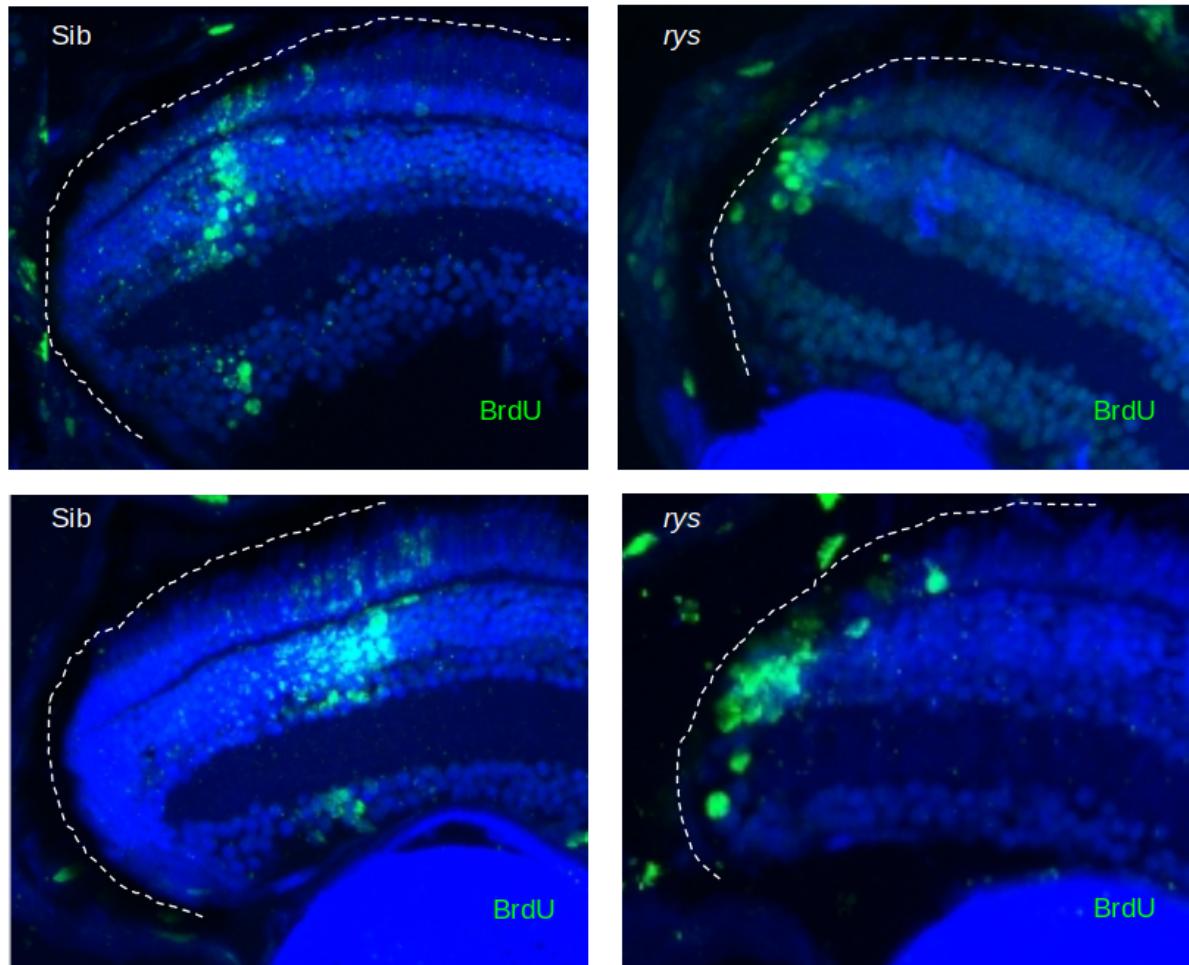
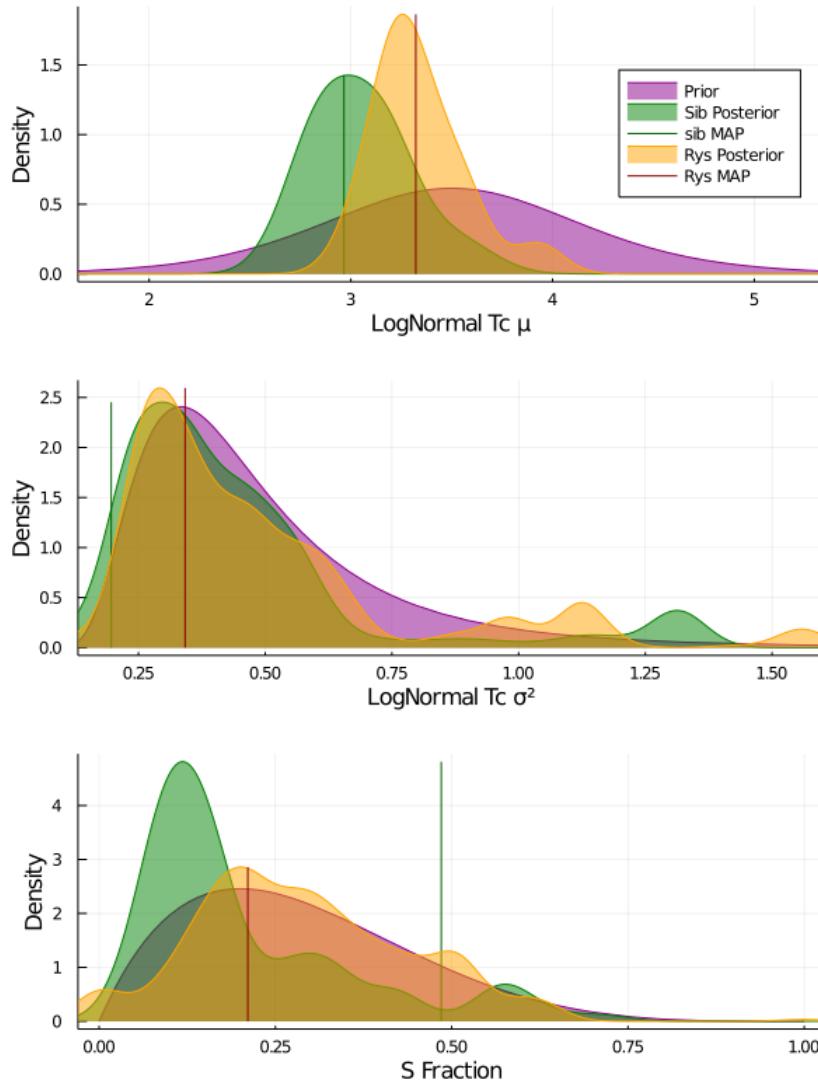


Figure 5.4: *rys* CMZ RPCs fail to contribute to the neural retina

14 $\mu$ m coronal cryosections through representative sib (left panels) and *r ys* (right panels) eyes at 10dpf, 7 days after an 8hr BrdU pulse at 3dpf. Green: anti-BrdU labelling. Blue: Hoechst 33342 counterstain. Methods in Section 13.1.6.

The thymidine analogue pulse-chase studies described above give little direct information about cell cycle characteristics, so we performed a cumulative thymidine labelling experiment in *r ys* mutants and siblings at 5dpf, when similar proportions of the PCNA+ve CMZ population retain thymidine label. We inferred cycle parameters by nested sampling, using the thymidine slice model simulator presented in Chapter 4. Kernel density estimates of relevant cycle posterior distributions are presented in Figure 5.5, while maximum a posteriori model output for sib and mutant data are presented in Supplementary Figure 13.2 and Figure 13.3, respectively. These analyses indicate that the mean cycle time in *r ys* mutant cells is likely to be longer than that of siblings, and a larger fraction of this time may be spent in S phase.

Examining 5dpf CMZ RPC nuclear number and morphology (Figure 5.6) suggests the enlarged appearance of *r ys* CMZs is the failure to contribute to the neural retina, leading to *r ys* RPCs contributing



**Figure 5.5: Cycling RPCs in 5dpf *rys* mutant retinas have longer cell cycle times**  
 Shared prior and estimated posterior marginal distributions for parameters of sib and rys cell cycle models, given cumulative EdU labelling data. The position of the maximum a posteriori model in the ensemble is marked with a vertical line. Top panel: Marginal posterior on the mean of the LogNormal model of cycle time. Middle panel: Marginal posterior on the variance of the LogNormal model of cycle time. Bottom panel: Marginal posterior on the fraction of the cell cycle spent in S phase. Methods in Section 11.1.2.2 and Section 12.1.12. Code in Section 17.8.16.

half as many central neurons as their siblings (Panel B), despite having similar overall populations at his age (Panel A). This appearance may be enhanced in the dorsal CMZ population, which tends to be more numerous in *rys* than sibs, although this tendency is reversed in the ventral population (Panels C and D). More significantly, *rys* nuclei are usually larger than their siblings (Panel E), as well as consistently less spherical (Panel F).

To rank these contributions to the *rys* nuclear phenotype, we calculated the Bayesian evidence (marginal probability) for combined Log-Normal (population measurements) and Normal (nuclear mea-

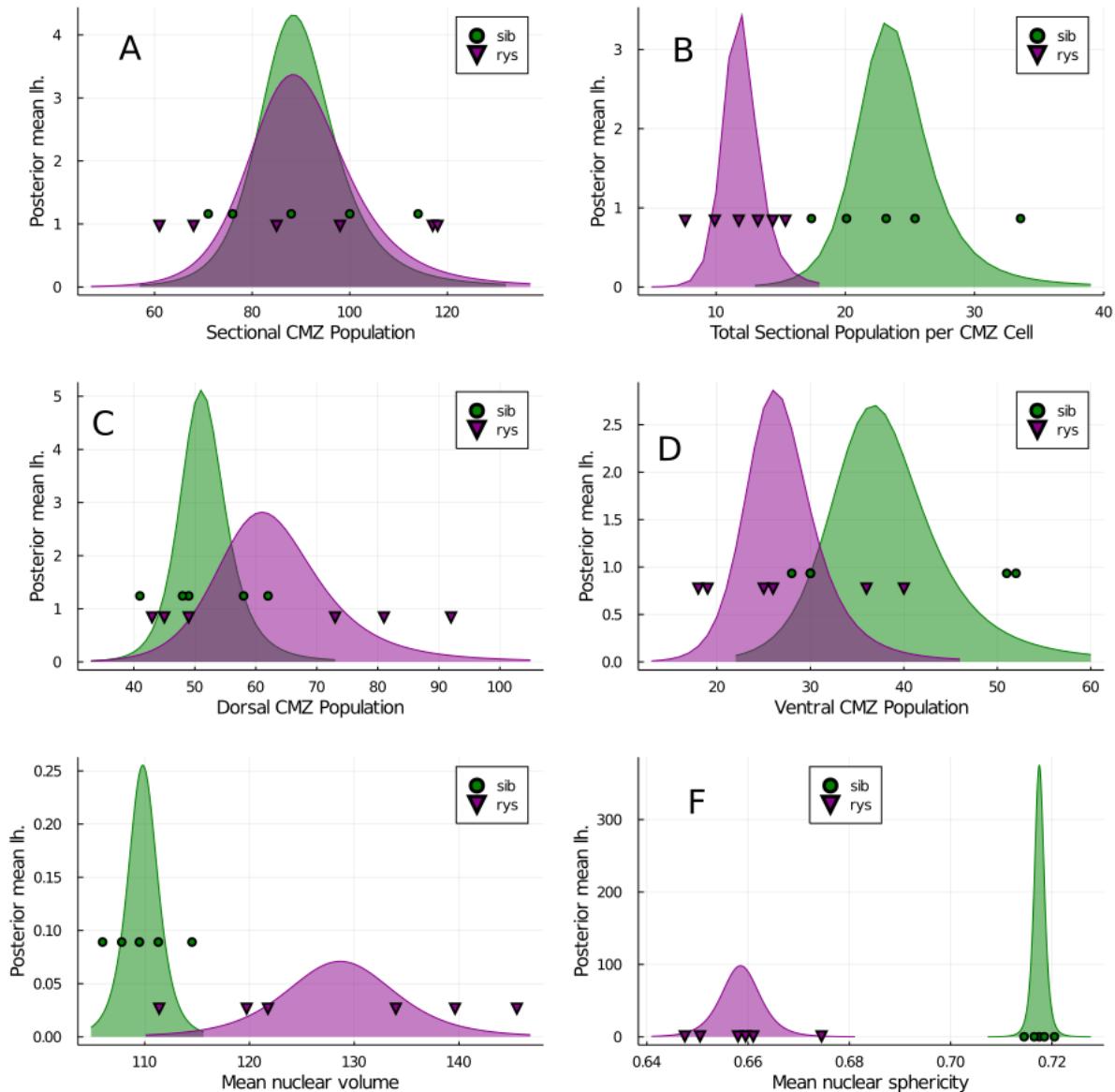


Figure 5.6: 7dpf *rys* CMZ RPCs display enhanced asymmetry, increased volume, decreased sphericity, and relative but not absolute enlargement

All panels display sib observations as green circles and *rys* as magenta triangles. Plotted behind the underlying observations are the calculated marginal posterior distributions of the mean, given log-Normal models of the population data and Normal models of the volume and sphericity data. The y-axis shows the relative likelihood of underlying mean values on the x-axis, given the data. Panel A: Total PCNA-positive RPC population per central coronal section. Panel B: Number of PCNA negative, specified central retinal neurons per PCNA positive RPC. Panels C and D: Dorsal and Ventral RPC populations per central coronal cryosection. Panel E: Mean volume of nuclei in a given individual's central coronal cryosection. Panel F: Mean sphericity of nuclei in a given individual's central coronal cryosection. Methods in Section 13.1.5. Code in Section 17.8.18.

surements) models of sib and *rys* data, against the joint evidence for separate models. These estimates

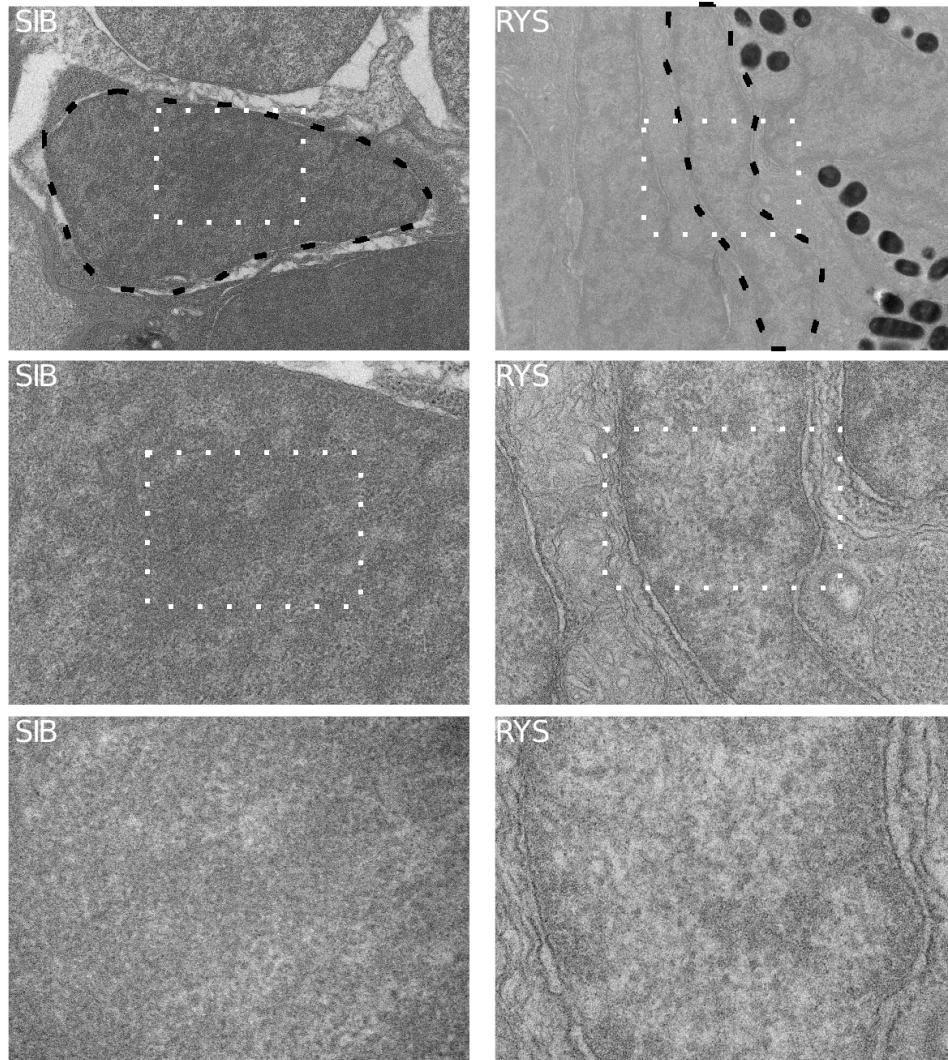
are presented in Table 5.1. Of the effects that differentiate *ryst*s from sibs (positive evidence ratio, logZR in Table 5.1), the largest is decreased nuclear sphericity, followed by decreased nuclear volume. These are followed by differences in dorsal and ventral CMZ populations, then by the mutant's decreased number of central retinal neurons relative to the RPC population. These models suggest the nuclear morphological changes are the most important contributors to the *ryst* enlarged-CMZ phenotype.

Table 5.1: Evidence-based ranking of phenomenal contributors to *ryst* phenotype

Parameter	Separate rys-sib logZ	Combined rys-sib logZ	logZR	$\sigma$ sign.
Sectional CMZ pop.	-299.9 $\pm$ 1.1	<b>-203.4 <math>\pm</math> 1.0</b>	-96.6 $\pm$ 1.5	-65.4
Central pop./RPC	<b>-56.11 <math>\pm</math> 0.27</b>	-79.52 $\pm$ 0.17	23.41 $\pm$ 0.32	73.7
Dorsal CMZ pop.	<b>-175.7 <math>\pm</math> 0.92</b>	-228.4 $\pm$ 1.4	52.7 $\pm$ 1.6	32.2
Ventral CMZ pop.	<b>-136.6 <math>\pm</math> 0.24</b>	-167.15 $\pm$ 0.13	30.55 $\pm$ 0.27	113.8
Nuclear Volume	<b>-164.57 <math>\pm</math> 0.22</b>	-224.3 $\pm$ 1.7	59.7 $\pm$ 1.7	34.4
Nuclear Sphericity	<b>-87.19 <math>\pm</math> 0.27</b>	-265.5 $\pm$ 2.3	178.3 $\pm$ 2.3	78.4

logZ: logarithm of p(D), the marginal likelihood of the data, or model evidence. logZR: evidence ratio; positive ratios in favour of separate modelling of the sib and *ryst* data. Largest evidence values bolded. Methods in Section 13.1.8. Code in Section 17.8.24.

Enlarged *ryst* RPC nuclei have a billowy appearance suggestive of chromosomal disorganization. We used electron microscopy to examine the nuclear ultrastructure of RPC nuclei in *ryst* and sibling CMZs. Representative electron micrographs are presented in Figure 5.7. At 36000x magnification, the stretched and pancaked structure of *ryst* nuclei is apparent, in contrast with the consistently teardrop-shaped nuclei of sibling RPCs. When the chromatin is imaged at 210000x magnification, it appears less electron-dense in *ryst* mutants, with larger tracts of presumptive euchromatin, and less regular spacing of chromosomal material.



**Figure 5.7: RPC nuclei of the *rys* CMZ display disorganized, loosely packed chromatin**  
 Representative electron micrographs of *rys* sibling and mutant CMZ RPC nuclei. Left panels: siblings. Right panels: *rys*. Top panels: 36000x magnification, general overview of the area around the nucleus, dashed black. Area displayed in middle panels dashed white. Middle panels: 110000x magnification nuclear detail. Area displayed in bottom panels dashed white. Bottom panels: 210000x magnification, chromosomal ultrastructure. Methods in [Section 13.1.11](#).

If RPCs in *rys* CMZs fail to enter the specified neural retina, but by 7dpf are becoming mitotically active, an explanation for the proliferative niche's population declining by 10dpf is required. We suspected *rys* RPCs may be apoptosing *in situ*. Although we did not detect pyknotic nuclear fragments in our EM investigations, it may be that the apoptotic events are too rare in *rys* to detect in this manner. We therefore assayed the presence of the apoptotic marker caspase-3 in *rys* and sib CMZs. As displayed in [Figure 5.8](#), caspase-3-positive nuclei can be detected in the *rys* CMZ at both 4 and 6 dpf (mean  $6.5 \pm 7.5$  and  $2.6 \pm 1.1$  cells, respectively) but are not found in sib CMZs, though both display a similar level of central apoptotic activity (4dpf *rys*:  $1.25 \pm 1.0$ ; 4dpf sib:  $1.4 \pm 1.1$ ; 6dpf *rys*  $0.6 \pm 0.9$ ; 6dpf sib:  $1.0 \pm 1.4$ ). The lack of debris observable in *rys* CMZs is likely attributable to the activity of 4C4-positive

microglia active in the area; we observed one such cell actively phagocytosing TUNEL-labelled *rzs* RPC nuclear fragments in the CMZ, displayed in Supplementary Figure 13.4.

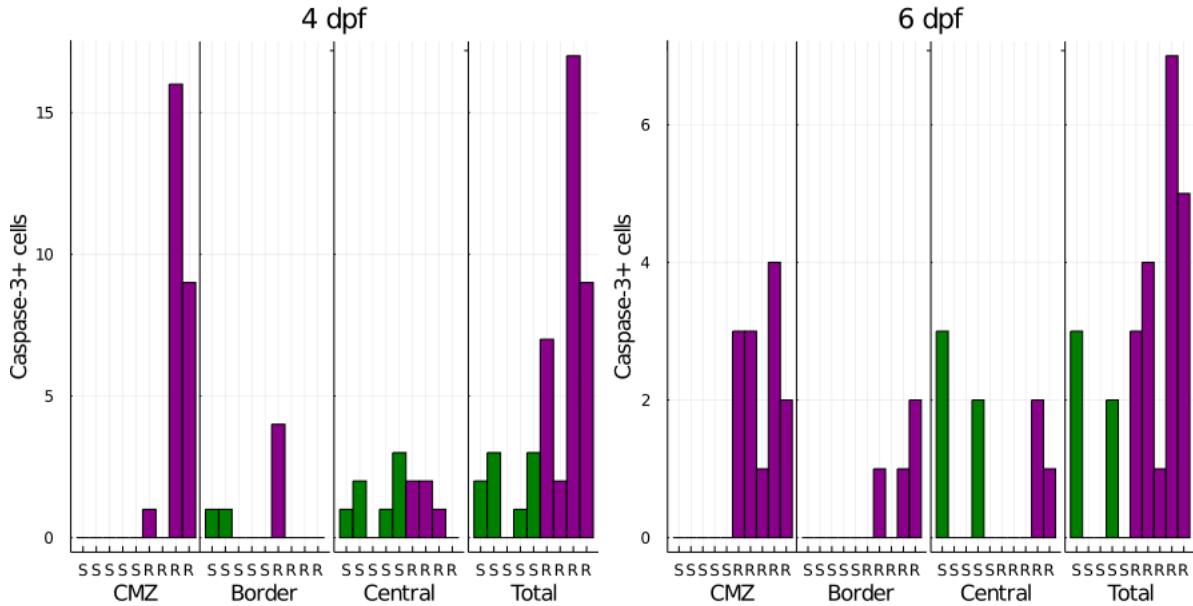
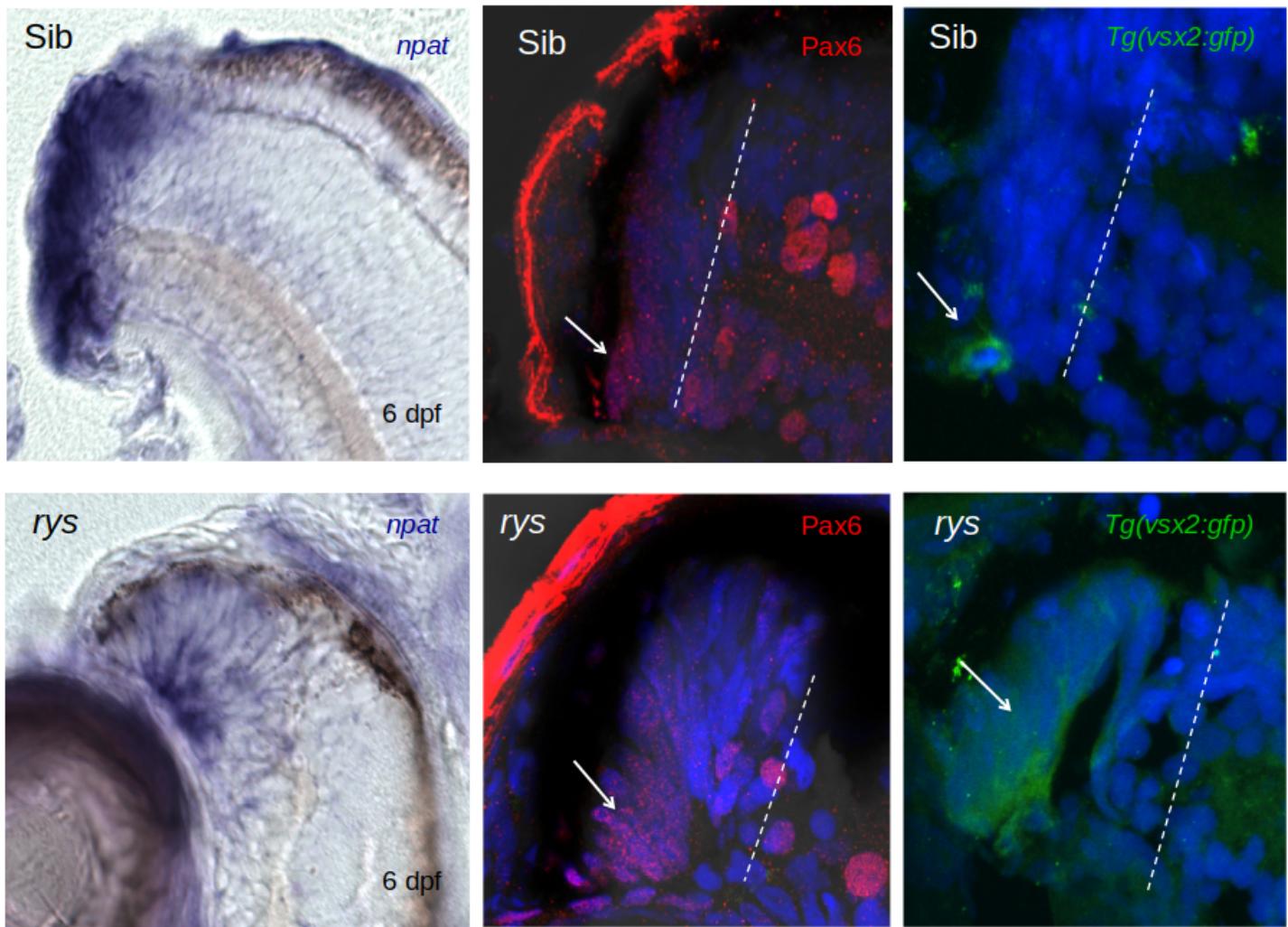


Figure 5.8: *rzs* mutant CMZs have increased caspase-3 positive nuclei

Number of caspase-3 positive nuclei counted in CMZ, the border between the CMZ and specified central retina, the specified central neural retina, and the overall total, at 4 dpf (left panel) and 6 dpf (right panel). One central 20  $\mu$ m section per sibling (S) and *rzs* (R) larva. Methods in Section 13.1.10. Code in Section 17.8.20.

Suspecting that *rzs* CMZ RPCs may maintain an early progenitor identity, we examined pax6a immunostaining of this population, which is normally restricted to putative progenitors in the peripheral and middle CMZ, as well as the retinal ganglion cell layer [RBBP06]. The Pax6-stained region was enlarged in *rzs* CMZs relative to their siblings (Figure 5.9, center panels). As vsx2 is also known to be a marker of retinal progenitor cells in the CMZ [RBBP06], we also generated a transgenic vsx2::eGFP *rzs* line, which produces eGFP expression in the utmost retinal periphery in siblings. Mutant fish from this line displayed substantially expanded eGFP expression in the CMZ (Figure 5.9, right panels).



**Figure 5.9: Mutant *rys* RPCs display expanded expression of early progenitor markers**  
 Representative transmitted light and confocal micrographs of *rys* sibling and mutant CMZ RPCs. Top panels: siblings. Bottom panels: *rys* mutants. Left panels: in-situ hybridization using npat probe (blue) on 20 $\mu$ m coronal cryosections. Middle panels: anti-Pax6 (red) on Hoechst 33342 nuclear counterstain. Right panels: anti-GFP (green) on Hoechst 33342 nuclear counterstain, in a Tg(vsx2:GFP) line introgressed into *rys*. Methods in Section 13.1.9.

### 5.2.2 The microphthalmic zebrafish line *rys* is an npat mutant

We performed linkage mapping to identify the causative mutation responsible for the *rys* phenotype, followed by PCR analysis of the transcript products of candidate genes. This study revealed a single G>A transition at position 24862961 on chromosome 15 (NC\_007126.5, Zv9), annotated as the first base of intron 9 in the zebrafish npat gene, in a canonical GU splice donor site.

We observed that this mutation reliably results in the retention of npat intron 8, and, less frequently, in the retention of both introns 8 and 9, in 6dpf *rys* mutants, shown in Figure 5.10. The predicted mutant protein is truncated by a stop codon at residue 283, which would preclude the translation of predicted phosphorylation sites and nuclear localisation signals cognate to those identified in human

NPAT [Ma00, STN<sup>+</sup>02], as displayed in Figure 5.11.

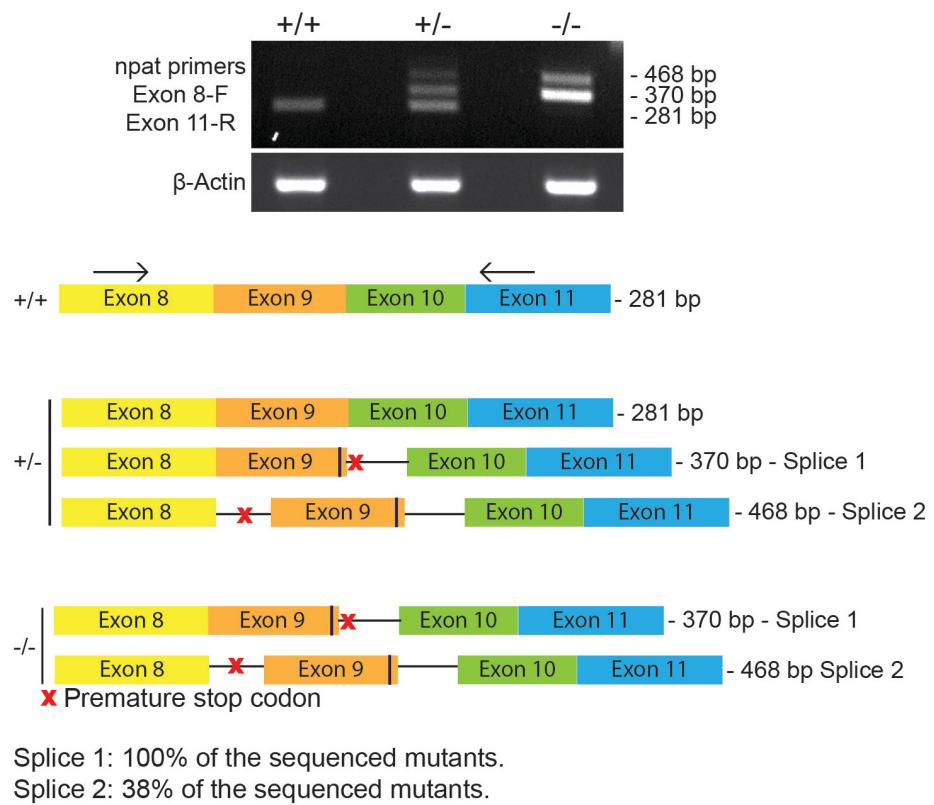


Figure 5.10: RT-PCR analysis reveals two aberrant intron retention variants in *ryst* mutant *npat* transcripts

Top panel: Agarose gel electrophoresis of mRNAs prepared from 3dpf homozygous wild type (+/+), heterozygote (+/-), and homozygous mutant (-/-) animals, as identified by genomic PCR. Fragments were amplified from a forward primer sited in exon 8 and a reverse primer sited in exon 11. Bottom panel: exon/intron layout schematics of the putative transcripts detected. Methods in Section 13.1.12.

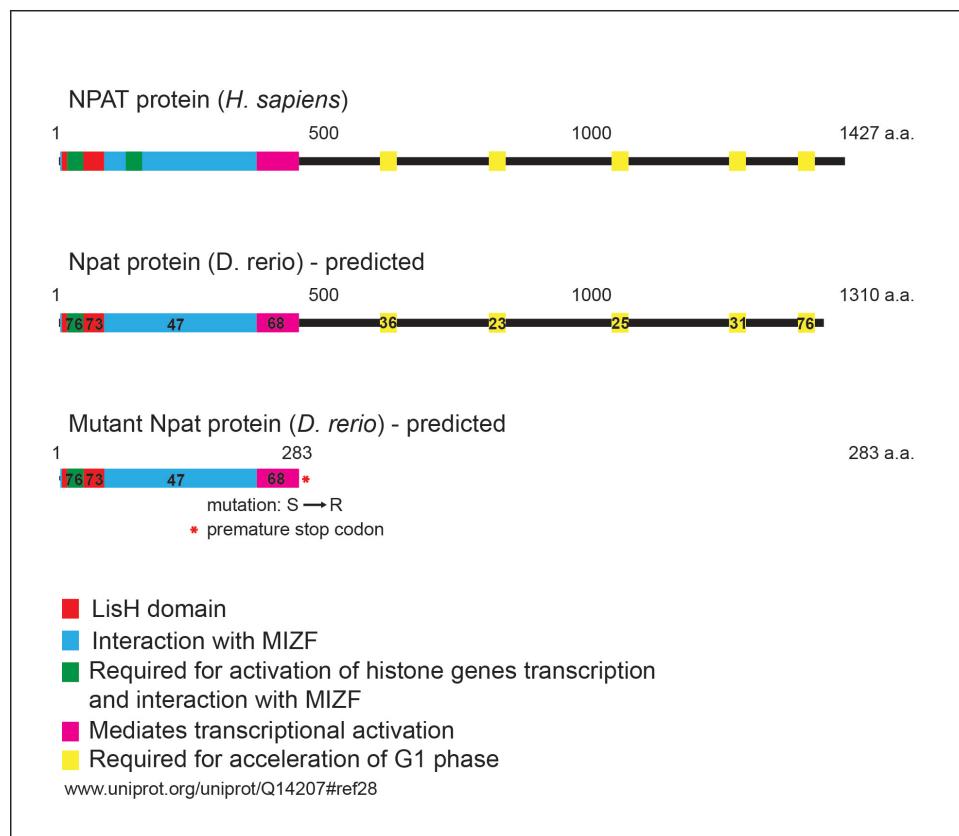


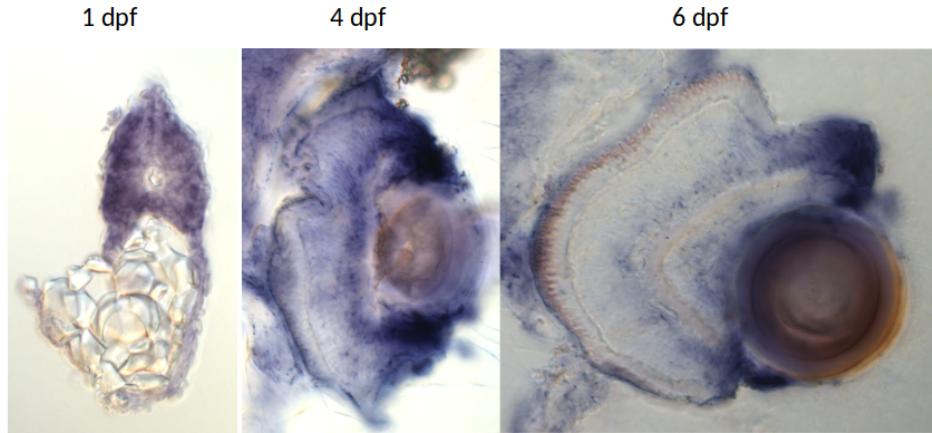
Figure 5.11: **Functional domains of Human NPAT compared to predicted wild-type and *rys* *Danio* npat**

Proteins presented as linear domain schematics with number of amino acids along the horizontal axis.

Because *D. rerio* is a teleost known to have undergone genome duplication in an ancestral clade, we used the Synteny Database tool [CCP09] identify any possible npat duplicates. A duplicated gene could explain the lessened severity of the mutant *Danio* phenotype when compared to mammalian proviral inactivants. The results of this analysis are presented in Supplementary Figure 13.5. While npat is in the midst of a region which appears to be duplicated on chromosomes 5 and 15, relative to the homologous synteny run on *H. sapiens* chromosome 11, it is not, itself, duplicated.

We performed in situ hybridisation against npat transcript to confirm that the gene is indeed expressed in wild type CMZs; we found that npat expression is progressively restricted to the CMZ from 4 to 6 dpf in wild-type fish. A representative time-course of 20 µm cryosections through ISH-treated embryos is depicted in Figure 5.12, focusing on the retina at the times when it has formed. Although npat remains transcribed in both the specified GCL and amacrine-rich inner INL to a degree, it is most intensely expressed in the proliferative CMZ, as suggested by its cell cycle functions.

Given unusual chromatin ultrastructure in *rys* CMZ RPCs, and npat's role in histone regulation, we investigated the transcriptional status of npat and its histone regulatory targets by RT-PCR. Homozygous *rys* mutants overtranscribe npat by about 3-fold compared to their wild-type counterparts at both 6dpf and 8dpf. Sibling overabundance compared to wild-type declines from about 2.5-fold to 1.5-fold over this time period, as shown in Supplementary Figure 13.6.



**Figure 5.12: In situ hybridization reveals progressive restriction of npat expression to the CMZ**

20 $\mu$ m sections of wild-type embryo (1dpf) and retinae (4 and 6 dpf), displaying progressive restriction of npat expression (blue) as assayed by in-situ hybridisation. Methods in Section 13.1.9.

Since mammalian NPAT regulates histone transcription and is critical for coordinating expression of replication-dependent histones required to package genomic DNA during S-phase [ZKL<sup>+</sup>00], we hypothesized that the altered nuclear morphology we observed in *rys* CMZ RPCs may be a consequence of perturbed histone transcription. To test this, we performed qPCR on random-hexamer-primed cDNAs produced from 6 and 8dpf wild type, sibling, and *rys* embryo mRNA extracts. These qPCR assays were performed using degenerate primers<sup>2</sup> directed toward zebrafish core histone gene families H2A, H2B, H3 and H4. As a role for NPAT in 3' end processing of histone transcripts has been identified [PJ10], we also set out to determine whether 3' end processing of histone transcripts is altered in *rys*. By repeating these qPCR assays on oligo-dT-primed cDNAs, we examined the population of polyadenylated histone transcripts in isolation.

These results demonstrated that both mutant *rys* and siblings overexpress histone transcripts to various degrees. In order to determine which histone families are most plausibly involved in the *rys* mutant phenotype, we calculated the statistical significance for mean *rys* transcript levels exceeding mean sibling transcript level or the WT standard (set to 1.0). To be implicated in the phenotype, mutant transcript levels should be higher than both siblings and wild-type animals. We used a cutoff of 2 SDs of significance for both the sibling and WT tests for a given transcript family. These values are presented in Table 5.2.

Both total and polyadenylated *rys* H2A are overexpressed in mutants relative to both siblings and WT controls at 8 dpf, while total H2B is overexpressed at 6dpf and polyadenylated H2B at 8dpf, using the criterion of 2 SDs of significance on both tests. We note that the overabundances of polyadenylated H2A and H2B are approximately 2-20 fold greater than those observed for any other transcript pool or histone family.

The joint probability that mean *rys* mutant total H2A is greater than the sib mean at both 6dpf and

---

<sup>2</sup>Zebrafish have notably populous histone clusters with numerous variants and pseudogenes not present in non-genomically-duplicated vertebrates, so degenerate primers are an appropriate way to survey the population of transcripts. A full catalogue has not been undertaken, to my knowledge.

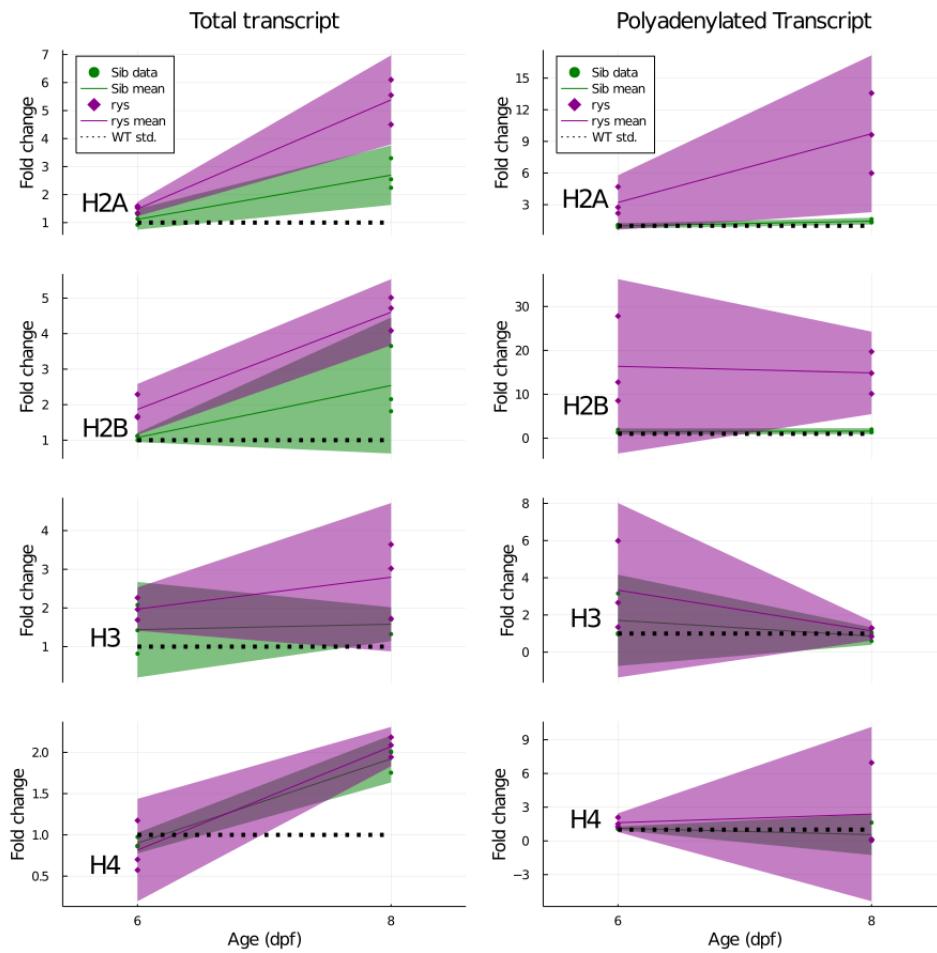


Figure 5.13: *rys* overexpress total and polyadenylated core histone transcripts  
qPCR results for degenerately-primed families of core histone total transcripts (left panels), and polyadenylated transcripts (right panels). From top to bottom, H2A, H2B, H3, and H4 family results are displayed in separate panels. Animal age in dpf on the x axis, fold change vs. wild-type standard on the y axis. Methods in Section 13.1.13. Code in Section 17.8.23.

8dpf to be  $99.7\% \pm 5.0$ ; the same figure for H2B is  $98.5\% \pm 5.1$ . The joint probability that mean total H2A and H2B are elevated in *rys* mutants compared to sibs at both ages is therefore  $98.2\% \pm 7.1$ . For the polyadenylated transcripts, the joint probability that the *rys* mutant polyA H2A mean is greater than the sib mean at both ages is  $94.1\% \pm 9.9$ ; the same figure for H2B is  $92.6\% \pm 3.3$ . The joint probability that mean polyadenylated H2A and H2B are elevated in *rys* relative to sibs at both ages is then  $87.1\% \pm 9.7$ .

On the basis of these calculations, the most plausible causal contributors to the *rys* phenotype are H2A and H2B family transcripts. While the relative magnitude of total H2A and H2B transcript overexpression is less than that of the polyadenylated pool, we have less uncertainty about these figures. Still, the polyadenylated transcript results suggest mutant npat increases polyadenylation of these transcripts, possibly as a result of a loss of regulation of this system. Unlike the total transcript pool, siblings retain tight, WT-like control over polyadenylated transcripts. *rys* mutants display much more extensive

Table 5.2: Standard deviations of significance for *rys* mutant transcript > sib or WT

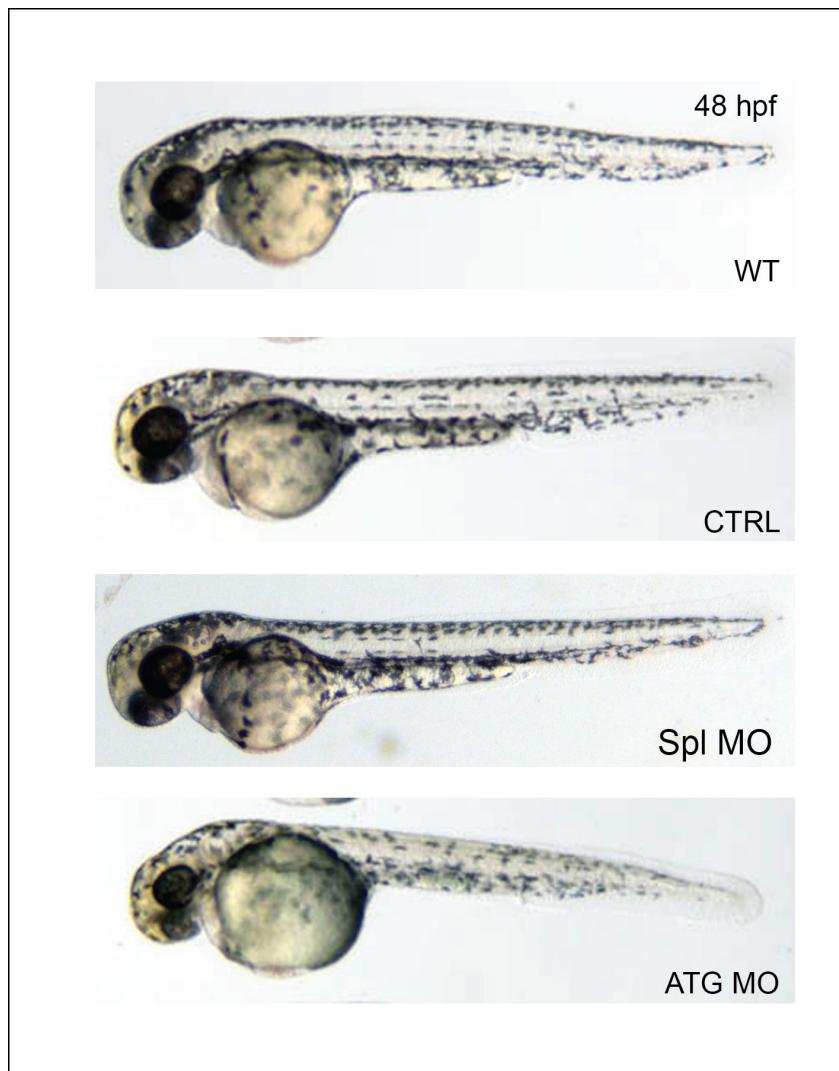
Transcript Family	Age (dpf)	Total		polyA	
		Sib mean $\sigma$	WT std $\sigma$	Sib mean $\sigma$	WT std $\sigma$
H2A	6	1.59	3.72	1.68	1.68
H2A	8	<b>2.75</b>	<b>5.4</b>	<b>2.18</b>	<b>2.3</b>
H2B	6	<b>2.12</b>	<b>2.37</b>	1.47	1.52
H2B	8	1.9	7.59	<b>2.76</b>	<b>2.89</b>
H3	6	0.77	3.39	0.6	0.97
H3	8	1.21	1.83	0.78	0.54
H4	6	-0.26	-0.58	1.15	1.46
H4	8	0.79	8.83	0.45	0.35

Standard deviations of significance for differences between mean *rys* mutant transcript expression, relative to siblings or WT standard (1.0), at 6 and 8 dpf. Values for histone transcript pools with  $>2$  SDs of significance for both sib and WT tests in bold. Methods in Section 13.1.13. Code in Section 17.8.23.

variability in polyA transcript expression, even where the mean is not elevated above siblings or WT controls.

We further investigated npat’s involvement by perturbing its transcription using morpholino injections of wild-type (AB strain) fish. As we do not suppose early morpholino transcriptional blockade will exactly replicate the cellular conditions of a genetic null some days after birth, particularly given the maternal contribution of npat transcript [HSK<sup>+</sup>13], we did not seek to recapitulate the *rys* phenotype. Rather, we sought to determine if any of *rys*’s constituent phenomena would appear after an early blockage of *rys* transcription, further substantiating the general involvement of npat in *rys*. We tested morpholinos directed both to the npat start codon and to the splice site affected in *rys*, alongside control morpholinos and uninjected animals. We used both a morpholino directed to the transcript ATG start site, as well as to the affected splice site.

The splice morpholino replicated the *rys* overexpression of both total and polyadenylated core histone transcript, while the ATG morpholino more narrowly replicated the effect on polyadenylated transcript (Supplementary Figure 13.8). The ATG morpholino also produced some animals with small eyes by 72dpf, as shown in Figure 5.14, although, in general, the disorganisation observed in *rys* retinae was not recapitulated, as displayed in Supplementary Figure 13.7. We also conducted an analysis of the nuclear morphological parameters measured in Figure 5.6 for *rys* mutants and siblings. For each of the ATG and splice morpholinos, we estimated the joint evidence for separate experimental morpholino and control morpholino models against a combined model. The evidence calculations are presented in Supplementary Table 13.1. We found substantial evidence in favour of separate models for the ATG and control morpholino effects on the number of central retinal neurons per proliferating RPC, suggesting that the ATG morpholino tends to decrease CMZ contribution to the specified retina. This result is displayed in Figure 5.15; the effect is more subtle than that displayed by mutants in Figure 5.6. These results establish that the identification of npat as the causative mutation in *rys* is plausible, as experimental perturbation to npat in WT fish can result in *rys* phenotypic phenomena, notably overexpression of core histone transcripts and small eyes arising from a decrease in the population of the central retina without a concomittant decrease in CMZ population.

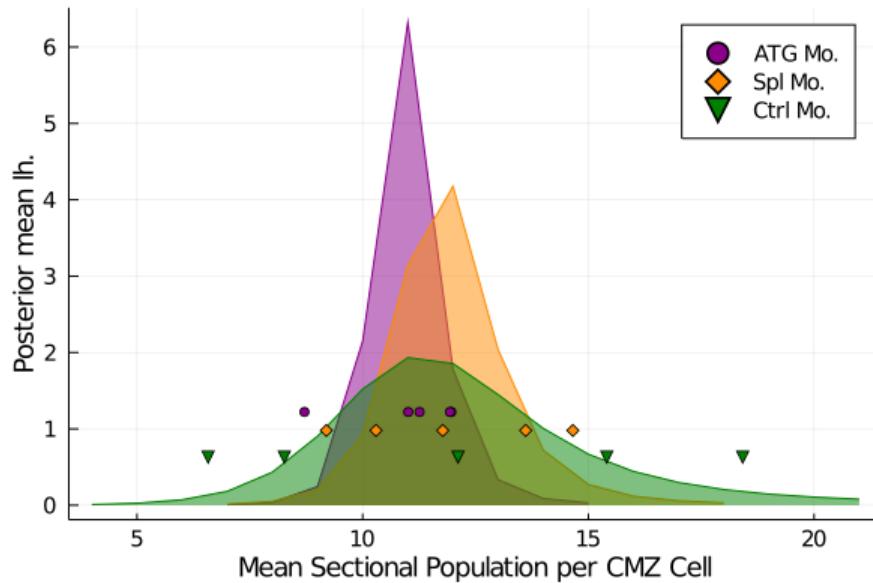


**Figure 5.14: 48hpf embryos injected with an npat ATG-targeted morpholino display a small-eye phenotype**

48 hpf wild-type embryos injected at the 1-2 cell stage with a control morpholino (CTRL), a morpholino targeting the npat splice site affected by the *rys* mutation (Spl MO), and a morpholino targeting the npat start codon (ATG Mo). Methods in [Section 13.1.3](#).

### 5.2.3 *rys* siblings and mutants have unique sets of nucleosome positions, best explained by different sequence preferences and increased sequence-dependent positioning in mutants

Perturbed histone expression in *rys* embryos suggests disrupted chromatin organisation as an explanation for the mutant nuclear phenotype, a possible consequence of altering the dynamic composition of the histone pool available for nucleosome formation during cell cycle. We therefore characterised nucleosome positioning in *rys* and wild-type siblings by micrococcal nuclease (MNase) digestion of pooled genomic DNA (gDNA) [CZ12]. Nucleosome-protected fragments from the MNase digests were sequenced and positions called as described previously [CXP<sup>+</sup>13].



**Figure 5.15: ATG morpholino perturbation of npat recapitulates decline in mean central retinal population relative to CMZ**

Populations of 14 $\mu$ m central coronal cryosections from retinas of animals injected with control (green triangles), npat splice-directed (yellow diamonds), and npat start codon-directed (magenta circles), standardized by number of CMZ RPCs. Plotted behind the underlying observations are the calculated marginal posterior distributions of the mean, given log-Normal models of the data. The y-axis displays the relative likelihood of underlying mean values on the x-axis, given the data. Methods in [Section 13.1.3](#), [Section 13.1.5](#). Code in [Section 17.8.19](#).

We first examined the genomic disposition of nucleosome positions within wild-type siblings and *rys*. Nucleosome positions are distributed relatively evenly over sib genomic material, with only tiny deviations from a neutral assumption of a uniform distribution, as displayed in Fig. 5.16, panel A. Bulk *rys* chromatin displays minor differences from the proportions of positions found in each sib scaffold (panel B), with a tendency toward more even distribution of positions. Sib nucleosome positions are differentially occupied across scaffolds, with Chr 10 and scaffolds not yet mapped to chromosomes (NC) being notably more occupied than expected from scaffold length alone (panel C). Similarly to the distribution of positions, *rys* chromatin is more evenly occupied than sibs; scaffolds with positions that are more heavily occupied in sibs tend to be depleted in *rys* and vice versa (panel D).

By mapping the called nucleosome positions in *rys* to those in sibs, and calculating the number of bases the *rys* position is displaced from its mapped position, we characterised the translational displacement from the arrangement found in siblings. The probability distribution of this displacement parameter for the population of *rys* positions is displayed in [Figure 5.17](#). The notable bimodality of this distribution arises as a result of the inclusion of unmapped positions, to show the relative size of this population: 23% of positions are not mapped to any sibling position at all; these are located at 141, greater than the full length of a called nucleosome position in our pipeline. The other three-quarters of the positions are mapped to sibling positions, with 11% of overall *rys* positions coinciding entirely with the sibling positions, 19% displaced 10 bases or less from a sib position, and a further 21% displaced between 10 and 30 bases. The last quarter of *rys* positions display more extreme displacements, up to

the entire length of the position. The periodic multimodality observed in the displacement distance of *rys* positions relative to sibs is commonly observed in nucleosome positioning and arises from the 10-base periodicity of nucleosome contacts with DNA [WC17]. This distribution of population displacement suggests that the npat mutation in *rys* results both in a loss of translational control of nucleosomes that are in approximately the expected positions, most commonly by a nucleosome “roll” of one contact, as well as dysregulation of the chromatin remodelling processes which get nucleosomes to the correct positions to begin with, represented by the novel *rys* positions, with no mapped sibling counterpart.

We sought to identify position subpopulations that might represent those involved in the nuclear phenotype of mutant *rys* RPCs. Interestingly, not only are there *rys* nucleosome positions which are not found in siblings, but there is likewise a subset of sib positions which are never found to be occupied in mutant *rys*. Intriguingly, sib positions not observed in *rys* are compensated for quite evenly across the genome by new positions gained in the mutants, with a small excess of new *rys* positions on most chromosomes, as shown in Supplementary Figure 13.9. Moreover, the novel positions in *rys* are much less typical of the genome, relative to lost sibling positions, than are *rys* positions which overlap ones found in sibs, summarised in Supplementary Table 13.2. This result suggested that a particular subpopulation of wild type nucleosomes are mislocalised in *rys*. This could occur for a variety of reasons, which may not be reflected in the position sequences themselves. For instance, mutant npat could affect chromatin remodelling enzymes without this having a systematic effect on the primary sequence of positions. Mutant npat could also alter the pool of available histones in proliferating progenitors, resulting in altered physico-chemical interactions with DNA, which could manifest as a preponderance of nucleosomes with unusual sequence preferences. If so, this would explain the disappearance of a subset of positions in *rys* and the appearance of a new, similarly sized subset of positions (the ‘differential set’).

To distinguish between these different possibilities, we calculated the Bayesian evidence ratio for models of separate emission processes for sib and *rys* position sequences, against a combined process for both sets of sequences. If the effect of npat is unrelated to direct interaction between nucleosomes and DNA, the combined model should receive the most support, since any signals appearing in the separate models would be unrelated to the status of npat. On the other hand, if separate models receive more support, DNA sequence preferences are implicated in the differential positioning of mutant and sib nucleosomes. If the latter proves to be the case, the maximum a posteriori parameters of these models can be compared to reveal the specific changes in the prevalence and identity of detected sequence signals.

The emission model used was the Independent Component Analysis form described by Down and Hubbard [DH05], with a fixed number of independent, variable length position weight matrices related to observations by a Boolean mixing matrix. An observation is scored by the background likelihood of its sequence, given a model of genomic noise, convolved with a sequence-length- and cardinality-penalized score<sup>3</sup> for each source which the mix matrix indicates is present in the observation. Therefore, we first needed to construct background models of *D. rerio*’s genomic “noise” from which repetitive sequence signals characteristic of nucleosome positions could be extracted. Following the suggestion of Down and Hubbard [DH05] that a principled approach to the selection of background models is to train and test a variety of them on relevant sequence, we used the Julia package BioBackgroundModels (presented in Chapter 9) to screen a panel of 1, 2, 4, and 6-state HMMs against 0th, 1st, and 2nd order encodings of samples from the zebrafish genome, partitioned grossly into exonic, periexonic, and intergenic sequences.

---

<sup>3</sup>That is, the additional score provided by a source to an observation is penalized both by the expected number of motif occurrences given an observation of that length, as well as by the number of sources which explain the observation in the model.

We found that each of these partitions is best represented by 6-state HMMs trained on a 0th order genome encoding (i.e. the HMMs emit the 4 mononucleotides), as determined by model likelihood given an independent test sample, displayed in Supplementary Figure 13.10.

We used the Julia library BioMotifInference (presented in Chapter 10) to sample from the posterior distributions for these models, given the differential set of *rys* nucleosome positions and the composite background model of genomic noise. We initialized separate ensembles from uninformative priors on the *rys* and *sib* data alone, as well as the combined *rys* and *sib* data, allowing for 8 independent PWM sources in all cases. We compressed three model ensembles to within 125 orders of magnitude between the maximum likelihood model sampled and the minimum ensemble likelihood<sup>4</sup>. This process produced the model evidence estimates summarized in Table 5.3, as well as maximum a posteriori samples for each of the ensembles. We estimate that there are greater than 360 orders of magnitude of evidence in favour of separate generative processes for the differential *sib* and *rys* than for a combined model, with an estimated 525 standard deviations of significance. This large evidentiary weight and significance give us statistical almost-certainty in selecting separate models for these sequences over a combined model. We present the MAP PWM source samples from the better-evidentiated separate *sib* and *rys* models in Figure 5.18 and Figure 5.19 respectively, while the inferred combined sources are available in Supplementary Figure 13.11.

Table 5.3: Evidence favours separate emission models for the *rys* mutant and sibling differential position sets

Sib logZ	<i>rys</i> logZ	Joint Sib/ <i>rys</i> logZ	Combined logZ	logZR
-1.117572e6 ± 0.46	-1.161415e6 ± 0.37	<b>-2.278988e6 ± 0.59</b>	-2.279356e6 ± 0.45	368.49 ± 0.74

Methods in Section 13.1.17. Code in Section 17.8.33.

BioMotifInference’s source detection was highly conservative. The background models were found to explain a majority of observed nucleosome positions adequately, without any PWM sources, in most models of the maximum a posteriori estimate, for both sibling and *rys* ensembles. In siblings, the top three sources are the only ones detected in more than 10% of observed sequences, suggesting that the influence of sequence preferences is weakly explanatory for these sites. By contrast, in the mutant *rys* positions, we found twenty-eight separate PWM sources detected in more than 10% of observations, in a variety of posterior modes. This indicates sequence preferences are more explanatory for the *rys* differential position set.

The CWG motifs detected in sibling sequences are commonly reported in nucleosome positions, and have been described as promoting nucleosome formation [Has03]. The majority of the sources detected in the various posterior modes were of this form, with rarer CT- and CA- dinucleotide repeats, as well as a rare ATGG repeat. *rys* positions have a much more diverse set of sources detectable above background genomic noise; notably, AG- dinucleotide repeats that are not found in *sib* sources at all. The CWG motifs also exhibit a preference for flanking A positions that is not evident in the sibling differential position set. CA- dinucleotide repeats are more commonly detected in *rys* positions, and the rare ATGG repeat is not found at all.

---

<sup>4</sup>That is, the convergence criterion was that the ensemble difference  $\log(L_{max}) - \log(L_{min})$  be <125).

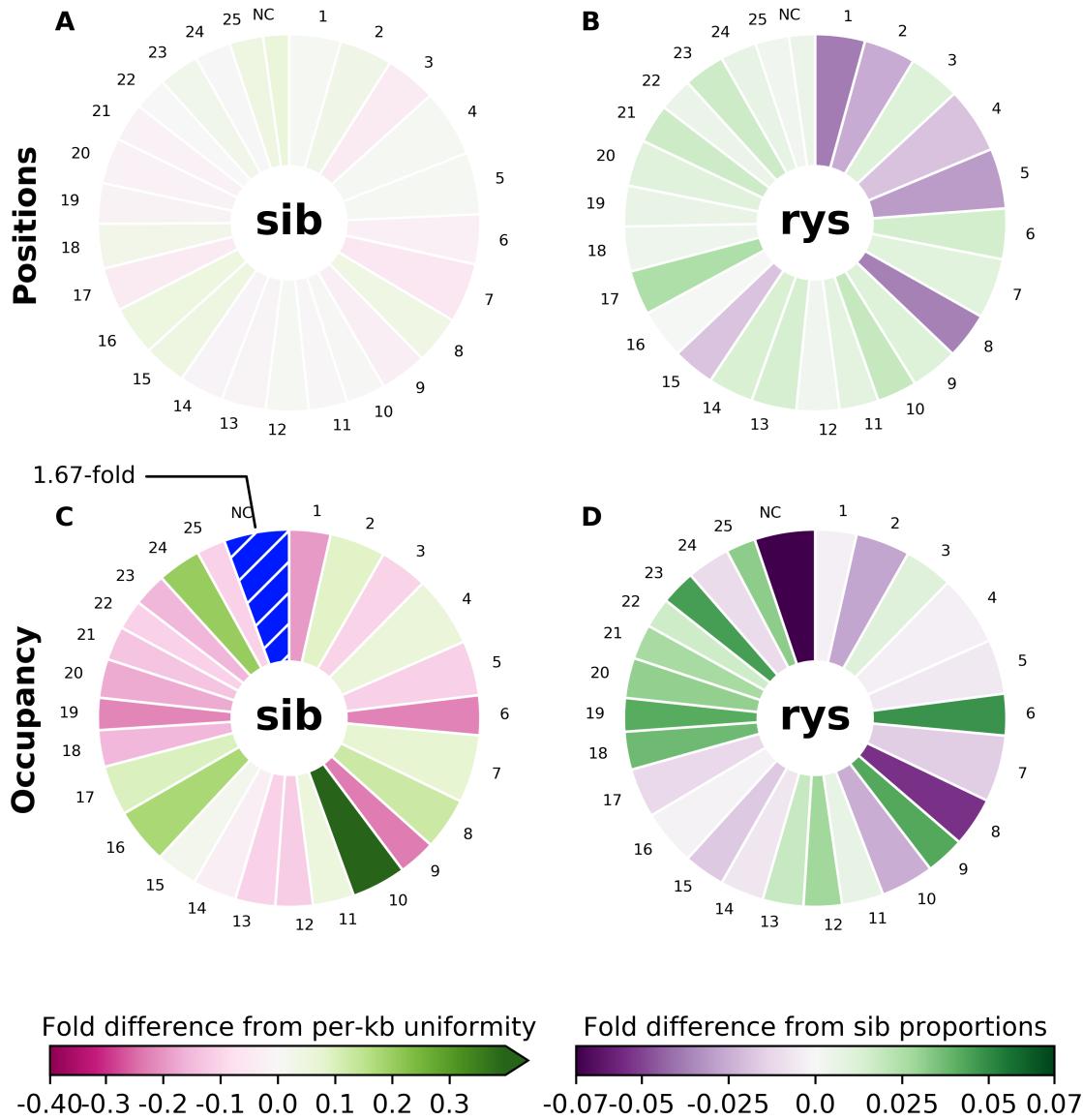
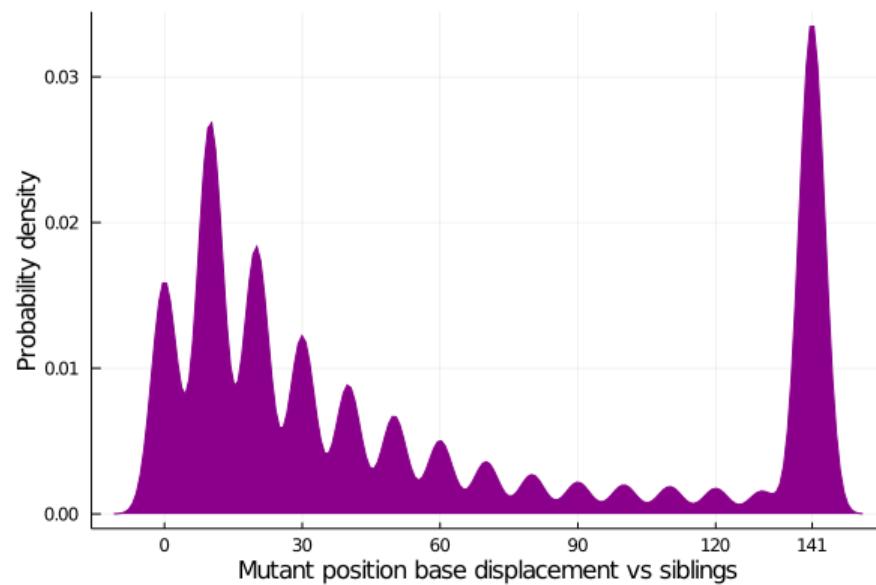
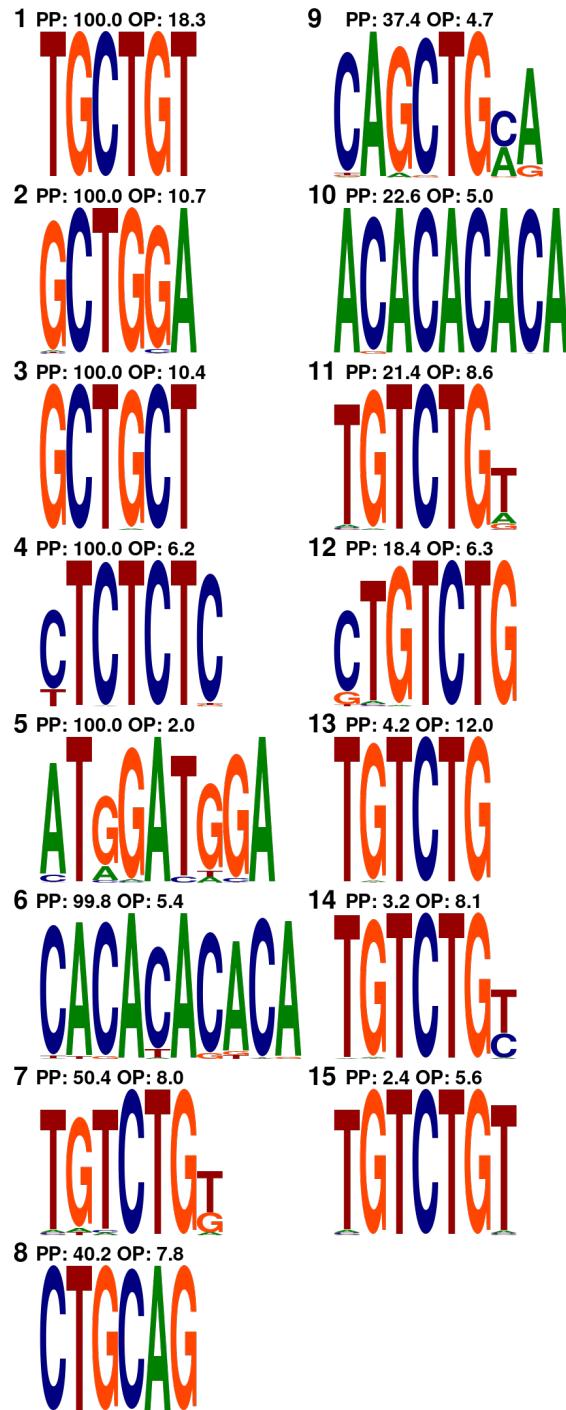


Figure 5.16: *rys* chromosomes are differentially enriched and depleted of nucleosome position density and occupancy.

Pie charts of nucleosome position density and occupancy by chromosome. Width of pie slices in panels A and B indicates the fraction of the total number of positions occurring in the numbered chromosomes and nonchromosomal scaffolds (NC). In panel A, depicting the sib genome, slices are colored according to the extent of deviation from an assumption of even distribution of nucleosomes across the genome. In panel B, depicting the *rys* genome, slices are colored according to the extent of deviation from the sib distribution. The width of slices in slices B and D indicate the fraction of the total nucleosome occupancy signal detected. Slice coloration depicts deviations from nucleosome occupancy distributions analogous to position distributions in A and B. Blue and white diagonal bars in the NC slice of panel C denote an out-of-scale positive deviation from the assumption of even distribution, i.e., sib NC scaffolds have 1.67-fold more of the total nucleosome occupancy signal than is expected from their length alone. Methods in Section 13.1.15. Code in Section 17.8.22.



**Figure 5.17: PWM sources detected in sibling differential nucleosome positions.**  
Probability distribution of *rys* position displacement distance from mapped sibling positions, in base pairs. Methods in Section 13.1.16. Code in Section 17.8.39.

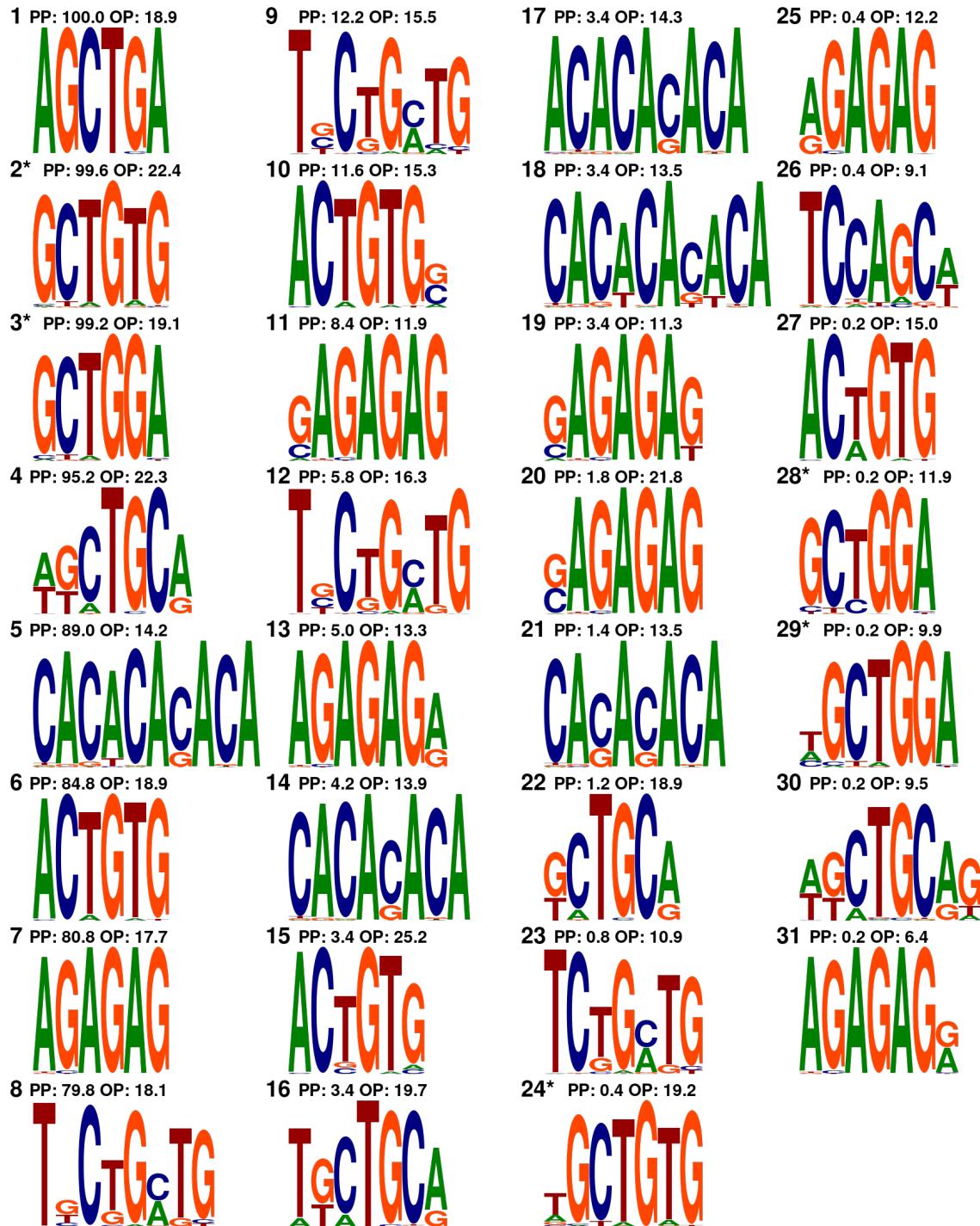


**Figure 5.18: PWM sources detected in sibling differential nucleosome positions.**  
PWMS presented as sequence logos with letter height proportional to position informational contentemission probability.

PP: Posterior prevalence; proportion of models within the compressed maximum a posteriori ensemble which have this PWM as a signal source

OP: Observation prevalence; mean proportion of observed position sequences this sequence is used to explain, over all posterior samples

Methods in Section 13.1.17. Code in Section 17.8.33.

Figure 5.19: PWM sources detected in *rys* mutant differential sources.

PWMs presented as sequence logos with letter height proportional to position informational contentemission probability.

PP: Posterior prevalence; proportion of models within the compressed maximum a posteriori ensemble which have this PWM as a signal source

OP: Observation prevalence; mean proportion of observed position sequences this sequence is used to explain, over all posterior samples

Methods in Section 13.1.17. Code in Section 17.8.33.

### 5.3 Discussion

We began our investigations by revising the original description of the *rys* mutant CMZ phenotype. The apparently increased size of the CMZ in *rys* mutants, relative to siblings, is produced as an effect of one or more of the following phenomena:

1. Inappropriately retained RPCs in animals older than 7dpf
2. Reduced neural population of the specified central retina, relative to the CMZ
3. Expanded and spread-out RPC nuclei

The first two effects are consistent with RPCs failing to specify as particular retinal neural lineages, demonstrated by retention of early progenitor markers in these cells. *rys* RPCs are not, however, arrested in cell cycle. Unlike systems where elevated polyadenylated histone transcript levels resulting in cell cycle arrest [KKT<sup>+</sup>13], we find that the mitotic activity of these cells actually accelerates under an overabundance of polyadenylated H2A and H2B transcript. We therefore suggest that the *rys* phenotype is best characterised by chromatin disorganisation (phenomenon 3) and a failure of RPCs to correctly specify (which covers phenomena 1 and 2) as retinal neural subtypes. The enlarged appearance of the CMZ is a result of these effects, and not a general increase in the CMZ population, RPC cytoplasmic volume<sup>5</sup>, or other variables.

Moreover, in identifying npat as the lesioned gene underlying the *rys* phenotype, we nominate a plausible macromolecular mechanism underlying phenomenon 3 which could produce phenomena 1 and 2. Mutant npat's effects on cell-cycle dependent histone transcription and stability may result in aberrant nucleosome positioning within proliferating cells, by altering the pool of histone proteins available for nucleosome formation. We posit that this is what causes the observed loss of a subpopulation of sibling nucleosome positions in *rys*, together with the gain of novel, aberrant positions. When we investigated whether the sequences of these subpopulations were better modelled by separate emissions processes than a shared emissions process, we found substantial evidence in favour of separate emissions processes, suggesting that nucleosome localisation to unique *rys* positions occurs by different mechanisms than those unique to sibs. Interestingly, the PWM signals we detected in this differential nucleosome position set suggest some detail as to how changes in histone expression might result in the observed nuclear disorganisation of *rys*. The observation that the background models explain the positions unique to siblings better than those unique to mutant *rys* strongly suggests that sequence preference is less important in the process generating sibling positions than in the one generating mutant positions.

The relative importance of primary sequence in nucleosome positioning has been hotly debated; some have advocated for a nucleosome positioning code intrinsic to primary sequence [KMF<sup>+</sup>09], while others have disavowed the existence of any such code [ZMR<sup>+</sup>09]. In general, the nucleosome dynamics community has moved on from these discussions in favour of emphasis on rotational sequence preferences [TCM<sup>+</sup>07] and translational nucleosome positioning by *trans*-acting factors [KKZ<sup>+</sup>18]; a common interpretation of the earlier debate is that sequence preferences tend to dominate only at limiting concentrations of histones, when chromatin formation is rare [Poi13]. The fact that sequence preference is less explanatory for the sibling positions than for the mutant ones is suggestive against this background. It seems likely that the shifted and novel nucleosome positions observed in mutants reflects all of the following:

---

<sup>5</sup>That is, alone, without an increase in nucleus size. We did not directly measure this parameter.

1. The appearance of nucleosomes with aberrant subunit composition
2. A loss of translational control over *rys* mutant nucleosomes by *trans*-acting factors
3. The limiting concentration of aberrantly-expressed nucleosomes, resulting in increased influence of “default” sequence-preference positioning

All of the above could plausibly be caused by disrupted histone expression arising from altered npat activity. As regulation of chromatin dynamics is known to be important for both replication timing [GTR<sup>+</sup>10] and cell identity and fate [SVT13], altered chromatin architecture could explain how mutated npat protein produces the observed shift in proliferative activity and failure of RPCs to specify, via altered histone expression. Moreover, chromatin density has recently been specifically implicated as fundamentally involved in the process by which pluripotent cells restrict gene expression to achieve their stable specified fates [GJH<sup>+</sup>17]; it may be this overall process which has been disrupted in mutant *rys*.

It is possible that the sequence-level inference is compromised by the ad-hoc nature of the sampler used in our nested sampling procedure. For reasons explained in Chapter 10, the ICA model structure makes euclidean space representations difficult, and ensuring sampling is in detailed balance may not be possible, given changing PWM signal lengths. We suggest that it is appropriate to magnify our error estimate. 100-fold more error than estimated has the effect of reducing the significance of the result from 498 to 4.98 standard deviations, which is still in the vicinity of the  $5\sigma$  significance typically accepted as a discovery in the physical sciences. We conclude that it is more likely that the differential nucleosome populations found in *rys* mutants and their phenotypically normal siblings are the result of separate generative processes. It is also possible that our identification of the differential subset of nucleosome positions with the disordered chromatin present in *rys* RPCs is inaccurate. We have not conducted a full survey of *rys* proliferative niches, and it is not clear how widely affected other cell types might be. Still, within the retina, only proliferative cells display the chromatin phenotype, and it is on this basis that we make the identification, which is the most parsimonious available.

There are a number of important caveats for this explanation. It remains unclear what form of the npat protein is expressed, if any. The overall effect on the expressed pool of histones in RPCs also remains uncharacterised. It is likely thhat there are macromolecules involved in the *rys* pathology which intermediate between the npat lesion and the observed chromatin phenotype which we have not examined. We suggest, however, that the overall effect on specification must be due to a failure of RPCs to organize chromatin for specification, and that, interestingly, this failure does not prevent proliferation. *rys* therefore presents a useful model of the dissociability of proliferative and specifitative behaviours in neural progenitors, which has recently been documented elsewhere in the developing *D. rerio* retina [ESY<sup>+</sup>17]. Given the methodological limitations we encountered in probing protein expression in *rys* RPCs, the most interesting and productive avenues of research to pursue in *rys* pertain to this chromatin organisation phenotype. Further experimentation is required to determine how particular changes in chromosome organisation (e.g. alterations in 3-dimensional chromatin organisation of gene expression, number of replication foci, etc.) produce specific features of the *rys* phenotype.

The zebrafish *rys* mutant model provides a unique opportunity to study the role of a human NPAT homologue in a complete, developing tissue. This has previously proven difficult using mouse Npat, proviral inactivation of which resulted in early embryonic arrest, prior to tissue formation [DFWV<sup>+</sup>97]. The survival and development of *rys* mutant embryos beyond this stage may reflect the presence of wild-type npat transcript contributed maternally [HSK<sup>+</sup>13]; *rys* mutants nevertheless do not survive

beyond metapmorphosis (~21dpf), so npat seems to be similarly obligatory for normal development in zebrafish, if over a longer timeframe. Still, we observed many differences between the function of npat in zebrafish and documented effects in other vertebrates, which require some explication.

As teleost fish are known to have undergone whole-genome duplication subsequent to their radiation from vertebrates, it is possible that zebrafish may have multiple npat paralogues. We have excluded this possibility by failing to identify any significantly similar CDS sequences in the Zv9 zebrafish genome using BLAST, and the synteny analysis in Figure 13.5. The zebrafish npat gene does have a substantially different genomic context from human NPAT, however: it is not associated with the eponymous ATM locus (which has been duplicated, and is present in this duplicate form elsewhere on chromosome 15). ATM's 5' position is, in zebrafish, occupied by hif1al, with the intergenic region lacking canonical E2F promoter sites. Of the 80 vertebrate genomes currently available from Ensembl, this organisation is shared only with the cave fish (*A. mexicanus*), with the human-like ATM-NPAT association preserved in all other species. This apparently evolutionarily novel genomic organisation for npat may be responsible for some of the differences in npat function we report here, relative to its homologues.

We identified alterations in histone mRNA transcription and 3' end processing as likely mechanistically involved in the *rds* phenotype. In both 6 and 8 dpf *rds* larvae as well as npat-morpholino treated 1dpf embryos, we observe increased abundances of total and polyadenylated histone transcripts. The mechanism by which these effects are produced in *rds* remains unclear. The increases in histone transcript abundance observed in both mutant and morpholino-treated animals are at odds with observations that NPAT knockouts display decreases in histone transcription [YWNH03], and that the destruction of CDK phosphorylation sites on NPAT protein, which would be the case in the putatively truncated *rds* npat protein, result in similar declines in histone transcript abundance [Ma00, MGv<sup>+</sup>09]. This implies the role of zebrafish npat in regulating histone transcription differs from human NPAT. We are unable to determine from these data whether the overall increases in histone transcript abundance are a consequence of increased histone transcription, or of the greater stability of improperly polyadenylated histone transcript. The increased abundance of polyadenylated transcript in *rds* mutants and npat morpholino-treated embryos is consistent with the observed role of NPAT in recruiting CDK9 to replication-dependent histone gene clusters, known to be important in generating the normal stem-loop structure at the 3' end of these transcripts [PSS<sup>+</sup>09], suggesting that this role is conserved in zebrafish. It is also possible that particular histone genes give rise to polyadenylated transcripts in zebrafish, as observed in a variety of cell lines [KKT<sup>+</sup>13]. Increased transcription of these particular genes might also account for the observed increase in polyadenylated histone transcript abundance. Although increased abundance of polyadenylated histone transcript has been associated with both cell cycle arrest and differentiation [KKT<sup>+</sup>13], we found that in the *rds* mutant CMZ, this was associated with altered cell cycle parameters, failure to differentiate, and ultimately, cell death by apoptosis. This suggests that the presence of polyadenylated histone transcripts are not directly related to particular cell cycle states or differentiated fates, but rather that a particular regime of coordination and control of the expression of polyadenylated and replication-dependent, stem-looped histone transcripts is required to achieve appropriate transitions between these states. In the case of *rds*, this seems to be related to the effect of altered histone expression on chromatin structure. Future experiments focusing on npat could include the testing of cell autonomous versus non-autonomous effects by the use of conditional knockouts in mice, for example by using a floxed npat gene in combination with tissue-specific promoter expression of Cre. Another avenue in zebrafish could be the selective interference of e.g. H2A and H2B expression

by morpholino, to directly assay the effect of histone expression on chromatin organisation.

# Chapter 6

## Inferring and modelling specification dynamics in retinal progenitors

### 6.1 Specification as a function of non-proliferative chromatin dynamics

The npat mutant *rys* investigated in Chapter 5 reinforces a basic theme running through this thesis, that lineage commitment or specification can be, and often is, dissociable from the proliferative behaviour of retinal progenitors, as has been suggested by Engerer et al. [ESY<sup>+</sup>17]. *rys* illustrates this: RPCs undergo a final burst of proliferation before the death of the animal<sup>1</sup>, but never contribute appropriately to the neural retina. Indeed, older *rys* larvae display a proliferative spillage of aberrant RPCs into the intraocular space between the lens and GCL; the progeny of these aberrant RPCs literally cannot be integrated into the tissue.

The importance of this dissociability for our inferences about RPCs can be gleaned from the basic problems encountered in each of the modelling studies. To begin with, the Stochastic Mitotic Mode Explanation models examined in Chapter 2 failed, in large part, because the underlying Simple Stochastic Models conflate proliferation and specification under the “mitotic mode” concept. This leads immediately to the introduction of model structure to explain specification outside of “mitotic mode”, assuming a phased temporal structure to RPC behaviour, without attempting to explain it. Lineage specification and niche exit processes need to be modelled separately from mitotic processes to differentiate their effects on outcomes. Next, proliferative data and morphological measurements in Chapter 4 fail to constrain the posterior distributions on the rate of niche exit in simple phased models, demonstrating that this parameter cannot be inferred with any certainty from overall population-level proliferative dynamics. To accurately model the behaviour of RPCs, data which provides accurate volumetric or temporal constraints on modelled specification or niche exit rate is of paramount importance. Finally, in *rys*, we encounter the problem of explaining RPCs that proliferate but fail to specify; we explain this phenomenon in terms of the effects of mutant npat on chromatin organization by way of effects on histone expression. This directs us to the relevant level of organization for explanation of RPC lineage outcomes: the progression of chromatin states that underlay long-term changes in gene expression, which

---

<sup>1</sup>Likely, on this view, arising from systemic failure of progenitors to specify appropriately.

are required for the generation of the specified protein complement. Our approach, while providing evidence for the chromatin dynamics we suggest arise from the effects of npat on the available histone pool, does not give us insight into how the appropriate progression of chromatin states might be usefully parameterised.

While a large number of techniques for studying aspects of chromatin structure exist, few suggest a means to describe an overall “nuclear state”. That is, if we consider Waddington’s idea of a cell’s lineage state being a ball rolling down a canalized terrain [Wad57], it remains unclear what sort of parameter space would allow us to infer where the ball is. Older ideas of cellular state space, that rely exclusively on measures of protein expression, are unable to address hypotheses about chromatin structure. It remains unclear what aspects of chromatin organisation are causally important in fate commitment, as little formal model comparison has taken place. One might propose a joint structure-sequence-expression space with thousands of variables drawn from the vast literature on chromatin organisation. Such a project is unlikely to succeed with reasonable compute budgets, even with the sophisticated Bayesian inference tools developed here.

Still, if it is not obvious how one might build a general model of the evolution of “nuclear state”, we may suggest some more limited approaches arising from the work presented in this thesis, that may make some headway here.

## 6.2 Future directions

*ryst* provides a useful platform for investigating the processes required to organize the progression of chromatin through the sequence of states we presume to be required for progression through the series of stages implied by the studies presented in Chapter 2 and Chapter 4, and the associated shifts in RPC lineage contributions. Despite the evidence amassed for the involvement of npat effects on nucleosome positioning in *ryst* mutants, many aspects of the suggested mechanism remain obscure. This arises, in part, from the lack of reliable macromolecular tools for investigating npat and histone proteins in *D. rerio*. It would be useful to focus inquiry here on a small number of candidate histones. However, the large number of zebrafish histone genes, with no crystallographic data available, suggests that prediction of the involvement of particular histones would be difficult.

This situation is now rapidly changing, however. The AlphaFold algorithm has recently been recognized as a solution to the Critical Assessment of protein Structure Prediction (CASP) problem set in the CASP13 competition [AIQ19]. This is the first time protein folding has been adequately predicted *in silico* from sequence data alone. If it is possible to use this algorithm to predict *D. rerio* nucleosome structures from histone folding predictions, this may obviate the need for crystallographic data. Such structural predictions could be used with the posterior estimates of signals putatively arising from nucleosome-DNA contact produced in Figure 5.19 to, in turn, predict which nucleosome compositions are most likely to favour the detected sequences. This might be assisted by 3d pattern matching for DNA sequences [HPG07], or roll-and-slide modelling of the contact motifs [TCM<sup>+</sup>07]. Such a study would produce hypotheses that, if interesting enough to pursue *in vivo*, would supply estimates of how likely various histones are to be involved in the *ryst* mutant phenotype, and in what regard, which would help prioritise resources for the development of assays for interrogating the specific transcript and protein status of the most likely loci.

A more fundamental way of interrogating *ryst* “nuclear state” might be provided by focusing on

transcription itself, which has been proposed as a primary driver of chromatin organisation [CM18]. Such a project would likely require a transgenic line developed to monitor the transcription of one or more loci of interest in live cells, perhaps using bio-orthogonally labelled transcripts, as has recently been demonstrated in zebrafish [WCG<sup>+</sup>20]. This approach could possibly be used either to investigate the status and trafficking of npat and histone transcripts themselves, or to determine the impact of the npat lesion on the transcriptional shaping of chromatin organisation by tracking the transcription of progenitor markers, for instance.

Ultimately, it is likely that models that allow the prediction and control of RPC behaviour *in situ* in zebrafish retinae will treat the chromatin organisation of RPC nuclei as an integrator of multiple relevant sources of information, including physical forces, cytoskeletal dynamics, and extracellular signals. This type of functional organisation, in which biological “meaning” arises from the integration of salient cellular information by a combination of nonarbitrary physicochemical relations and arbitrary biological coding reltaions, is currently best described by the subdiscipline of biosemiotics [Hof08, Fav15, Hof15]. Of particular interest here is the clear-eyed and unobscurantist work of Marcello Barbieri, which may well provide the metaphysical framework necessary to plumb the depths of this topic [Bar14].

## **Part II**

# **Software Technical Reports**

# Chapter 7

## GMC\_NS.jl: Nested sampling by Galilean Monte Carlo

`GMC_NS.jl` implements a basic Galilean Monte Carlo nested sampler [Ski12, Ski19] in pure Julia. It uses the improved Galilean reflection scheme of Skilling’s 2019 GMC publication [Ski19], with increased performance and detailed balance preserved. It uses the natural attrition of model-particles getting stuck in posterior modes to regulate the number of live particles in the model ensemble. This approach partially achieves the same end as algorithms which dynamically adjust the ensemble size based on parameters of the ensemble [FHB09, HHHL19]. A minimum ensemble size may be supplied by the user, which is maintained by initializing a new trajectory isotropically from the position of an existing particle; this ensures that the ensemble may be sampled to convergence.

It should be noted that `GMC_NS.jl` is a prototype; while it finds posteriors in relatively few iterates, it oversamples less important early regions of the posterior. See Section 7.3.5 for more information.

### 7.1 Implementation notes

The basic sampling scheme followed is, first, to initialize an ensemble of models with parameters sampled evenly from across the prior, then to compress the ensemble to the posterior by applying GMC moves to the least-likely model in the ensemble at a given iterate, constrained within its current likelihood contour. GMC tends to move model-particles across the parameter space evenly and efficiently, particularly in the initial portion of compression across the uninformative bulk of the prior mass. That said, GMC particles may get “stuck” in widely separated minima, so there are instances when the least-likely particle has no valid GMC moves. In this case, the trajectory is not continued. Instead, the trajectory is removed from the ensemble and sampling continues, unless the number of particles in the ensemble would decline below the specified minimum. In this case, a new trajectory is begun by resampling from the remaining prior mass within the ensemble, as in vanilla nested sampling. This is accomplished by random “diffusive” proposals, in contrast to the ordinary, directed Galilean movements. A random particle remaining within the ensemble is selected, and isotropic velocity vectors are sampled from this particle to find a starting location for the new trajectory (this vector is conferred to the new particle).

`GMC_NS.jl` follows the common convention of transforming parameter space to a hypercube. The density of the prior is uniform, that is  $\pi(x) = 1$ , over the -1:+1 range of the hypercube dimension (this

is referred to as the “unit ball” in the code). Utility functions `to_unit_ball` and `to_prior` transform prior coordinates to the unit ball, and unit ball positions to the prior coordinate, respectively. In future versions, this scheme is likely to be modified to a 0:1 unit hypercube, which does not require operations beyond obtaining the cumulative probability of a parameter value on the prior distribution. The user may also specify a sampling “box” to bound particles from illegal or nonsensical areas of the parameter space (eg. negative values for continuous distributions on positive-valued physical variables). The box reflects particles specularly in the same manner as the likelihood boundary, with the exception that because the orientation of the  $n$ th-dimensional box “side” is known, the reflection is performed by stopping the particle at the box side and giving it reversing its velocity vector in dimension  $n$ .

`GMC_NS.jl` attempts to maintain efficient GMC sampling by per-particle PID<sup>1</sup> tuning of the GMC timestep [VV12]. GMC treats models as particles defined by their parameter vectors, which give their positions in parameter space, and a velocity vector, initialised isotropically and only changed when the particle “reflects” off the ensemble’s likelihood contour. In GMC, when a model-particle is moved through the parameter space (in this case, because it is the least likely particle in the ensemble), a new position is proposed by extending some distance along its velocity vector. This distance is determined by scaling the velocity vector by a “timestep” value, which defines the per-iterate rate at which the particle traverses the parameter space. As the model ensemble is compressed to the posterior, this timestep must decline in order for proposals to be acceptable, as the amount of available parameter space declines rapidly. Additionally, GMC particles may enter convoluted regions of parameter space that form small likelihood-isthmuses to other, more likely regions. In order to deal with this, each trajectory is assigned its own PID tuner, which maintains an independent timestep value for the trajectory. The PID algorithm is tuned to target an unobstructed move rate. In short, PID tuning will reduce the timestep if the particle is frequently reflecting off the likelihood boundary (in which case it is crossing the remaining prior mass without sampling much from within it, or it is in a highly convoluted area of the local likelihood surface), and extend the timestep if it the particle frequently moves without encountering the boundary, so that particles that are closely sampling an open region without encountering the boundary begin to “speed up”.

PID constants and default GMC settings have been crudely tuned to produce the best possible result with the stated sampling scheme on the eggbox problem, as described in Section 7.3.5.

## 7.2 Ensemble, Model, and Model Record interfaces

In operation, `GMC_NS.jl` maintains a `GMC_NS_Ensemble` mutable struct in memory. The directory to which the sampler ought to save the ensemble is specified by its `path` field. `GMC_NS.jl` does not maintain calculated models in memory, instead serializing them to this directory. In order to track the positions and likelihoods of model-particles within the ensemble and as samples from the posterior, a `GMC_NS_Ensemble` maintains two vectors of `GMC_NS_Model_Records` as fields; `models` for live particles, and `posterior_samples` for the previous positions along trajectories. Observations against which models are being scored, prior distributions on model parameters, model constants (if any), and the sampling box are specified by the `GMC_NS_Ensemble`’s `obs`, `priors`, `constants`, and `box` fields, respectively. The

---

<sup>1</sup>PID stands for Proportional, Integral, Derivative, which nominate the three constants used to tune a process variable by the error of a measurement variable from a supplied setpoint. The P term is directly proportional to this error, the I term is proportional to the integral of the error over time, and the D term is proportional to the rate of change of the error.

`GMC_NS_Engsemble` also specifies settings for the GMC sampler and the PID tuner. Sensible defaults for these values may be passed (usually, splatted<sup>2</sup>) en masse to an ensemble constructor with the `GMC_DEFAULTS` constant exported by `GMC_NS.jl`.

Therefore, to use `GMC_NS.jl` with their model, the user must write a model-appropriate version of these three structs (the ensemble, model, and model record) with the fields specified by the docstrings in the `/src/ensemble/GMC_NS_Engsemble.jl` file, as well as `/src/GMC_NS_Model.jl`. The user is responsible for an appropriate constructor and likelihood function for the model, any required parameter bounding functions, and so on. The user may optionally write a overloaded `Base.show(io::IO,m::GMC_NS_Model)` function for displaying the model, or a `Base.show(io::IO,m::GMC_NS_Model,e::GMC_NS_Engsemble)` function for displaying the model with observations or other ensemble data. This function can be used with the package's `ProgressMeter` displays for on-line monitoring of the current most likely model.

## 7.3 Usage notes

### 7.3.1 Setting up for a run

Given an appropriately coded model, as outlined above, the user must define a number of important values for the sampler before a `GMC_NS_Engsemble` may be constructed. These values are stored in fields of the ensemble. Firstly, the user must prepare the observation data in whatever format is suitable for the model's likelihood function. Next, one must define a vector of `Distributions` on parameters of the model as the `prior` field of the ensemble. These must be univariate, as the CDF of the distribution is used to determine position on that dimension. Some functions to produce univariate marginal distributions from multivariate Normal Gamma and Normal Inverse Gamma priors are available in `NGRefTools.jl`. Next, any constants used in the likelihood function must be available in the `constants` field of the ensemble. A `box` matrix with  $n$ -dimensional rows containing minimum and maximum values in the first and second columns defines the maximum extent of parameter space to be explored. The sampler expects the box in the transformed hypercoordinates mentioned above; a one-dimensional box across the entire prior would be  $[-1 1]$ . It is convenient to compose the matrix in the original parameter space and perform the transformation with a broadcast call to `to_unit_ball()`, either in the ensemble constructor or in the calculation script.

### 7.3.2 GMC and PID tuner settings

The GMC sampler is governed by settings specified in `GMC_NS_Engsemble` fields. These include the PID tuner settings, and are summarized below:

`GMC_Nmin` (Integer; Default 50): The minimum number of active model-particles in the ensemble. After the initial sample from the prior, new particles will only be generated when required to maintain this value. Otherwise, when a valid proposal cannot be found for the current least-likely trajectory (i.e. the PID tuner drives the timestep below `GMC_τ_death`), the nested sampling step consists of removing that particle to the posterior samples and moving on to the next live particle, without replacement.

---

<sup>2</sup>"Splattting" in Julia refers to the expansion of a argument vector into a function's call by using the ... operator following the vector's name, i.e. `function(arguments...)`.

**GMC\_τ\_death** (Float; Default  $1e - 5$ ): Value of trajectory’s  $\tau$  timestep, below which the particle is considered to be dead. A dead particle’s trajectory is not continued. This value is denominated in -1:1 hypercube coordinates. If this value is very small ( $\leq 1e - 6$ ), trajectories may carry on for thousands of iterates in relatively unimportant regions.

**GMC\_init\_τ** (Float; Default 1.): Initial value of the timestep for all trajectories. This applies to the particles assembled on ensemble construction; particles generated from diffusional resampling inherit the  $\tau$  of their parent, but with the isotropic vector which generated the proposal for their new location.

**GMC\_tune\_μ** (Integer; Default 4): Length of tuner memory; the proposal acceptance rate  $\alpha$  is calculated over the previous  $\mu$  constructor reports for this trajectory. Tends to produce oversampling if it is much more than 10.

**GMC\_tune\_α** (Float; Default .25): Target acceptance rate for proposals. Sampler efficiency benefits from this being fairly low (.2-.4).

**GMC\_tune\_PID** (NTuple3,Float; Default (.3,0.,0.)): kP, kI, and kD constants for the tuner. kP can be somewhat larger. kI and kD are probably detrimental when using short tuner memory, but could be useful if an efficient longer memory regime is available.

**GMC\_timestep\_η** (Float; Default .25): If this value is  $>0.$ , a normally distributed error with a standard deviation of  $\tau \cdot \text{GMC\_timestep}_\eta$  is applied to  $\tau$  before calculating the GMC proposals, as suggested by [Ski12].

**GMC\_reflect\_η** (Float; Default .005): If this value is  $>0.$ , it is used to perturb reflection vectors, as suggested by [Ski12]. It is applied to sampling box reflections as well as likelihood contour reflections. It should be a small value. If it is too large, sampling box reflections can “bend” the wrong way and produce an inappropriate series of samples at the box boundary.

**GMC\_exhaust\_σ** (Float; Default 10.): Scale factor multiplied into ensemble size to determine the maximum number of unsuccessful searches to perform before terminating sampling. This virtually never happens.

**GMC\_chain\_κ** (Integer, Default typemax(Int64)). Number of iterates at which to terminate a sampling chain/trajectory. By default, this does not occur. Can prevent oversampling in some cases.

### 7.3.3 Parallelization

`GMC_NS.jl` does not have an explicit parallelization scheme; the sampler proceeds linearly to the next least-likely particle and performs the appropriate Galilean trajectory sampling procedure. Parallelization may nonetheless be achieved at two levels; the likelihood function may be parallel according to the needs of the user, and individual nested sampling runs may be arbitrarily combined. That is, if one desires to parallelize a `GMC_NS.jl` job, it is best to think in terms of the total number of trajectories to be sampled, dividing this by the number of machines available to perform the sampling work, to obtain the number of particles to be sampled on each machine. The sampling runs can then be combined post hoc to achieve the desired final accuracy. In this scheme, the likelihood function is best parallelized by threading on a single machine.

### 7.3.4 Displays

Parameters of the ensemble and most-likely model can be monitored on-line, while `GMC_NS.jl` is working. This is done by specifying a vector of function vectors (ie a `Vector{Vector{Function}}`). One such

vector may be supplied to specify the displays to be shown above the `ProgressMeter`, one for those below, supplied as the `upper_displays` or `lower_displays` keyword arguments to `converge_ensemble!()`. `GMC_NS.jl` will rotate the upper and lower displays to the next vector of display functions every `disp_rot_its` sampling iterates, supplied as another `converge_ensemble!()` keyword argument. Default display functions are available in `/src/utilities/progress_displays.jl` and are exported for easy composition of the function vectors.

### 7.3.5 Eggbox diagnostic

The “eggbox test” is a toy model designed by Feroz et al. [FHB09], and used by others for use in characterising nested sampling algorithms [Buc16]. It uses the likelihood function  $\log L = (2 + \cos(5 \cdot x_1) \cdot \cos(5 \cdot x_2))^5$ . Within the unit square, this function has 18 optima and a canonical logZ evidence score of 235.88. The most efficient solution to this problem is afforded by MultiNest; it accurately sums the model evidence in approximately 10000 iterates with 400 particles. `GMC_NS.jl` correctly finds the 18 optima of the eggbox, and converges in fewer iterates, as displayed in Figure 7.1. However, it substantially underestimates the model’s evidence; typical `GMC_NS.jl` logZ evidence estimates for the eggbox are consistently 229. This tendency to systematically underestimate model evidence should be kept in mind when examining the evidence estimates in this thesis. The cause of this underestimation is likely an oversampling of the prior volume arising from the minimum allowable timestep not being tied to the volume of prior mass remaining; this allows trajectories to contribute many closely-sampled points from the peripheral prior volume where the trajectory repeatedly encounters the likelihood contour (in the “corners” or less-likely peripheral modes). Suggestions for fixing this issue are given in Section 7.4.

The eggbox may be estimated with the following code:

---

```

1  using GMC_NS, Distributions
2
3  #compose priors
4  prior=Uniform(0,1); priors=[prior,prior]
5
6  #sampling box
7  box=[-1. 1.
8      -1. 1.]
9
10 #monitor convergence
11 lds=Vector{Vector{Function}}([[ensemble_display]])
12
13 #terminate slowly moving particles
14 gmc=copy(GMC_DEFAULTS)
15 gmc[2]=1e-3
16
17 #assemble 400 particle ensemble
18 e=Eggbox_Engsemble("eggbox_test", 400, priors, box, gmc...)
19
```

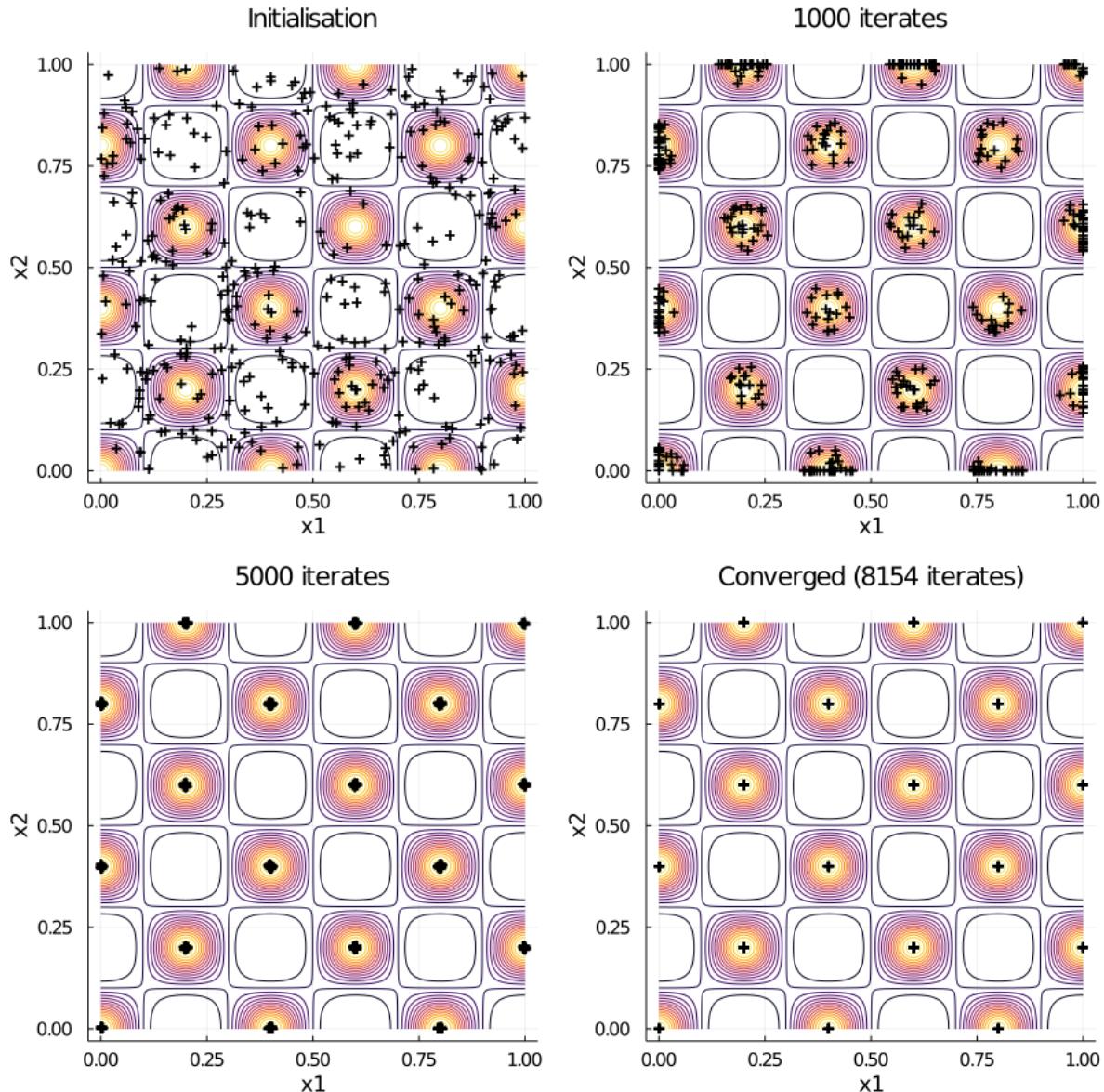


Figure 7.1: GMC-NS.jl nested sampling the eggbox toy model

---

```

20  #converge the ensemble
21  converge_ensemble!(e,backup=(true,100), lower_displays=lds, converge_factor=1e-6)

```

---

### 7.3.6 Example use

`GMC_NS.jl` is supplied with two simple, but useful, statistical models by default. These are the `Normal_Model` and `LogNormal_Model` subtypes of `GMC_NS_Model`. Users may familiarize themselves with `GMC_NS.jl`'s operation by the use of these models. A simple demonstration is supplied in the `/test` directory of the package. The program is detailed with additional comments below. The demonstration involves

converging an ensemble of 10 `Normal_Model` chains initialized from an inaccurate `NormalGamma` prior on data generated from a known `Normal` distribution.

---

```

1 #Import dependencies
2 using GMC_NS, Distributions, ConjugatePriors
3
4 #Generate some data from a Normal model with \mu 100. and \sigma 5.
5 sample_dist=Normal(100.,5.)
6 samples=rand(sample_dist, 10000)
7
8 #An inaccurate ConjugatePrior
9 prior=NormalGamma(300.,2e-3,1.,10.)
10
11 #Box for the two parameters of the Normal model.
12 #First column: parameter minimums.
13 #Second column: parameter maximums
14 #First row: \mu
15 #Second row: precision, ie 1/sqrt(\sigma)
16 #eps() is machine epsilon, ~1e-16; a small number
17 box=[0. 1000.
18      1/1000. 1/eps()]
19
20 #Start with the default sampler settings
21 #Index 1: Lower the minimum number of ensemble particles to 5
22 gmc=GMC_DEFAULTS
23 gmc[1]=5
24 gmc[2]=1.e-15
25
26 #Assemble the ensemble. Begin with 10 particles
27 e=Normal_Ensemble("NE_test", 10, samples, prior, box, gmc...)
28
29 #Define the upper displays to be rendered above the status line. Here we rotate a
30   ↳ single display between three readouts.
31 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]])
32
33 #Define the lower displays to be rendered below. We want to look at the most likely
34   ↳ model in the ensemble plotted against the observations the whole time.
35 lds=Vector{Vector{Function}}([[model_obs_display]])
36
37 #Converge the ensemble, serializing to pwd/NE\test every 100 iterates and rotating
38   ↳ the chosen displays every 1000 iterates.
39 converge_ensemble!(e,backup=(true,100),upper_displays=uds, lower_displays=lds,
40   ↳ disp_rot_its=1000)
```

Typical output after executing this program is presented in Figure 7.2. The maximum a priori parameters may be read off the  $\theta$  line of the `model_obs_display` below the status line. The algorithm estimates that the model from which the sample was drawn has a  $\mu$  of  $\sim 100.29$  and a precision of  $\sim .045$ , corresponding to a  $\sigma$  of  $\sim 4.7$ ; the global optimum has been found despite the poor starting prior.

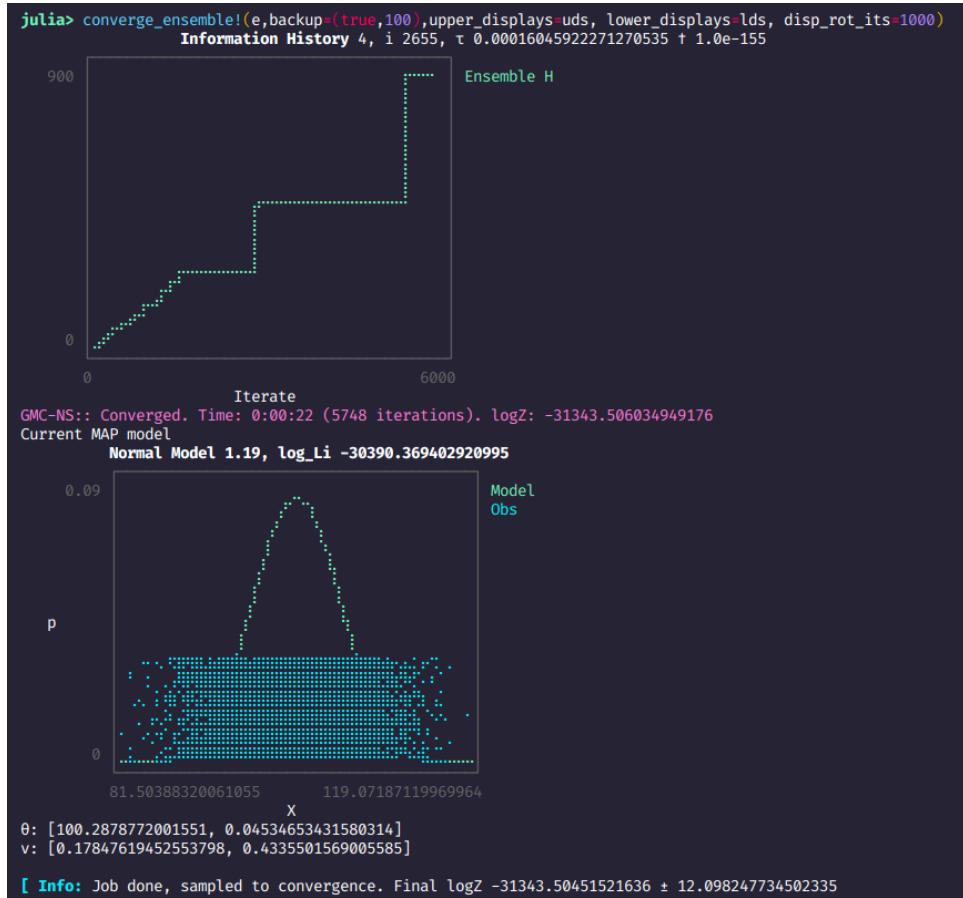


Figure 7.2: Example GMC-NS.jl output

## 7.4 Future Directions

`GMC_NS.jl` could be improved in at least four ways. Firstly, it ought to be trajectory- or sampling chain-centric rather than ensemble-centric, so that ensembles can be dynamically assembled in the manner suggested by Speagle [Spe19], which might fix the evidence accuracy problem this sampler exhibits. Briefly, this involves constructing an initial ensemble with a small number of trajectories or sampling chains, then adding to this ensemble by initializing new trajectories in the region of the prior volume that is most of interest to the user, prioritizing either evidence estimation by adding chains in the posterior mass, or posterior estimation by adding chains in the likelihood density. Secondly, an ellipsoidal search routine has been coded for `GMC_NS.jl`, but remains to be implemented; this could be

used both to initialize new chains in a dynamic sampling routine, and to estimate the size of posterior mode particles are currently in, for tuning purposes. Thirdly, the GMC PID tuning system likely needs improvement. The pI and pD constants are largely superfluous, and there may be a more fundamentally justified way to confer the information supplied by boundary contact to the particle by way of timestep tuning. Moreover, the minimum allowable timestep (`GMC_τ_death`) should be adjusted as the ensemble is built, such that it disallows small timesteps early in the sampling process. Lastly, the sampler should be refactored to use multiple different random number generation algorithms, not solely the standard Mersenne twister, which may cause problems with resonance under certain unusual circumstances [John Skilling, personal correspondence, 2020]. Beyond improvements to the implementation, `GMC_NS.jl` ought to be benchmarked against Dynesty and MultiNest to ensure that the sampler overhead is comparable or smaller to these implementations.

# Chapter 8

## CMZNicheSims.jl: *D. rerio* CMZ RPC niche simulators

### 8.1 Introduction

`CMZNicheSims.jl` (hereafter `CNS.jl`) comprises a set of three simulators of zebrafish retinal progenitor cells (RPCs), representing the postembryonic Circumferential Marginal Zone niche at varying levels of abstraction. These simulators were used in [Chapter 4](#) and [Chapter 5](#). This package extends `GMC_NS.jl`, described in [Chapter 7](#), and requires it. The simulators are organized around the nested sampling procedures implemented in that package. This means that simulations are executed as sample chains (“trajectories”) from the simulations’ parameter space in an ensemble. The basic operation of this package is accomplished by assembling an ensemble of models from the appropriate simulator submodule of `CNS.jl`, then elongating the trajectory of this ensemble by Galilean Monte Carlo using `GMC_NS.jl`’s `converge_ensemble` function. Note that `GMC_NS.jl`’s sampler settings are specified by fields of the `CNS.jl` model ensemble struct and not as arguments to `converge_ensemble`; even if default sampler settings are used, sampler operation should be understood before model priors are selected.

The three model ensemble types specified by `CNS.jl` are as follows:

- **CMZ\_Ensemble**: Phased difference function models of CMZ population and overall retinal volume, intended to model total retinal CMZ population and its contributions to overall retinal volume over long time periods (months)
- **Slice\_Ensemble**: Basic implementation of the “slice model” concept elaborated in [Chapter 3](#) for phased difference function CMZ models, uses a submodel of lens growth to determine “out-of-slice” growth contributions, intended to represent the population of a CMZ slice over long time periods (months)
- **Thymidine\_Ensemble**: “Slice model” of proliferative CMZ population exposed to a pulse of thymidine analogue, intended to model cell cycle kinetics in the CMZ over short time periods (hours).

The operation of these simulators is described in more detail below. In general, the workflow is to assemble the ensemble, converge it, and to estimate the model evidence and posterior parameters from the converged ensemble. For more on evidence and posterior estimation by `GMC_NS.jl`, see [Chapter 7](#).

## 8.2 CMZ\_Ensemble model and usage

A CMZ\_Ensemble consists of CMZ\_Models whose likelihood is assessed by Monte Carlo evaluation of a system of difference equations for simulated retinae, initialized from independent samples from the population and volume distributions specified in the ensemble constants vector. The update functions are specified in general by [Equation 4.1](#) and [Equation 4.2](#). However, the implementation of these functions in the model likelihood function (`CMZ_mc_11h`, found in [Section 17.4.3](#)) differs from this presentation; the equations are not evaluated stepwise day-by-day, but are rather updated for the next model event, either an observation timepoint or a phase transition date. The specific equations used in this function are of the computationally efficient factorial form suggested by Langtangen [[Lan12](#), p. 559], so that the equation for the CMZ population  $n$  days after some time 0 becomes:

$$p_n = p_0 \cdot (2^{\frac{24}{CT}} - \epsilon)^n$$

while the equation for retinal volume becomes:

$$v_n = v_0 + p_0 \cdot \epsilon \cdot \mu_{cv} \cdot \frac{1 - (2^{\frac{24}{CT}} - \epsilon)^{n-1}}{1 - (2^{\frac{24}{CT}} - \epsilon)}$$

, where  $p_n$  and  $v_n$  are the population and volume at day  $n$ ,  $p_0$  and  $v_0$  are the initial population and volumes, CT is the cycle time in hours,  $\epsilon$  is the exit rate from the niche (expressed as a fraction of the population), and  $\mu_{cv}$  is a constant expressing mean cellular volume contributed to the retina by exiting cells.

After Monte Carlo iterative simulation of many retinae, the resultant simulated population and volume distributions at the observed time points are estimated by fitting LogNormal Distributions, after which the joint log likelihood of the observations given the simulation is calculated. The likelihood function is threaded so that the total number of Monte Carlo iterates for any given model parameter vector is divided equally between threads. For very large numbers of iterates, this can result in some load balancing inefficiency; in general, it is unnecessary to specify more than 5 million iterates in the ensemble constants vector ([item 3](#) below). 1 million should be adequate for most purposes and allows the efficient sampling of  $\geq 3$  phase models on one performant machine.

A CMZ\_Ensemble may be assembled by specifying the path of the folder in which its constituent model files will be serialized, the starting number of chains in the ensemble (a few thousand may reasonably be expected to converge within a day or two on a single high performance machine), the observations to estimate the models against, a vector of prior distributions, a vector of constants, a box matrix, and the sampler settings vectors. The parameter vectors must follow these guidelines:

1. Observations vector: Must be a vector of vector tuples. The first member of the tuple is the population observation vector (total CMZ population at the vector's date index, see [3](#)), the second member is the date's total retinal volume observation vector (in cubicmicrometers). Ie. the observations vector takes the form of a `AbstractVector{<:Tuple{<:AbstractVector{<:Float64}, <:AbstractVector{<:Float64}}}`; the tuple of population and volume data found at index `x` corresponds from animals measured on date `constants_vec[1][x]`.
2. Prior vector: For each desired phase of the model, the prior vector must contain a distribution over that phase's mean population cycle time (in hours) and its mean exit rate ( $\geq 0.$ , may exceed 1.).

Paired cycle time and exit rate for each phase is followed by one less phase transition time in days than the number of phases. Ie. a two-phase prior vector might be, in pseudocode, [Phase1CT, Phase1ER, Phase2CT, Phase2ER, Phase1End].

3. Constants vector: A `Vector{<:Any}` containing, in order:
  - (a) T vector: Vector of integers or round floats, age in days of animals for each observations tuple
  - (b) Population distribution: `Distribution` defining the initial range of population values. Should normally be a fit from the initial population size observations
  - (c) Volume distribution: `Distribution` defining the initial range of retinal volume values. Should normally be a fit from the initial retinal volume observations
  - (d) Monte Carlo iterates: Integer specifying the number of retinae to be simulated at each point sampled from the parameter space. 1e6-5e6 is usually fine.
  - (e) Phases: Integer defining the number of phases specified by the prior vector

4. Box matrix: Two columns each of the same length as the prior, defining the minimum (column 1) and maximum (column 2) bounds for each parameter dimension to be sampled.

As an example, an ensemble with 3000 chains and using the default GMC sampler settings may be initialized and sampled as follows, where variable names correspond to the vectors described above:

---

```

1 e=CMZ_Ensemble(path, 3000, observations, prior, constants, box, GMC_DEFAULTS)
2 converge_ensemble!(e, backup=(true,50), mc_noise=.3)

```

---

The `mc_noise` parameter for `CNS.jl` models can be estimated by calculating the standard deviation of repeated evaluations of a parameter vector with a reasonably high likelihood, given the data vectors. The suggested value is roughly appropriate for 1e6 Monte Carlo iterates, given the data vectors used in Section 17.8.8.

### 8.3 Slice\_Ensemble model and usage

A `Slice_Ensemble` consists of `Slice_Models`, whose likelihood is assessed by `Monte Carlo` evaluation of a hybrid system of a difference equation and a power-law equation, representing a simulated CMZ RPC population in a retinal slice of arbitrary thickness. The `Slice_Model` likelihood function uses the same factorial solution to calculate the CMZ population after time steps of  $n$  days as that described for `CMZ_Models`. However, these population values are penalized by a value determined by using the power-law model of lens growth to determine the change in CMZ circumference over this time period, and the per-slice contribution required to expand the CMZ by this amount, given the present slice population and slice thickness (see Section 17.4.10 for this calculation).

`Slice_Models` are broadly similar to `CMZ_Models`. The parameter vectors required to assemble a `Slice_Ensemble` differ from those mentioned for a `CMZ_Ensemble`, however, and are summarised below.

1. Observations vector: Must be a vector of vector tuples. The first member of the tuple is the population observation vector (total CMZ population at the vector's date index, see 3), the second

member is the date's total retinal volume observation vector (in cubicmicrometers). Ie. the observations vector takes the form of a `AbstractVector{<:Tuple{<:AbstractVector{<:Float64}, <:AbstractVector{<:Float64}}}}`; the tuple of population and volume data found at index `x` corresponds from animals measured on date `constants_vec[1][x]`.

2. Prior vector: For each desired phase of the model, the prior vector must contain a distribution over that phase's mean population cycle time (in hours) and its mean exit rate ( $\geq 0.$ , may exceed 1.). Paired cycle time and exit rate for each phase is followed by one less phase transition time in days than the number of phases. Ie. a two-phase prior vector might be, in pseudocode, `[Phase1CT, Phase1ER, Phase2CT, Phase2ER, Phase1End]`.
3. Constants vector: A `Vector{<:Any}` containing, in order:
  - (a) T vector: Vector of integers or round floats, age in days of animals for each observations tuple
  - (b) Population distribution: `Distribution` defining the initial range of population values. Should normally be a fit from the initial population size observations
  - (c) Lens model: `Lens_Model` defining the lens circumferential growth power law and the slice's sectional thickness.
  - (d) Monte Carlo iterates: Integer specifying the number of retinae to be simulated at each point sampled from the parameter space. `1e6-5e6` is usually fine.
  - (e) Phases: Integer defining the number of phases specified by the prior vector
4. Box matrix: Two columns each of the same length as the prior, defining the minimum (column 1) and maximum (column 2) bounds for each parameter dimension to be sampled.

## 8.4 Decay\_Ensemble model and usage

`Decay_Models` are `Slice_Models` that are parameterised to represent a continuous process of cell cycle rate decay (that is, lengthening cycle times). They use identical `Lens_Models` to abstract the effects of the stretching 3-dimensional retina on the CMZ slice. Unlike the phased `CMZ_Models` and `Slice_Models`, `Decay_Models` are parameterised solely by an initial cycle time  $CT_i$ , an exponential decay constant  $\kappa$ , and a constant exit rate  $\epsilon$ . Daily cycle time is evaluated as  $CT_t = CT_i \cdot e^{\kappa t}$ . This is used to update the population value by the `Slice_Model` difference equation, penalized by the exit rate for the amount of CMZ growth implied by the `Lens_Model`.

A `Decay_Ensemble` is identical to a `Slice_Ensemble` except that it has only the three parameters in the order mentioned, and the corresponding prior vector, and omits the `Phases` constant.

## 8.5 MultiSlice\_Ensemble model and usage

A `MultiSlice_Model` consists of more than one slice observations set which are governed by identical parameter vectors. This allows independent observations from different areas of the CMZ to be collectively modelled by one `Slice_Model` or `Decay_Model` parameter vector. This can be used to represent a CMZ with morphological asymmetry but regulatory homogeneity, and is used as an alternative to separate models of morphologically distinguished slices in Chapter 4.

A `MultiSlice_Ensemble` is parameterised identically to the corresponding `Slice_Ensemble` or `Decay_Ensemble`, except that a vector of per-slice observations subvectors, and a vector of per-slice population distribution is provided. The `MultiSlice_Model` uses the paired observations and population distributions in vector order to build a vector of the appropriate slice model type, of the same length.

## 8.6 Thymidine\_Ensemble model and usage

Unlike the population-level models described above, the `Thymidine_Model` is a cell-based lineage simulator. It retains the slice model concept of comparing simulated observations directly to counts obtained from cryosections. Because these counts are obtained over a short time period (1 day), no effort is made to account for the growth of the eye. It uses an explicitly modelled cell cycle with G1, S, and G2M phases.

The model assumes a LogNormal distribution of overall cell cycle lengths. It is parameterised by  $TC_\mu$  and  $TC_{\sigma^2}$  values for this LogNormal distribution (a Normal Inverse Gamma prior should be used), a fraction  $s$  which gives the length of S-phase by multiplying into the sampled  $TC$ , a fraction  $g1$ , which gives the fraction of the remaining time which comes before S-phase as G1, and implies the length of the G2M phase afterwards. Finally, the fraction  $sis$  is randomly added to or subtracted from the cell cycle length determined for one sibling of a mitosis to give the cell cycle length of the other. This represents the correlated sibling cell cycle times which characterise *Danio* RPC populations, as usefully observed and modelled by He et al. [HZA<sup>+</sup>12]. The parameter vector is given in the order `[tcμ, tcσ², g1_frac, s_frac, sis_frac]`.

The model assumes linear labelling of nuclear contents over the cell cycle, and implements a minimum value for detection as a thymidine-labelled cell (`DETECTION_THRESHOLD`, set to .15 by default). This represents cells with a dusting of thymidine signal that are not counted as labelled, which normally occurs in confocal stack segmentation routines.

To simulate more than one subpopulation with independent cycle parameters, additional sets of the 5 parameters outlined above can be concatenated, with this vector appended by a parameter denoting the fraction of the population to which each additional parameterisation applies.

The `Thymidine_Ensemble` is constructed similarly to the other models, except that it requires a constants vector as follows:

Constants vector: A `Vector{<:Any}` containing, in order:

1. T vector: Vector of integers or round floats, chase time in hours from beginning of pulse for each observations subvector
2. Pulse: Float giving the length of the thymidine pulse in hours
3. Monte Carlo iterates: Integer specifying the number of retinae to be simulated at each point sampled from the parameter space. 1e5-5e5 usually gives sufficient density over the discrete distribution to avoid noise associated with simulated count values that were not observed at all.
4. End time: Float defining how long to run the simulation- should usually be the time of the last set of observations

Unlike the population-level models, which fit LogNormal distributions to samples of output collected by Monte Carlo execution, the `Thymidine_Model` constructs discrete nonparametric distributions of

simulated counts of labelled cells. This allows the model parameters to produce population distributions that diverge from smooth, continuous distributions. This is particularly notable in the banding pattern of some output, where some counts are very unlikely to be observed. This can allow the model to better reflect the prevalence of even-sized RPC lineages, for instance [HZA<sup>+</sup>12].

## Chapter 9

# BioBackgroundModels.jl: Parallel optimization of Hidden Markov Model zoos of genomic background noise

`BioBackgroundModels.jl` (`BBM.jl` hereafter) is a pure Julia package intended to automate the optimization and selection of large numbers (“zoos”) of Hidden Markov Models (HMMs), using sequence sampled from specified partitions of a given genome, on arbitrary collections of hardware. It is supplied with extensive utility functions for genomic sampling, high performance implementations of both the Baum-Welch and Churbanov-Winters algorithms for optimization of HMMs by expectation maximization (EM), as well as reporting functions for summarizing the results of the optimized model zoos. These tasks are necessary to select of models of genomic background noise, from which motif signals may be detected using a package such as `BioMotifInference.jl`.

HMMs generated by `BBM.jl` are standard  $\geq 1$  state HMMs that emit symbols corresponding to stretches of 1 or more base pairs. Following Down et al. [DH05], the number of bases encoded by an HMM symbol is denoted by the “order” of the HMM. 0th order HMM emit 4 symbols, one for each of the genomic nucleotides, while 1<sup>st</sup> order HMMs emit 16 symbols, one for each 2-base combination, and 2<sup>nd</sup> order HMMs emit 64 symbols, one for each 3-base combination, and so on. It is expected that users will be interested in comparing the explanatory value of a variety of background HMM state numbers and orders in selecting appropriate background models. These can all be optimized in one batch; because high state and, to a lesser extent, high order numbers impose much more significant memory costs, the use of the Churbanov-Winters linear memory algorithm is strongly recommended<sup>1</sup>. A basic load balancing system has been provided to allow users to prefer particular machines for some optimization jobs over others, which permits the efficient use of clusters of dissimilar hardware in the batch context.

---

<sup>1</sup>While the theoretical performance of the Churbanov-Winters algorithm is lower than Baum-Welch [CW08], the implementation herein is highly optimized and performant. For many use cases, it is both less memory-intensive and faster than Baum-Welch.

## 9.1 Genome partitioning and sampling

The optimization of an HMM zoo is performed against a sample of a “genome partition”, which defines a portion of a genome in relation to a genomic feature set. The required files to sample from a genome partition are:

- A whole-genome sequence in valid FASTA format (eg. a .fna or other appropriate nucleotide sequence file).
- An index for the genomic sequence file (eg. an .fna.fai file)
- A genomic feature file, annotating the genomic sequence, in GFF3 format

At present, BBM.jl offers a default strategy of sampling without replacement from exonic, periexonic, and intergenic partitions of genomes. This partition scheme is intended to differentiate between genomic sequences that are likely to differ in terms of the appropriateness of HMMs of different orders as background models. That is, the selective pressure imposed on genomic sequence by the codon code (exonic partition) and other higher-order functional features of gene organization (periexonic partition) are likely to favour different sorts of background HMM model than the simpler repetitive structure of intergenic material.

Sampling functions should normally be accessed through the API provided to the user. Firstly, sampling jobs may be set up by using the `setup_sample_jobs` function, which uses user-supplied paths to the files listed above, as well as user-specified partitioning variables, to divide the supplied genome into the partitions described and set up channels for these sampling jobs. For example:

---

```

1 using BioBackgroundModels
2 #FILE PATHS
3 danio_gff_path = "Danio_rerio.GRCz11.94.gff3"
4 danio_genome_path = "GCA_000002035.4_GRCz11_genomic.fna"
5 danio_gen_index_path = "GCA_000002035.4_GRCz11_genomic.fna.fai"
6
7 #CONSTANTS FOR GENOMIC SAMPLING
8 const sample_set_length = Int64(4e6)
9 const sample_window_min = 10
10 const sample_window_max = 3000
11 const perigenic_pad = 500
12
13 channels = setup_sample_jobs(danio_genome_path, danio_gen_index_path, danio_gff_path,
→ sample_set_length, sample_window_min, sample_window_max, perigenic_pad)

```

---

Here, `sample_set_length` defines the overall number of bases to be sampled from each partition, while `sample_window_min` and `sample_window_max` define the minimum and maximum length of each individual sample to be taken, without replacement, from the partitions. `sample_set_length` should be chosen to supply both training and test sets for background model selection; ie. it should be twice the desired length of the observation set in training. Lastly, `perigenic_pad` defines the number of bases up-

and down-stream of the first and last codon to consider, along with intronic material, as constituting the “perigenic” partition. This will include promoter elements, splicing signals, and the like.

Secondly, the partitions are sampled (in parallel, by worker, if desired), to produce dataframes of samples and relevant coordinate and strand orientation information:

---

```

1 using Distributed,Serialization
2 sample_output = "samples"
3 worker_pool = addprocs(3)
4 sample_record_dfs=execute_sample_jobs(channels, worker_pool)
5 serialize(sample_output, sample_record_dfs)

```

---

Here, one worker per partition is used to sample; lower numbers of workers will also work while larger numbers are of no benefit. In this example, the sample dataframes are serialized to the file “samples” for later use; they can also be used directly as described below.

## 9.2 Optimizing BHMMs by EM algorithms

Background HMM optimization by the expectation-maximization algorithms is performed by a similar two-step setup-and-execute functional pattern as genome sampling. Firstly, `setup_EM_jobs!` is used to transform supplied “job IDs” and the observations supplied by genome sampling into sets of EM chains to be elongated by the EM algorithm. `BBM.jl` “job IDs” take the form of `Chain_ID` structs, whose fields specify a string denoting the partition that forms the observations for the optimization, the number of states in the HMM, the order of the HMM’s emitted symbols, and an integer denoting the replicate number of the chain. In general, for any given partition, number of states, and emission order, three or more replicates are advised, to confirm that the converged values of any particular HMM are likely to be near the global optimum [YBW15]. That is, if replicates of some job ID chain all terminate with similar parameter values (see Section 9.3 to examine this using `BBM.jl`), one such chain may reasonably be selected as a representative of the likely global optimum present in this region of the parameter space.

The helper function `split_obs_sets` is used to divide the genomic sample dataframes generated by the sampling API, described above, into training and test sets for optimization. The training set is supplied for the use of the EM algorithms, while the test set is used solely to calculate the final likelihoods of optimized models.

By default, `BBM.jl` initialises new HMM chains by sampling randomly from possible emission vectors, with strong priors on transition matrices with robust autotransition. Generally speaking, this tends to prevent chains from being optimized to trivial local minima with very short state residency times. This reflects our exclusive interest in those HMMs whose states represent stretches of nucleotides with shared biological significance; HMMs which tend to have state residency emission lengths of less than a few bases are of questionable biological relevance. If desired, the user may supply their own initialisation function as the `setup_EM_jobs!` keyword argument `init_function`, which defaults to `autotransition_init`. The default function should serve as the template for such user-defined initialisation functions.

`BBM.jl` operation is designed to be robust to most kinds of interruption, and the `setup_EM_jobs!` function may be used safely on existing sets of EM chains generated by `BBM`. This allows for resumption

from interruption, reconfiguration of hardware, decreasing the desired chain termination threshold, and so on. The dict<sup>2</sup> containing existing serialized chains, if any, should be supplied to the setup function as the `chains` keyword variable to allow this.

---

```

1 using BioBackgroundModels, DataFrames, Distributed, Serialization
2
3 #FILE PATHS
4 hmm_output = "chains"
5 sample_output = "samples"
6
7 #JOB CONSTANTS
8 const replicates = 3 #repeat optimisation from this many seperately initialised
   → samples from the prior
9 const Ks = [1,2,4,6] #state #s to test
10 const order_nos = [0,1,2] #DNA kmer order #s to test
11 const delta_thresh=1e-5 #stopping/convergence criterion (log probability difference
   → btw subsequent EM iterates)
12 const max_iterates=50000
13
14 #PROGRAMATICALLY GENERATE Chain_ID Vector
15 job_ids=Vector{Chain_ID}()
16 for (obs_id, obs) in training_sets, K in Ks, order in order_nos, rep in 1:replicates
17     push!(job_ids, Chain_ID(obs_id, K, order, rep))
18 end
19
20 #SPLIT GENOME SAMPLES INTO TRAINING AND TEST SETS
21 sample_dfs = deserialize(sample_output)
22 training_sets, test_sets = split_obs_sets(sample_dfs)
23
24 #INITIALIZE HMMS
25 if isfile(hmm_output) #if some results have already been collected, load them
26     hmm_results_dict = deserialize(hmm_output)
27 else #otherwise, pass a new results dict
28     hmm_results_dict = Dict{Chain_ID,Vector{EM_step}}()
29 end
30
31 em_jobset = setup_EM_jobs!(job_ids, training_sets; delta_thresh=delta_thresh,
   → chains=hmm_results_dict)
```

---

Once the EM jobset is set up, it is splatted<sup>3</sup> into the `execute_EM_jobs!` function with any valid worker pool and the path for the Dict of EM chains to be serialized. All workers in the pool should be in

<sup>2</sup>A "Dict" in Julia is a dictionary of key-value pairs.

<sup>3</sup>"Splattting" in Julia refers to the expansion of a argument vector into a function's call by using the ... operator following the vector's name, i.e. `function(arguments...)`.

`:master_worker` topology with the master process. BBM.jl workers will obtain new chains to elongate by EM until no more are available; the number of jobs, therefore, defines the useful upper limit for the number of workers in the pool.

Load balancing is achieved by using BBM.jl's LoadConfig struct, whose fields define acceptable ranges of states, orders, and optional vectors of blacklisted and whitelisted `Chain_IDs`. A Dict of LoadConfigs keyed by Integer defines the appropriate LoadConfig to use for any given worker's integer id (ie. the output of `Distributed.myid()`).

The results of EM optimizations are serialized as they are “pulled off the wire”, so that they cannot be lost by the algorithm being interrupted, network failure etc. Consequently, `execute_EM_jobs!` does not return anything that needs to be serialized for subsequent analyses; results are always located at the path provided to the function and may be freely inspected by the tools described in Section 9.3 at any time during or after the optimization.

An example of `execute_EM_jobs!` usage, following from the setup example given above:

---

```

1 #SETUP LOAD BALANCING
2 local_config=LoadConfig(1:6,0:2)
3 remote_config=LoadConfig(1:4,0:1)
4
5 load_dict=Dict{Int64,LoadConfig}()
6
7 worker_pool=addprocs(no_local_processes, topology=:master_worker)
8 for worker in worker_pool
9     load_dict[worker]=local_config
10 end
11
12 remote_pool=addprocs([(remote_machine,no_remote_processes)], tunnel=true,
13     ↳ topology=:master_worker)
14
15 for worker in remote_pool
16     load_dict[worker]=remote_config
17 end
18
19 worker_pool=vcat(worker_pool, remote_pool)
20 #EXECUTE EM JOBS
21 execute_EM_jobs!(worker_pool, em_jobset..., hmm_output; load_dict=load_dict,
22     ↳ delta_thresh=delta_thresh, max_iterates=max_iterates)

```

---

### 9.3 BHMM analysis and display

A report generation and display API is provided in order to analyse the results of optimizing a zoo of background HMMs using BBM.jl. One function, `generate_reports`, serves to prepare all of the

available reports on the zoo. Its use is straightforward:

---

```

1 chains=deserialize(hmm_output)
2 sample_dfs = deserialize(sample_output)
3 training_sets, test_sets = split_obs_sets(sample_dfs)
4
5 report_folders=generate_reports(chains, test_sets)
6 serialize(survey_folders, report_folders) #save reports

```

---

`generate_reports` creates a Dict of `Report_Folders`, keyed by partition id string. Each `Report_Folder` contains a `Partition_Report` with information about all HMMs trained on that partition, a `Replicate_Report` which enables the comparison of the replicates of the best model available for the partition, and a Dict of `Chain_Reports` keyed by `Chain_ID`, which give specific information concerning each chain produced for the partition. Examples of report output follow, derived from the model zoo training task whose results are summarised [Figure 13.10](#). The contents of the reports on background models trained on the *D. rerio* “exonic” partition follow.

[Figure 9.1](#) displays partial output from the `Partition_Report` associated with BHMM training on the exonic partition. The report identifies the partition in question, and, for each order of BHMM trained on the partition, displays a plot of the likelihood of the models vs. the number of states in the model. In this case, 0<sup>th</sup> and 2<sup>nd</sup> order models are displayed. The likelihood of a naive, 0<sup>th</sup>-order, 1-state model with equal emission probability for each symbol serves as a benchmark for BHMM likelihood; this value is plotted as a magenta line across the state axis. In this case, all 0<sup>th</sup>-order models are more likely descriptions of *D. rerio* exonic material than a naive model, while all 2<sup>nd</sup>-order models fall below this benchmark likelihood. This reflects the greatly expanded parameter space associated with 64 emission symbols. Thus, the additional exonic order capturable by 3mers is insufficient to offset the decreased probability of any given sequence using this many symbols.

Individual models are plotted as scatterplot points. Converged models are rendered in green, unconverged models in red<sup>4</sup>. In this case, no unconverged models are present. The 0<sup>th</sup>-order models are tightly converged such that their likelihoods are not differentiable, while the 2<sup>nd</sup>-order models are more spread out across the larger parameter space. The intended use of this report is to perform a quick visual assessment of the state of the zoo trained on the specified partition, before proceeding to the `Replicate_Report` and particular `Chain_Reports`.

[Figure 9.2](#) displays partial output from the exonic `Replicate_Report`. The chains displayed in the report are the most likely replicate set for the partition. The report consists of paired convergence plots for the autotransition probabilities and 2-dimensional symbol emission probabilities for each state and each replicate in the set. Because there is no guarantee that HMMs from the same optima will have the same ordering of states, similar states are identified by minimizing the Euclidean distance between state emission vectors across the trained replicates. In this case, replicate 1’s second state (red) is also identified with the second state of replicate 2 (green), but the sixth state of replicate 3 (blue).

The autotransition probability subplot displays the transition matrix diagonal probability for the designated state and replicates (that is, the probability that the HMM will stay in the state). If the

---

<sup>4</sup>The reports API is agnostic about the state of the model zoo reported on; it can be used to monitor zoos before they are fully converged.

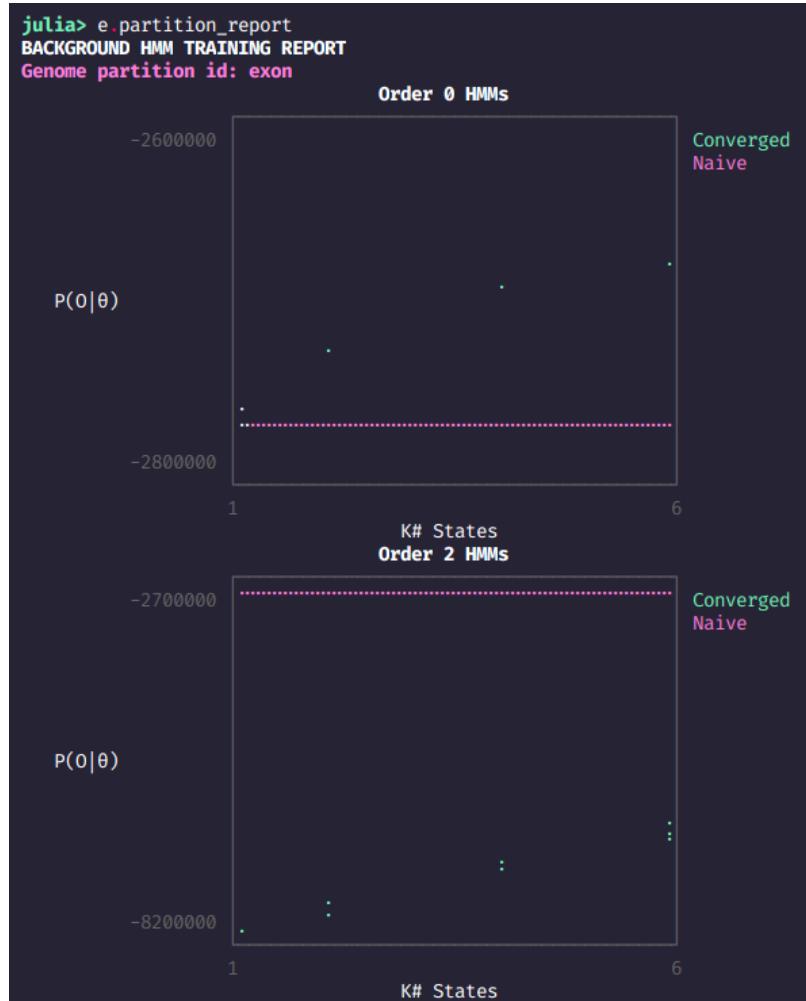


Figure 9.1: Example BBM.jl Partition Report

HMMs are correctly converged on a global optimum, the autotransition probabilities for all chains should terminate at nearly the same value. Given that this probability will be stationary on converged chains, this can be visually confirmed by looking for substantial horizontal overlap of the chain autotransition probabilities for the state, as seen in Figure 9.2.

The symbol probability subplot displays the evolution of the probability of the first symbol emitted by the state plotted against the probability of the second symbol, providing a simple two-dimensional view into the convergence of the state's emission vector. If the chains are correctly converged on the global optimum, the plot's appearance should show chains from divergent areas of this two-dimensional parameter subset converging on the same values for the two symbols, as seen in Figure 9.2.

`Chain_Reports` for all HMMs trained on the partition are available in a Dict keyed by `Chain_ID`, found at the `chain_reports` field of the report folder. Example output for replicate 1 from the above-described replicate set is presented below. `Chain_Reports` consist of three major sections, described in turn.

The first `Chain_Report` section begins by stating the state number and order of the BHMM, as well

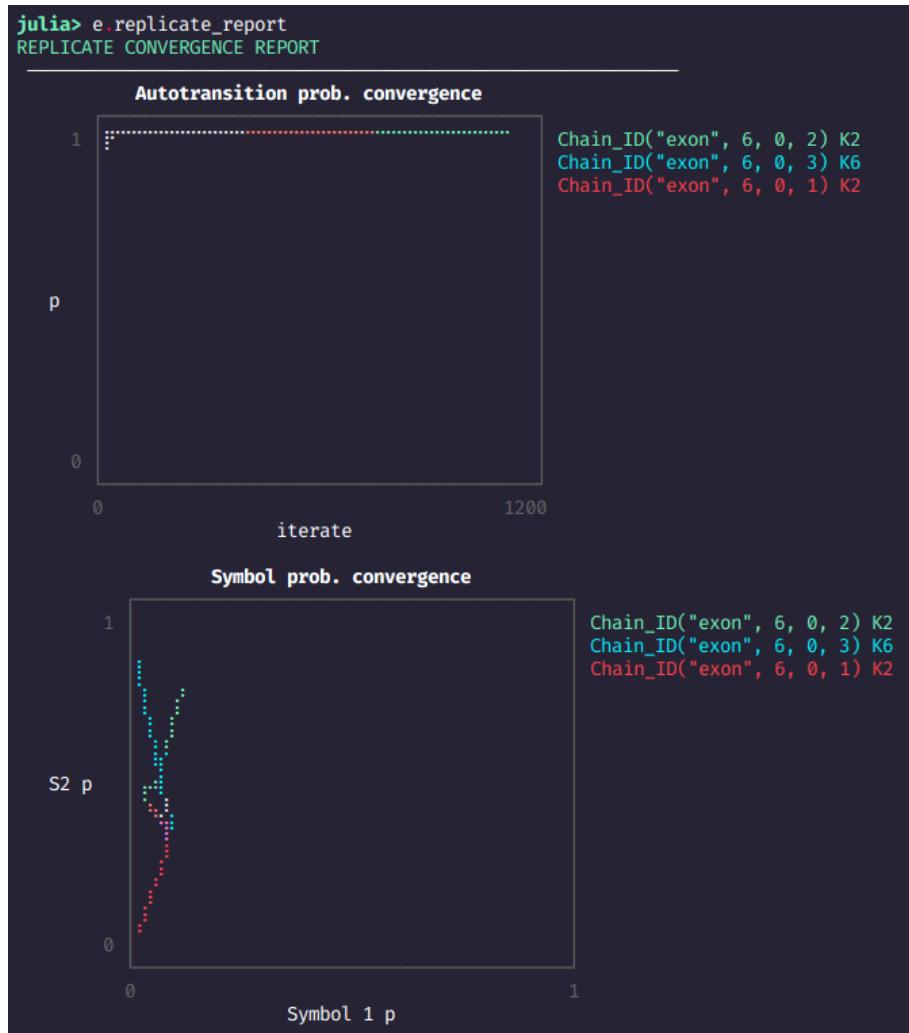


Figure 9.2: Example BBM.jl Replicate Report

as the genomic partition on which it was trained. This is followed by the likelihood of the model given the test observation set, compared to the naive model. The line stating these likelihoods will be green if the BHMM is better than the naive model, red if not. The general information concludes with the replicate number of the BHMM, as well as the convergence status of the chain and the final EM step size.

The section continues by stating the initial state probability vector  $a$  and the state transition matrix  $A$  for the converged BHMM. Emission vectors for the BHMMs' state are summarized by their informational content; symbols with  $\geq .7$  bits of information are printed in bold capital characters, those with  $.7$  information  $\geq .25$  are printed in capital characters, those with  $.25$  information  $\geq .05$  are printed with lowercase characters, and those with less than  $.05$  bits are not printed at all. In this case, state 2 emits adenine and cytosine symbols more frequently than guanine or thymidine, while this is reversed in state 4. The other states have similar enough emission probabilities for the four bases that they do not register on the informative symbols display. This gives a rough idea of what sort of information the BHMM

```
julia> e.chain_reports[Chain_ID("exon",6,0,1)]
Base.TTY(RawFD(0x000000017) open, 0 bytes waiting)BHMM EM Chain Results
6-state, 0th order BHMM
Trained on observation set "exon"
Test logP(0|θ): -2.677474495850883e6, greater than the naive model's -2.7733248445454747e6
Replicate 1
Converged with final step δ 0.0009887791238725185
Last θ:
Background BHMM
State Initial and Transition Probabilities
a: [0.31786232049327706, 0.08311610163606636, 0.14481102538589713, 0.1087121512373124, 0.09122496723393655, 0.2542734340135114]
A: 6×6 Array{Float64,2}:
 0.776466   0.00134197   0.000597762   9.3932e-17   1.12908e-29   0.221594
 4.9891e-18  0.993806   0.000114231   0.00035994   0.000666623   0.00505363
 0.00501082  2.53527e-23  0.993775   0.00119238   2.16171e-5   1.21753e-8
 4.75045e-22  1.85006e-24  9.91045e-118  0.979917   0.000395467  0.0196875
 7.04204e-5   0.000262805  0.00112526   0.000411873  0.996586   0.00154403
 0.174564    0.000300059  0.000513614  0.00213432   0.00127718   0.82121
INFORMATIVE SYMBOLS BY STATE
K1 <<< >>>
K2 <<< a c >>>
K3 <<< >>>
K4 <<< g t >>>
K5 <<< >>>
K6 <<< >>>
State Mean Feature Length (bp)
K1: 4.647
K2: 166.237
K3: 148.03
K4: 49.701
K5: 300.917
K6: 5.396
```

Figure 9.3: Example BBM.jl Chain Report - Part 1

states encode.

The last part of this section presents an estimate of the mean feature length for each state, meaning the mean number of bases emitted from the state before transitioning to a new state (otherwise known as “state residency time” and similar concepts). This is estimated by Monte Carlo sampling of run lengths, given the autotransition probabilities for the BHMM. This is intended to give an idea of the biological relevance of the sequence features emitted by the BHMM’s states. Generally speaking, most “background noise” features, like intergenic CpG islands, are held to be on the order of hundreds of bases. Users should expect good BHMMs to have some states that tend to emit features around this size.

The second part of the `Chain_Report`, displayed in Figure 9.4, displays the evolution of the model’s likelihood and state autotransition probabilities over the chain iterates. This is primarily intended as a visual aid to the convergence tests presented in the third part of the report. Both the model likelihood and autotransition probabilities should be stationary over the last portion of the chain for it to be considered legitimately converged. This can be numerically confirmed in the third part of the report.

The third part of the `Chain_Report` displays the output of the Heidelberger and Welch convergence diagnostic [HW81], as implemented by `MCMCChains.jl` [Pf20]. Both the BHMM likelihood and the autotransition probability for each state are tested for stationarity. A chain from a good replicate set (as assessed by the `Replicate_Report`) that passes these diagnostic tests is reasonably considered properly converged on the global optimum for the partition samples used as the training set. If this is so, a green test-passing statement will be printed, if not, a red failure statement. The specific reason for failure

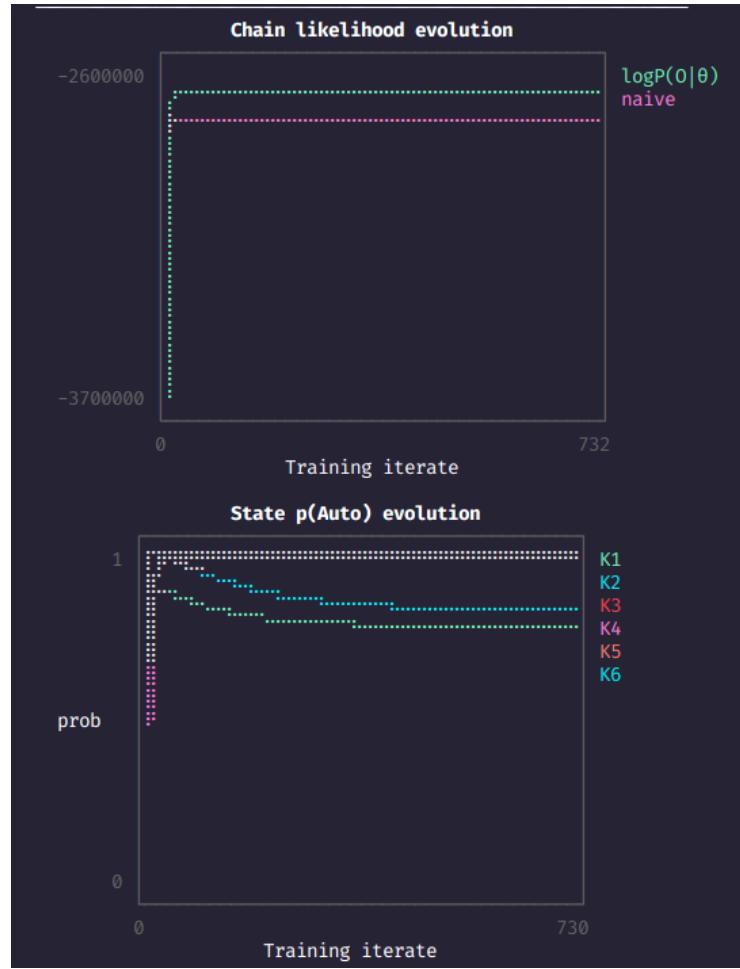


Figure 9.4: Example BBM.jl Chain Report - Part 2

Convergence Diagnostics						
Likelihood and autotransition probabilities converged and passing tests.						
Heidelberger and Welch Diagnostic - Chain 1						
parameters	burnin	stationarity	pvalue	mean	halfwidth	test
Symbol	Float64	Float64	Float64	Float64	Float64	Float64
$\log P(O \theta)$	0.0000	1.0000	0.9391	-2686293.3768	6528.6612	1.0000
K1	0.0000	1.0000	0.2135	0.7959	0.0262	1.0000
K2	0.0000	1.0000	0.7116	0.9998	0.0002	1.0000
K3	0.0000	1.0000	0.2333	0.9993	0.0006	1.0000
K4	0.0000	1.0000	0.7631	0.9712	0.0161	1.0000
K5	0.0000	1.0000	0.1643	0.9966	0.0000	1.0000
K6	0.0000	1.0000	0.1011	0.8615	0.0501	1.0000

Figure 9.5: Example BBM.jl Chain Report - Part 3

can be determined by finding the parameters with a test value of 0. Taken together, these reports can be used to verify that the EM optimization is functioning properly, and that appropriately converged models are chosen to represent genomic background noise from the partition of interest.

## 9.4 Numerical accuracy and tests

Because `BBM.jl` contains two different `EM` algorithms, these are used internally to check for numerical accuracy and consistency. In general, input to either Baum-Welch or Churbanov algorithms should produce nearly-identical output (i.e. `isapprox()` will return true). Additionally, some lower-performance implementations are available under `BBM.jl/src/test/ref_fns.jl` for comparison. A fairly comprehensive test suite is provided, and can be executed using `]testBioBackgroundModels`. So long as tests are passing, numerical accuracy of the EM algorithms is established. The tests make use of a synthetic FASTA-format genome to test the genome sampling routines. Passing tests indicates that this system is working as intended.

## 9.5 Future directions

`BBM.jl` is largely feature complete. However, the genomic sampling regime implemented for this thesis is unnecessarily restrictive. In particular, there is no provision for common tasks like assembling the upstream promoter sequences of some group of genes with a common functional annotation. This could be improved by implementing a partition definition system that uses user-defined feature- and coordinate-based masking of the genome to assemble custom partitions. This would make `BBM.jl` a useful generalist tool for establishing background models when inferring genomic signals. Finally, the Churbanov and Baum-Welch algorithms should be benchmarked against other multi-observation implementations.

# Chapter 10

## BioMotifInference.jl: Independent component analysis motif inference by nested sampling

### 10.1 Introduction

While many methods to detect overrepresented motifs in nucleotide sequences, only one is a rigorously validated system of statistical inference that allows us to calculate the evidence for these motif models, given genomic observations: nested sampling [Ski06]. Computational biologists almost immediately benefitted from Skilling's original publication, with the release of the Java-coded nMICA by Down et al. in 2005 [DH05]. Unfortunately, this code base is no longer being maintained. It is, moreover, desireable to take advantage of modern languages and programming techniques to improve the maintainability and productivity of important bioinformatic code. We therefore re-implemented Skilling's original algorithm, for use in inference of DNA motifs in Julia [BEKS15], as `BioMotifInference.jl`, hereafter `BMI.jl`.

In brief, `BMI.jl` is intended to estimate the evidence for Independent Component Analysis models of Position Weight Matrix (PWM) signals on Hidden Markov Models of genomic background noise, and to estimate the maximum *a priori* parameters of detected signals. This allows the user to perform Bayesian model selection and comparison for generative hypotheses about genomic sequences of interest. The ICA approach is discussed in more detail in the relevant sections of Chapter 15.

### 10.2 Implementation of the nested sampling algorithm

Skilling's nested sampling algorithm is accompanied with an almost-certain guarantee to converge an ensemble of models on the global optimum likelihood in the parameter space [Ski06]. An important assumption underpinning this guarantee is that the sampling density (in effect, the size of the ensemble) be high enough that widely separated modes are populated by at least one model each. Although early suggestions for the generation of new models within the ensemble were generally to decorrelate existing models by diffusional Monte Carlo permutation, later work generally acknowledges that this can be highly inefficient [Ski12], particularly in large parameter spaces with many modes, of which the sequence

parameter spaces explored in Chapter 5 are good examples. A number of ways of addressing this problem have arisen in the cosmology and physics literature. These include improved sampling methods, such as sampling within an ellipsoidal hypersphere encompassing the positions of the ensemble models within the parameter space [FH08, FHB09] or Galilean Monte Carlo (GMC) [Ski12], as well as methods of increasing computational efficiency, such as dynamic adjustment of ensemble size [HHHL19].

Generally speaking, these methods are not good candidates for `BMI.jl` because the PWM representation of sequence signal in the ICA PWM model (IPM) is makes ordinary parameter space representation much less efficient. This is for at least four reasons:

1. Identical IPMs may be expressed with their vector of PWM sources (or "channels") in different orders, so the parameter space becomes increasingly degenerate with higher numbers of sources.
2. Closely equivalent repetitive signals can be represented by PWMs on opposite ends of the parameter space (e.g. ATAT vs TATA), reducing the efficiency of ellipsoidal sampling methods and introducing a further degeneracy.
3. If model likelihoods are being calculated on the reverse strand of observations, sources on opposite ends of the parameter space (ie. reverse complements) can also represent closely equivalent signals, introducing another source of degeneracy.
4. IPM sources may be of different lengths. Even if we can relate sources in different models to one another by some consistent distance rule, it is unclear how one would decide which dimensions of longer sources to project the parameters of shorter ones into. BioMotifInference does maintain an index for each source against its prior, but this is no guarantee that sources remain in some way "aligned". Rather, some type of alignment would have to be done, probably significantly increasing computational cost<sup>1</sup>.

While it is possible to perform Markov Chain Monte Carlo in a manner that allows jumps between differently-dimensioned models in detailed balance, this applies only to diffeomorphic transformations [HG12]. PWM models are not merely matrices of unrelated parameters, but are an ordered sequence of 4-parameter discrete Categorical distributions. If the addition or removal of a discrete, indexed set of 4 parameters to an existing vector of such sets can be made reversible in this manner, the proof is beyond the scope of the present work. Therefore, BioMotifInference (BMI) is not likely in strict detailed balance over a well behaved parameter space, given its default permute functions and configuration. Permute functions are designed around pragmatic concerns associated with the flow of model information in the IPM ensemble, and are tuned for computational efficiency in producing permuted models that are more likely than the starting model. The justification for this is simply feasibility; doing this allows an ensemble to be converged on larger samples from genome-scale datasets. Because of the flexible nature of the sampler, the user may easily specify `BMI.jl`-compatible permute functions, should better ones for their application be apparent to them.

It should be noted that these considerations do not entirely preclude the operation of the `BMI.jl` in detailed balance over the prior. This can be achieved by the user specifying a single length for the PWM sources (rather than a range of permissible lengths), and using a permutation routine consisting solely of the functions `random_decorrelate` and `reinit_src`, along with any user-defined functions coded with

---

<sup>1</sup>Probably some combination of dimensional analysis and fast alignment algorithms could make some headway here.

the detailed balance stricture in mind. Operation in this fashion may improve the accuracy of both evidence estimates and the sources found in the maximum a posteriori (MAP) sample defined by the converged ensemble. That said, unless the supplied prior is unusually good, this mode of operation is only well suited to very small observation sets, on the order of kilobases. With uninformative priors and good background models, the vast majority of sources sampled from the prior will receive significant likelihood penalties relative to the background model alone. Performing nested sampling on such an ensemble by diffusional techniques, in good detailed balance, will usually fail to produce models more likely than the background model within any reasonable compute budget, and will generally never converge. The feasibility constraint thus justifies the use of bespoke sampling routines for genome-scale datasets.

Like its predecessor nMICA, BMI.jl samples new models by applying one of a collection of permutation functions to an existing model. Additionally, although BMI.jl is supplied prior distributions on IPM parameters from which the initial ensemble is sampled, there is no way provided to calculate a posterior from the models generated by the nested sampling process, largely because of the identical source issue, 1. The primary outputs of interest are the estimation of the Bayesian evidence ratios between models (the absolute values of evidence are unlikely to be accurate)<sup>2</sup>, as well as the models in the final ensemble (which constitute samples from the MAP mode or modes).

Because the more conventional methods to address the inefficiency of nested sampling by Monte Carlo used in cosmological models described above are unavailable, BMI.jl uses an ad hoc method of adjusting the mixture of permute patterns that are applied to models. The basic permute logic (encoded by `Permute_Instruct` arguments to the `permute_IPM` function) is as follows:

1. Select a random model from the ensemble.
2. Apply a randomly selected permutation function from the instruction's function list according to the probability weights given to the functions by the instruction's weight vector.
3. Repeat 2 until a model more likely than the ensemble contour, given observations, is found, or the instruction's `func_limit` is reached.
4. If the `func_limit` is reached without a new model being found, return to 1 and repeat 2 until a model more likely than the ensemble's contour is found, or until the instruction's `model_limit` is reached.

## 10.3 Usage notes

### 10.3.1 Preparation of observation set

BMI.jl relies on `BioBackgroundModels.jl` both for selecting appropriate background models for motif inference, as well as to code observations and prepare an appropriate matrix of background scores. The selection of background models is covered in Chapter 9. The coding of observations requires the construction of a `DataFrame` containing a column of type `Vector{LongSequence{DNAAlphabet{2}}}`, the unambiguous DNA alphabet of `BioSequences.jl`. This is accomplished as follows:

---

<sup>2</sup>It is not clear that evidence ratios solve the accuracy problem [Kevin Knuth, personal correspondence, 2021]. The best that can be said about this is that a larger ratio in favour of some model is more likely to reflect an actual preponderence of evidence in its favour. For this reason, the MAP sample may be the most informative part of BMI output.

---

```

1  using BioBackgroundModels, DataFrames, Serialization
2  obspath=/path/to/observations_dataframe
3  obs_df=deserialize(obspath)

```

---

Note that `observation_setup` must be supplied an `order` keyword argument of 0; while `BMI.jl` supports background HMMs of any order, signal motifs are 0<sup>th</sup> order. If the name of the sequence column of the observations `DataFrame` differs from “Seq”, a symbol specifying the column name should be passed as the `symbol` keyword argument.

To construct the background matrix, the observations must be masked by partition, and the positional likelihoods of the sequences calculated:

---

```

1  BioBackgroundModels.add_partition_masks!(obs_df, genome_gff_path, 500,
   →  (:chr,:seq,:start))
2
3  BHMM_dict = Dict{String,BHMM}()
4  reports_folder=deserialize(/path/to/BBM_Reports_Folder)
5  for (partition, folder) in reports_folder
6      BHMM_dict[partition]=folder.partition_report.best_model[2]
7  end

```

---

`add_partition_masks!` adds a mask column to the observations `DataFrame`, which is used by `BGHMM_likelihood_calc` to determine the correct background model to use for any given base in the observations sequence. It requires a path to a valid GFF3 annotation file for the genome. In order for the observations to be masked correctly, a tuple of column names containing the scaffold ID the observation is found on<sup>3</sup>, the sequence on the positive strand, and the start position of the sequence on the scaffold.

The calculation of background scores requires a `Dict` of `BBM.jl` background models keyed by the partition ID. In the example, this is constructed programmatically from a `Report_Folder`; see Chapter 9 for the generation of these reports.

### 10.3.2 IPM\_Ensemble assembly

Once the coded observation set and the matrix of background scores are prepared, an `IPM_Ensemble` may be initialised by sampling from a prior. Priors must first be assembled. This process is executed as follows:

---

```

1  num_sources=5
2  source_length_range=3:10
3  uninformative_mixing_prior=.1
4  source_prior = BioMotifInference.assemble_source_priors(num_sources,
   →  Vector{Matrix{Float64}}())

```

---

<sup>3</sup>It must be ensured that this ID corresponds to the one used by the GFF3

---

```

5     mix_prior = (falses(0,0), uninformative_mixing_prior)
6
7     e = BioMotifInference.IPM_Ensemble(worker_pool, "/path/to/ensemble/",
→   ensemble_size, source_prior, (falses(0,0), mixing_prior), background_scores,
→   coded_obs, source_length_range, posterior_switch=true)

```

---

Here, an uninformative prior is assembled for 5 sources. If we wanted to supply some PWM samples create informative Dirichlet priors from, these would be supplied in the vector of matrices passed as the second positional argument to `assemble_source_priors`. Informative priors are produced by multiplying the PWM values by a fixed weight, which may be passed as the keyword argument `wt`.

An uninformative prior is supplied for the mixing matrix. The prior tuple consists of an empty `BitMatrix`; if an informative prior is desired, the appropriate `BitMatrix` for the informative sources is passed instead. Sources for which no informative prior is supplied have their mix vectors sampled from a fractional floating point success rate prior. Here, the `uninformative_mixing_prior` is .1, meaning that an average of 10% of observations will be initialized with each source.

Sampling and likelihood calculations may be performed in parallel by submitting a valid integer vector worker pool as the first positional argument of `IPM_Ensemble`, or conducted by the master process if this is omitted. Passing `true` for the keyword argument `posterior_switch` will ensure that the least likely models, discarded from the ensemble during the sampling procedure, are collected as samples from the posterior distribution. This is `false` by default, as no rigorous method for estimating posterior distributions is available.

### 10.3.3 Permute routine setup

In order to converge an assembled `IPM_Ensemble` by nested sampling, it is necessary to provide an initial `Permute_Instruct`, which determines sampler behaviour, and is tuned by the `Permute_Tuner` to prioritize permute functions which currently produce the largest positive particle displacements over the likelihood surface in parameter space, per computational time. Some application-specific tweaking is usually necessary. Depending on how the computation is to be parallelized, the weights assigned to network-intensive functions by the tuner, in particular, may benefit from being clamped. In order to convey how to set up an initial `Permute_Instruct` successfully, notes regarding the operation of the stock permute functions on `ICA_PWM_Models` (IPMs) is summarized below. In this discussion, “premature homogenisation” refers to the loss of sampling density in relevant parts of the posterior, so that the ensemble particles are trapped in some local optimum.

- `permute_source`: randomly permutes a single source in the IPM by altering the PWM weights or changing the length of the source. Distributions for the frequency and size of these permutations may be supplied. If the size of the perturbations is small, and the frequency of length changes is set to 0., this function is useful for finding sources that are slightly more probable than the current ones.
- `permute_mix`: flip a random number of mix matrix positions. Retains flips that make observations more probable. If the range of flip moves is kept small, this is a useful function for “polishing” the mix matrix, particularly of models with highly probable sources. Effective throughout the sampling process.

- `perm_src_fit_mix`: as `permute_source`, but the permuted source is supplied with a new mix vector which mixes the source solely into observations that it makes more probable. Efficient and effective throughout the sampling process.
- `fit_mix`: checks each source against each observation, mixes only those sources that make the observation more probable in isolation. By far the most important permute during the early sampling process, since it can quickly move improbable models sampled from uninformative priors into the portion of the parameter space with likelihoods greater than the background model. In effect, this allows the extraction of maximum information about the data from the initial sampling process, which typically produces many widely divergent sources. This prevents premature homogenisation of the ensemble after extensive sampling below the background model likelihood, and subsequent distortions of evidence and maximum a posteriori parameter estimates.
- `random_decorrelate`: perform the operations of `permute_source` and `permute_mix` on the same model. This is an inefficient permute in general, but it is useful for introducing new sources during the early part of a sampling run, in particular, working to prevent premature homogenisation.
- `shuffle_sources`: copy the source PWM from another model in the ensemble, leaving the original mix matrix. This can be an effective permute early in the sampling process. It is network intensive.
- `accumulate_mix`: randomly select a source from the model to be permuted. From another model in the ensemble, select the source which is most similar to the one selected in the original model. Add the two mixvectors together, such that the original source is now mixed in to any observations in the second model that had the similar source. This is an extremely effective permute when the ensemble begins to have many similar sources in different models, and often is the best way to produce models more likely than those produced by `fit_mix`. Network intensive. This is sometimes the last permute that can find model parameterisations more probable than those in a well-converged ensemble. Network intensive.
- `distance_merge`: Randomly select a source from the model to be permuted. From another model in the ensemble, select the most dissimilar source, and overwrite the selected source in the original model with the PWM and mixvector from this dissimilar source. This can be an effective permute throughout the sampling process. It tends to encourage the propagation of likely sources through the ensemble. Network intensive.
- `similarity_merge`: As `distance_merge`, except the most similar source is selected, and the original mixvector is retained. This tends to be less efficient than `distance_merge`, but can find new models throughout the sampling process. Network intensive.
- `reinit_src`: Resample a random source in the model to be permuted from the prior. This is a very inefficient permute, mostly useful in the early part of the sampling process to reintroduce sources from undersampled parts of the prior.
- `erode_model`: Trim a source's PWM in the model to be permuted to only those positions whose weight vectors express information above a threshold. This has the effect of generating high-likelihood short sources from longer ones with stretches of uninformative positions. It is mostly useful in the early part of the sampling process.

- `info_fill`: For a random source in the permute model, find the least informative PWM position, and set the probability of the most informative base to 1. This function assists in preventing premature homogenisation by rapidly increasing the likelihood of sources that are well-supported by observations but are unlikely to achieve appropriate high PWM weights by random permutation alone.

A `Permute_Instruct` requires a vector of the functions to be used in the permutation routine, a corresponding vector of their initial weights (ie. the frequency with which they will be called), a limit on the number of models to permute before the worker abandons sampling, and a limit on the number of functions to apply to any given model before selecting a new one to attempt permuting. The annotated example provided below also passes, as keyword arguments, vectors of minimum and maximum weights (`min_clamps` and `max_clamps`), as well as a vector of keyword arguments for the vector of functions (`args`).

---

```

1  funcvec=full_perm_funcvec #exported by BMI.jl
2  models_to_permute=10000
3  func_limit=25
4
5  #add two ``polishing functions'' to the function vector
6  push!(funcvec, BioMotifInference.perm_src_fit_mix)
7  push!(funcvec, BioMotifInference.random_decorrelate)
8
9  #all functions will receive a minimum of 1 in 100 function calls
10 min_clamps=fill(.01,length(funcvec))
11
12 #give some important functions a minimum of 10 in 100 function calls
13 min_clamps[2:3]=.1 #perm_src_fit_mix & permute_mix
14 min_clamps[8]=.1 #difference_merge
15
16 #no function will receive more than 50 in 100 function calls
17 max_clamps=fill(.5,length(funcvec))
18
19 #prevent network intensive functions from swamping the cluster
20 max_clamps[6:7]=.15 #shuffle_source and accumulate_mix
21 max_clamps[9]=.15 #similarity_merge
22
23 #evenly distributed initial weights
24 initial_weights= ones(length(funcvec))/length(funcvec)
25
26 #empty arguments vectors for all arguments in the funcvec
27 args=[Vector{Tuple{Symbol,Any}}]() for i in 1:length(funcvec)]
28
29 #add arguments for the ``polishing functions''
30 → args[end-1]=[:weight_shift_freq,0.],(:length_change_freq,1.),(:length_perm_range,1:1)]
```

---

```

31 args[end]=[(:iterates,50),(:source_permute_freq,.3),(:mix_move_range,1:10)]
32
33 #instantiate the Permute_Instruct
34 instruct = Permute_Instruct(funcvec, initial_weights, models_to_permute,
→ func_limit;min_clmps=min_clamps, max_clmps=max_clamps, args=args)

```

---

### 10.3.4 Nested sampling of IPM\_Ensembles

When all of the above tasks are complete, the `IPM_Ensemble` may be converged by nested sampling using the specified permutation routine. This can be performed in parallel; any number of permutation workers may be used. However, because the master process is responsible for disk access, many such workers performing permutation functions which frequently require the deserialization of other models in the ensemble (those noted as “network intensive” above) will rapidly reduce the computational efficiency of the cluster. This can be mitigated by clamping network intensive permute functions to a lower frequency of total function calls.

`BMI.jl` is supplied with a display system similar to, but less customisable than `GMC_NS.jl`. An example of display setup and nested sampling execution is presented below:

---

```

1 display_rotation=[true,10,1,[[:tuning_disp,:lh_disp,:src_disp],[:conv_plot,:liwi_disp,:ens_disp]]]
2
3 logZ = converge_ensemble!(e, instruct, worker_pool, backup=(true,25),
→ tuning_disp=true, lh_disp=true, src_disp=true, disp_rotate_inst=display_rotation)

```

---

## 10.4 Test of model comparison logic with spiked motifs

Because the proper functioning of `BMI.jl` is difficult to verify, we constructed a test problem to ensure that the package can, in principle, be used to distinguish between synthetic observation sets spiked with two different sets of motifs, representing separate causal processes. This is done by generating two sets of 500 synthetic observations of 100-200 bases in length, and spiking them with different periodic, repetitive signals, as well as aperiodic, randomly occurring transcription factor motifs (a TATA box and a CAAT box). Ensembles of 250 `ICA` models with two PWM sources between 3 and 12 bases in length are assembled, one for each of the two separate observations sets, and one to model the combined set. Code to estimate the evidence for these ensembles is available in [Section 17.8.38](#). Execution of this code gives a `logZR` of  $110.87 \pm 0.88$  in favour of the separate models. While the uncertainty on this value is unreliable, the result suggests that the algorithm is capable of distinguishing properly between observations sets generated by separate causal processes on similar backgrounds.

## 10.5 Future Directions

The prospects for `BMI.jl` are limited by the difficulty in ensuring the sampling routines are in detailed balance. It is possible that this could be resolved. One possibility might be to implement a form of `Galilean Monte Carlo` by giving models a three-dimensional coordinate for each base pair in each source

channel (fixing the number of bases per source). This coordinate would encode the location of the model within a tetrahedral probability 3-simplex (this is discussed further in [Section 15.1.7](#), about Position Weight Motif models). The boolean mixing matrix could, perhaps, also be subject to GMC, if at each mix matrix position the model had either positive, negative, or zero velocity.

This solution would not directly solve the degeneracy issues mentioned above, but it may allow for more accuracy in evidence estimation. The dimensionality implied by this parameterisation is nonetheless high. The models used in [Chapter 5](#) contain eight PWM sources of 3-10 bases in length. If we fix the PWM length at 5 bases, there are 120 parameters for the PWM sources alone. Practically speaking, this kind of solution is likely to be best employed where only a few, short, repetitive sequences are of interest.

One way to address the error associated with ad hoc sampling routines might be to construct small toy problems with known model evidence; different sampling routines could be tested on these to perform a stretch test, measuring actual sampler performance against the theoretical distribution of compressive steps [[Buc16](#)]. If this is possible, the effects of the `BMI.jl` tuning regimen could be characterised and error minimised.

## **Part III**

# **Supplementary Materials**

# Chapter 11

## Supplementary materials for Chapter 2

### 11.1 Materials and methods

#### 11.1.1 Zebrafish husbandry

Zebrafish used in this study were of a wild type AB genetic background. Embryos were derived from pairwise crossings. Larvae were collected upon crossing and held in a dark incubator at 28°C. At 1dpf, embryos were treated with 100 $\mu$ l of bleach diluted in 170ml of embryo medium for 3 minutes, rinsed and dechorionated. Embryo medium was changed at 3dpf. After this, all animals were maintained at 28°C on a 14-hour light/10-hour dark cycle (light intensity of 300 lux) in an automated recirculating aquaculture system (Aquaneering). Animals were reared using standard protocols [Wes00]. Fish were sacrificed by tricaine overdose at the appropriate timepoints indicated in the text. All animal experiments were performed with the approval of the University of Toronto Animal Care Committee in accordance with guidelines from the Canadian Council for Animal Care (CCAC).

#### 11.1.2 Proliferative RPC Histochemistry

##### 11.1.2.1 Anti-PCNA histochemistry

In order to assay the number of proliferating cells in zebrafish CMZs, we used anti-PCNA histochemical labelling, as PCNA is, in zebrafish, detectable throughout the cell cycle in proliferating cells. After sacrifice as described above, fish (or their razor-decapitated heads, in the case of fish older than 30dpf) were fixed in 1:9 37% formaldehyde:95% ethanol at room temperature (RT) for 30 minutes, followed by overnight incubation in a fridge at 4°C. Subsequently, the samples were removed from fix and washed 3 times in PBS. They were then cryoprotected by successive 30 minute rinses at RT on a rocker in 5%, 13%, 17.5%, 22%, and 30% sucrose in PBS. Samples were allowed to incubate overnight at 4°C in 30% sucrose. The next day, samples were removed from the sucrose rinse and infiltrated with 2:1 30% sucrose:OCT compound (TissueTek) for 30 minutes at RT. Samples were then embedded for cryosectioning and frozen. All animals younger than 14dpf were embedded side-by-side in groups of 6 under a dissection microscope using a brass mold with milled 3mm channels for maintaining their precise axial orientation.

The cryosectioning block was completed by removing the frozen larval block from the mold and layering more sucrose:OCT mixture between the channels and on top of this filled-in block. Older animals were carefully oriented by hand under a dissection microscope.  $14\mu\text{m}$  coronal cryosections were cut through the fish heads and collected on Superfrost slides (Fisherbrand). These were stored in a freezer at  $-20^\circ\text{C}$  until staining.

Staining was begun by allowing the slides to dry briefly at RT. Sections were outlined with a PAP pen, then rehydrated in PBS for 30 minutes at RT. Subsequently, sections were blocked in 0.2% Triton X-100 + 2% goat serum in PBS for 30 minutes at RT. This was followed by incubation in mouse monoclonal (PC10) anti-PCNA primary antibody (Sigma), diluted 1:1000 in blocking solution, at  $4^\circ\text{C}$  overnight. The next day, primary antibody was removed, and slides were rinsed five times in PDT (PBS + 1% DMSO + 0.1% Tween-20). Cy5-conjugated goat-anti-mouse secondary antibody (Jackson Laboratories), diluted 1:100 in blocking solution, was applied to the sections, which were incubated at  $37^\circ\text{C}$  for 2 hours. Secondary antibody was removed, followed by five rinses in PDT as above. Sections were counterstained with Hoechst 33258 diluted to  $100\ \mu\text{g/mL}$  in PBS for 15 minutes at RT. Counterstain was then removed, five rinses in PDT were performed as above, followed by a final rinse in PBS. Slides were then mounted in ProLong Gold antifade mounting medium (ThermoFisher Scientific), coverslipped, and sealed with clear nail polish. Slides were kept at  $4^\circ\text{C}$  until imaging.

### 11.1.2.2 Cumulative thymidine analogue labelling for estimation of CMZ RPC cell cycle length

In order to obtain an estimate of cell cycle length in CMZ RPCs at 3dpf, we used cumulative thymidine analogue labelling following the method of Nowakowski et al. [NLM89]. 3dpf zebrafish were divided into ten groups of n5 animals and held in 10 mM EdU for 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 8.5, 9.5, and 10.5 hours, after which they were sacrificed as described above. Samples were processed as described under “Anti-PCNA histochemistry”, except that goat-anti-mouse Cy2-conjugated secondary antibody (Jackson Laboratories) was used to label anti-PCNA, after which the Alexa Fluor 647-conjugated azide from a Click-iT kit (ThermoFisher Scientific) was added to the sections for 30 minutes at room temperature to label EdU-bearing nuclei. This was followed by five PDT rinses as described above, and resumption of the protocol with counterstaining, mounting, etc. The raw data is available in `/empirical_data/cumulative_edu.xlsx`

### 11.1.2.3 Whole retina thymidine analogue labelling of post-embryonic CMZ contributions

We used thymidine analogue labelling as an indelible marker of CMZ contributions to the retina in Fig 2.6. Because thymidine analogues are incorporated into DNA during S-phase, postmitotic progeny of proliferating RPCs which take up the label may be detected at any point after this treatment. Dosing zebrafish with a thymidine analogue for a limited period of time ensures that the label is diluted to undetectable levels in cells which continue to proliferate. Repetitive dosing thus gives rise to labelled cohorts which represent a limited set of generations of cells which become postmitotic after the dose is provided. n=3 fish were held in 10mM BrdU (Sigma) diluted in facility water for 8 hours at 30, 60, 90, 120, and 150 days post fertilisation. 10 days after the last BrdU incubation, the fish were sacrificed as described above. One representative whole retina dissection is presented.

### 11.1.3 Confocal microscopy and image analysis

All histochemically processed samples (coronal sections of retinas and whole retinas) were imaged on a Leica TCS SP5 II laser confocal imaging microscope, using the Leica LAS AF software for acquisition. For coronal sections, central sections were identified by their position in the ribbon of cryosections. For Fig 2.8, the central section was imaged together with the flanking section on either side of it. Confocal data was analysed using Imaris Bitplane software (v. 7.1.0). Cell counting analyses were performed by segmenting the relevant channels using the Surfaces tool to produce counts of PCNA positive (Figs 2.8, S4 Fig) or PCNA/EdU double-positive (S4 Fig) nuclei. Lens diameters were measured in the DIC channel using the linear measurement tools, across the widest point of the section of spherical lens.

### 11.1.4 Estimation of CMZ annular population size

In order to estimate the total population of the CMZ in zebrafish eyes sampled throughout the first year of life, we counted PCNA-positive cells in the dorsal and ventral CMZ of central and flanking sections of these eyes as described above. We added the dorsal and ventral populations and took the average of the central and flanking section per-section populations to reduce the possibility of sampling error. We treated this as a sample of a torus mapped to the  $14\mu\text{m}$  sphere segment of lens intersected by the average section. As the surface area of this zone is given by  $S = \pi \cdot d_{lens} \cdot h_{section}$ , and the total surface area of the lens by  $A = \pi \cdot d_{lens}^2$ , where  $d$  is the diameter of the lens and  $h$  the section thickness, we may calculate an estimate of the total population by  $pop_{total} = \frac{pop_{section}}{S} \cdot A$ . We therefore obtained our annular CMZ population estimates using our measurements with the simplified formula  $pop_{total} = \frac{pop_{section} \cdot d_{lens}}{h_{section}}$ , calculating  $pop_{total}$  on a per-eye basis to account for covariance of the measurements, and subsequently averaging across  $n=6$  eyes per sampled timepoint (3, 5, 8, 12, 17, 23, 30, 60, 90, 120, 180, and 360 dpf). These data are presented in Fig 2.8.

### 11.1.5 Modelling lens growth

To simulate the annular CMZ population, we wanted to be able to simulate the stem cells at the periphery of the retina occasionally undergoing symmetric divisions, so as to keep up the same density of stem cells in the first ring around the lens. We did this by normalising our lens diameter measurements to the 72hpf value, to produce a measure of relative increase in lens size over the first year of CMZ activity. We performed a linear regression of the logarithm of this relative increase vs the logarithm of time using the Python statsmodels library, using the constants derived from this regression for a power-law model of lens growth. This model was used to supply the `WanStemCellCycleModel` with a target population value as described below.

### 11.1.6 Estimation of 3dpf CMZ cell cycle length

To produce an estimate of the length of the cell cycle in 3dpf RPCs, we first took the total count of EdU-positive cells in dorsal and ventral CMZ from our cumulative labelling experiment described above, and divided by the total dorsal and ventral CMZ RPC population, as measured by the number of PCNA positive cells. This gave the labelled fraction measurements displayed in S4 Fig. Following Nowakowski et al. [NLM89], we performed an ordinary least squares linear regression on these data using the Python statsmodels library. We then calculated the total cell cycle time  $T_c = \frac{1}{slope}$  and  $T_s = T_c \times intercept_y$ .

While this method is not suited for heterogenous populations of proliferating cells, its use is justified here, as the He SSM to which the estimate is being applied makes similar assumptions as Nowakowski et al., in particular the homogeneity of the population. It should not be considered more than a rough estimate.

### 11.1.7 CHASTE Simulations

#### 11.1.7.1 Project code

All simulations were performed in an Ubuntu 16.04 environment using the CHASTE C++ simulation package version 2017.1 (git repository at <https://chaste.cs.ox.ac.uk/git/chaste.git>). The CHASTE package is a modular simulation suite for computational biology and has been described previously [MAB<sup>+</sup>13]. All of the code, simulator output, and empirical data used in this paper is available in the associated CHASTE project git repository at <http://github.com/mmattocks/SMME>. In brief, the approach taken was to encapsulate the SSMs examined here as separate concrete child classes inheriting from the CHASTE `AbstractSimpleCellCycleModel` class. An additional model, representing the simple immortal stem cell proposed in Wan et al. [WAR<sup>+</sup>16], was produced similarly. Each model is generic and provided with public methods to set their parameters and output relevant simulation outcomes (and per-lineage debug data, if necessary). While these may be used in the ordinary CHASTE unit test framework, permitting their use in any kind of cell-based simulation, we wrote standalone simulator executables (apps, in CHASTE parlance) to permit Python scripting of the simulator scenarios used in this study. This allows for the multi-threaded Monte Carlo simulations performed herein. Therefore, the Python fixtures available in the project's `python_fixtures` directory were used to operate the single-lineage simulators (`GomesSimulator`, `HeSimulator`, and `BoijeSimulator`) and single-annular-CMZ simulator (`WanSimulator`) in `apps/`, including the `CellCycleModels` and related classes in `src/`; the simulators output their data into `python_fixtures/testoutput/`. These data, along with the empirical observations (both from the Harris groups' papers and our own described herein) available in `empirical_data/`, were processed by the analytical Python scripts in `python_fixtures/figure_plots/` to produce all of the figures used in this study.

We have attempted to make the project fully reproducible and transparent. CHASTE uses the C++ boost implementation of the Mersenne Twister random number generator (RNG) to provide cross-platform reproducibility of simulations. All of our simulators make use of this feature, permitting user-specified ranges of seeds for the RNG. We have also used the numpy library's RandomState Mersenne Twister container to provide reproducible results for the Python fixtures, notably the SPSA optimisation fixture. The output of the project code will, therefore, be identical every time it is run, on any platform.

It should be noted that none of the code Harris' group used in their reports has been published. We have made every effort to replicate the functional logic of the models as closely as possible. Nonetheless, it is not possible to determine precisely why, for instance, the original He SSM parameterisation failed so notably in our implementation.

#### 11.1.7.2 SPSA optimisation of models

In order to make a fair comparison between the He SSM and our deterministic mitotic mode alternative model, we coded an implementation of the SPSA algorithm described by Spall [Spa98]. This is available in `/python_fixtures/SPSA_fixture.py`. This algorithm, properly tuned, will converge almost-surely

on a Karush-Kuhn-Tucker optimum in the parameter space, given sufficient iterates, even with noisy output (eg. due to RNG noise from low numbers of Monte Carlo samples, permitting conservation of computational resources). It does so by approximating the loss function gradient around a point in parameter space, selecting two sampling locations at each iterate, then moving the next iterate's point in parameter space "downhill" along this gradient.

Our loss function was a modified AIC; we weighted the residual sum of squares from the PD-type mitotic probabilities by 1.5 in order to improve convergence, as the PD data are the most informative part of the dataset regarding the critical phase boundaries (PP mitoses occur in all 3 phases of the He model, DD occur in phases 2 and 3, PD mitoses occur only in phase 2). We also compared model induction count output (that is, panels A-C in Fig 2.5) up to count values of 1000 to penalise parameters producing very large lineage totals.

The values of the SPSA gain sequence constants were selected to be as large as possible without resulting in instability and failure to converge; this ensured that the algorithm stepped over the widest possible range of the parameter space in finding optima. The  $\alpha$  and  $\gamma$  coefficients were initially set at the noise-tolerant suggested values of .602 and .101 respectively [Spa98]. 200 iterations were performed. Initially, each iterate involved 250 seeds of each induction timepoint for the count data (panels A-C in Fig 2.5) and 100 seeds of the entire 23-72 hr simulation time for mitotic mode rate data (panels D-F). At iterate 170, these were increased to 1000 and 250 seeds, respectively, decreasing RNG noise to a low level. For the last 10 iterations, RNG noise was reduced to close to nil by increasing the seed numbers to 5000 and 1250, respectively;  $\alpha$  and  $\gamma$  were set to the asymptotically optimal 1 and  $\frac{1}{6}$  for these iterates, as well. The particular seeds used were kept constant throughout in order to take advantage of the improved convergence rate this affords [KSN99].

Because the simple SPSA can assign any value to parameters, our implementation uses constraints, projecting nonsensical values for parameters (e.g. negative values, mitotic mode probabilities summing to  $> 1$ ) back into legal parameter space. The use of constraints has no effect on the algorithm's ability to almost-surely converge within the given parameter space [Sad97] As we felt that the deterministic mitotic mode models' "sister shift" should be relatively small in order to be biologically plausible, we also constrained this parameter such that 95% of sister shifts would be less than the mean length of the shortest phase.

The numerical results of the SPSA algorithm are available in `/python_fixtures/testoutput/SPSA/HeSPSAOutput`, alongside png images of output from each iterate, enabling the visualisation of the progress of the algorithm.

**Monte Carlo simulations** Subsequent to SPSA optimisation, the parameters derived from this procedure were used to perform Monte Carlo simulations of large numbers of individual lineages, using ranges of 10000 non-overlapping seeds for each of panels A, B, C, G, H, and I, and 1000 for each of panels D, E, and F of Fig 2.5 (using the SPSA-optimised parameter sets) and [S1 Fig](#) (using the original He et al. parameterisation). This was performed using `/python_fixtures/He_output_fixture.py`.

Similarly, 100 seeds were used in Monte Carlo fashion to simulate individual annular CMZ populations using `/python_fixtures/Wan_output_fixture.py`, which scripts the WanSimulator. Each of these simulations is initialised with an RPC population drawn from a normal distribution given the mean and standard deviation of the CMZ torus estimated from our 3dpf observations, as described above. Each RPC is given a TiL value randomly chosen from 0 (newly born from a stem cell) to 17hr (exiting the

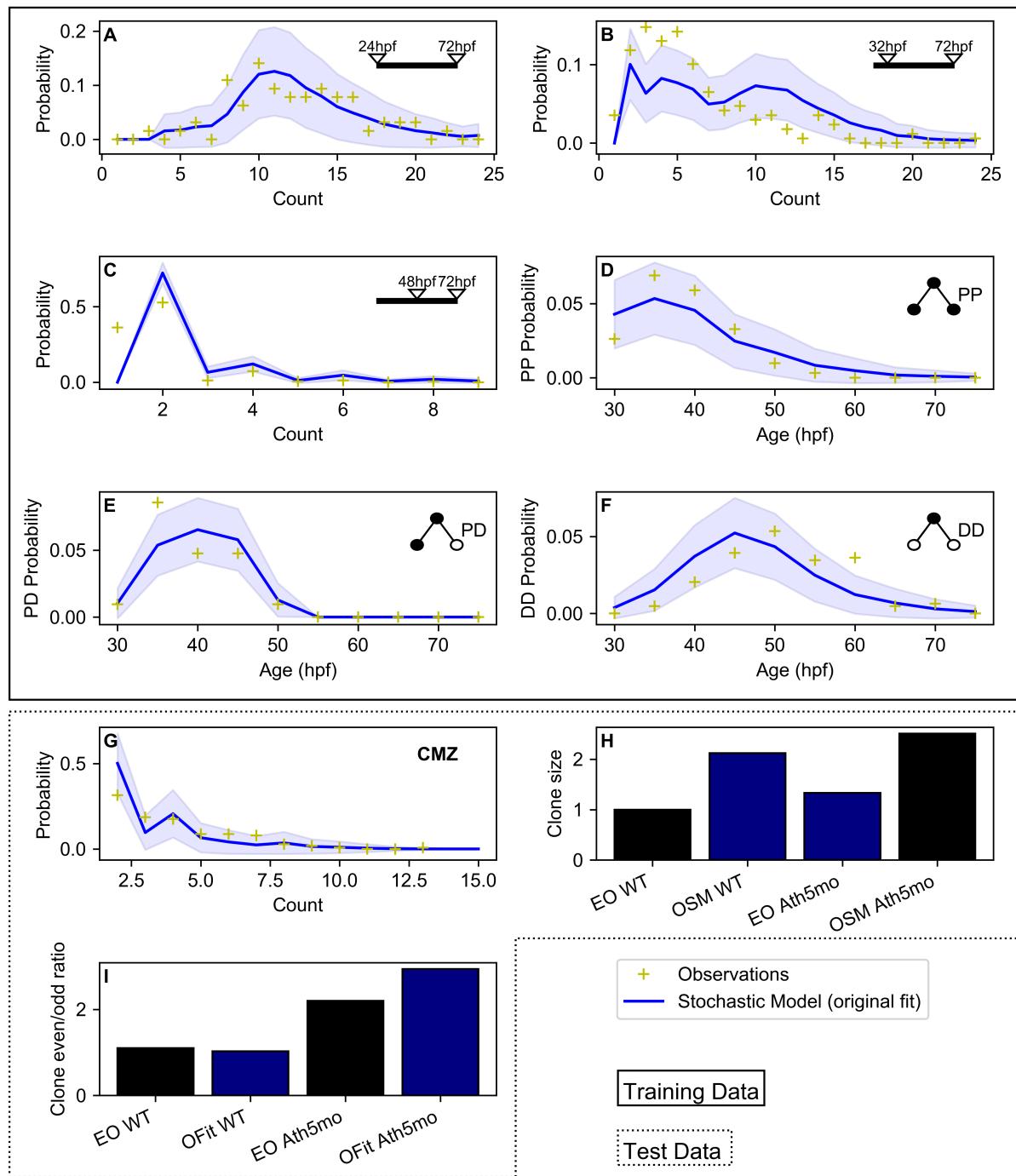
CMZ, “forced to differentiate”, in the terms of Wan et al.). A further population of stem cells, sized at  $\frac{1}{10}$  of the RPC population, is added using the `WanStemCellCycleModel`. This stem population divides asymmetrically except when it falls under its target population value determined by the model of lens growth described above; as long as this condition holds, the stem cells will divide symmetrically to keep up their numbers.

**Simulation data analysis** Not all of the numerical data used in He et al. and Wan et al. has been published. As such, we reconstructed the lineage data from He et al.’s Figure 5C, used by He et al. to obtain their mitotic mode probabilities; our reconstructed values are available in `/empirical_data/empirical_lineages\`. We also reconstructed the He et al. WT and Ath5 morpholino data (Fig 2.5 panels H, I) and Wan et al. lineage count probabilities (panel G) from the relevant figures. These values are entered into `/python_fixtures/figure_plots/He_output_plot.py`, where they are used in the AIC calculations described below.

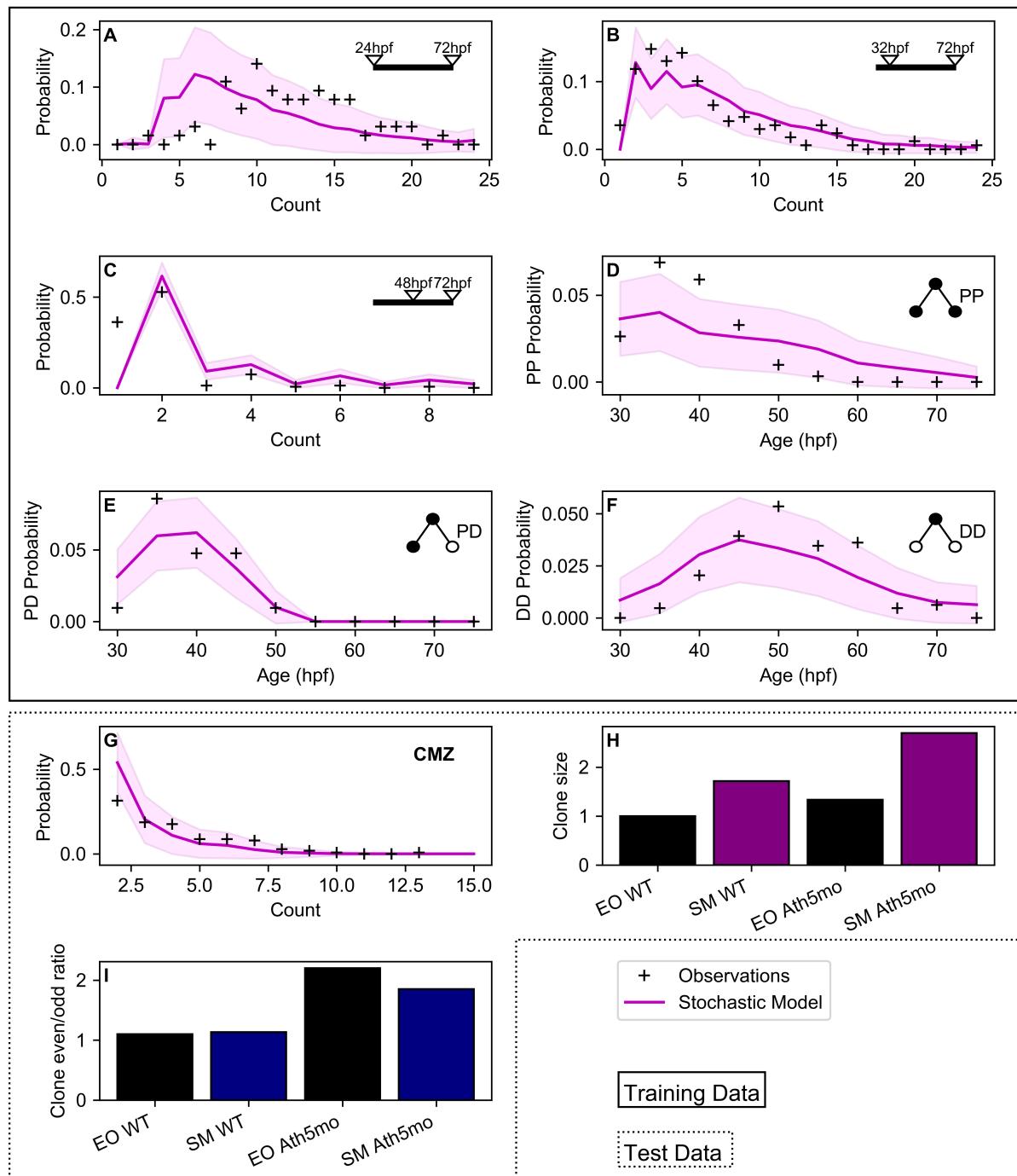
In order to assess the performance of the models used in our simulations, we used Akaike’s Information Criterion (AIC). This allows us to compare models with dissimilar numbers of parameters, as AIC trades off goodness-of-fit against parameterisation. The values reported in Table 2.1 were produced by `/python_fixtures/figure_plots/He_output_plot.py`. Residual sums of squares were combined across the training and test datasets.

The 95% CIs for the model output presented in this paper were estimated by repetitive sampling of the output data, using the same number of lineages observed empirically. 5000 such samples were performed. This provides a reasonable estimate of the confidence interval given the limited observational sample size.

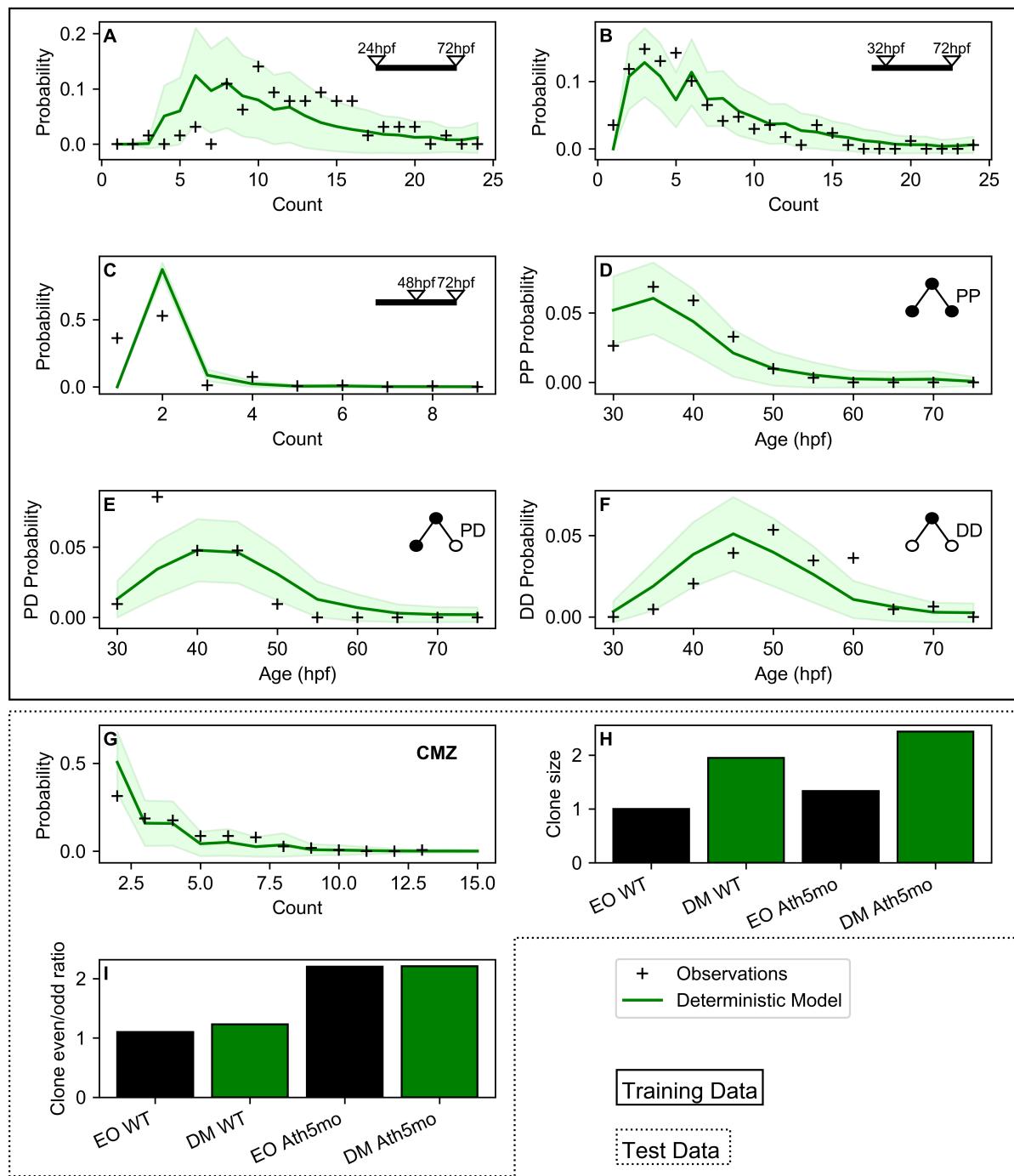
## 11.2 Supporting figures



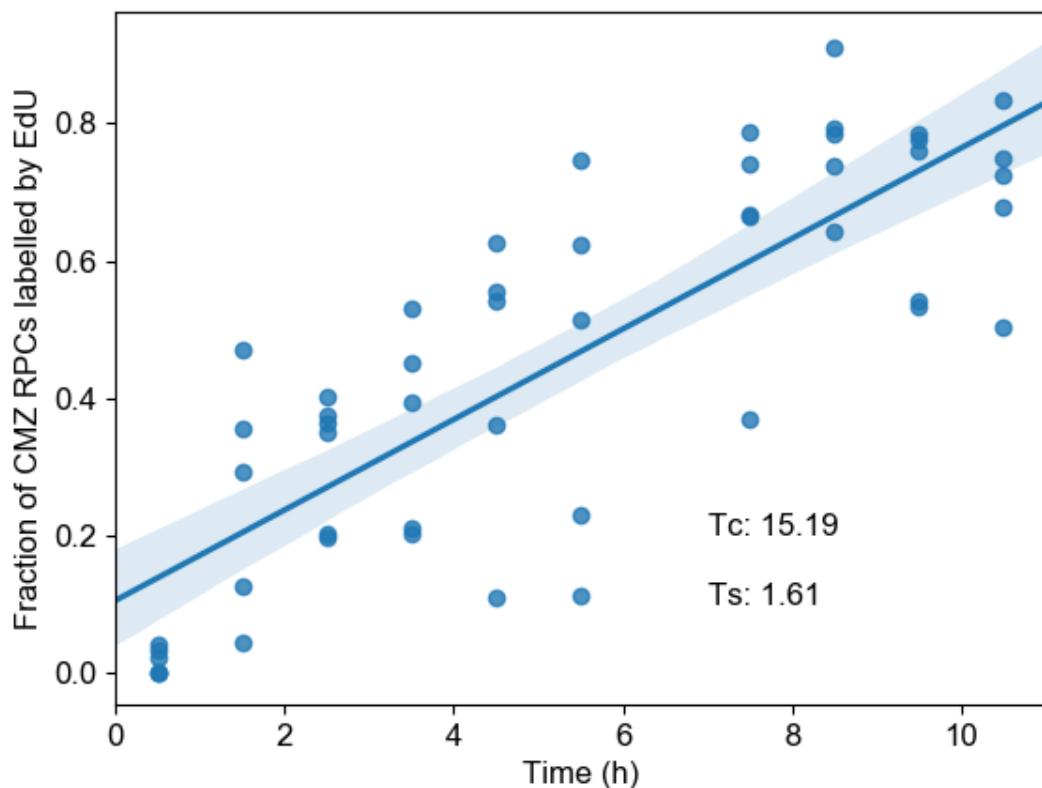
**S1 Fig. He SSM original parameterisation model output.** As Fig 2.5; model output uses the parameterisation given in He et al. [HZA<sup>+</sup>12]. Note significant deviation from observations in panel B, reflecting excess proliferative activity of modelled RPCs.



**S2 Fig. SPSA-optimised He SSM model output.** As Fig 2.5; stochastic model output displayed alone.



**S3 Fig. SPSA-optimised deterministic mitotic mode model output.** As Fig 2.5; deterministic model output displayed by itself.



**S4 Fig. Cumulative EdU Labelling of 3dpf CMZ RPCs** Fraction of PCNA-labelled CMZ RPCs which bear EdU label over time, as determined by histochemical labelling of central coronal cryosections of 3dpf zebrafish retinas. Dots represent results from individual retinas. Line is the ordinary least squares fit  $\pm$  95% CI. Cell cycle length (Tc) and S-phase length (Ts) are estimated from this fit following the method of Nowakowski et al. [NLM89]

# Chapter 12

## Supplementary materials for Chapter 4

### 12.0.1 Normal and LogNormal models of population data

We first investigated the appropriateness of Normal vs. LogNormal models for measured and calculated parameters of the CMZ and the neural retina, presented in [Table 12.1](#).

Table 12.1: Likelihood ratio comparison between Normal and Log-Normal models of retinal population parameters

Parameter	$\mathcal{N}$ logLH	Log- $\mathcal{N}$ logLH	logLR
Sectional PCNA+ve	-285.611	<b>-283.214</b>	2.397
Lens diameter	<b>-203.102</b>	-203.854	-0.752
CMZ annular pop.(†)	-484.768	<b>-482.733</b>	2.035
RPE length	<b>-368.232</b>	-368.609	-0.377
CR thickness (†)	<b>-240.858</b>	-241.032	-0.174
CR volume (†)	-1091.615	<b>-1091.146</b>	0.469

$\mathcal{N}$ : Normal distribution. logLH: logarithm of  $p(D|M)$ , the likelihood of the data given the model. logLR: logarithm of the likelihood ratio; positive ratios in favour of the log- $\mathcal{N}$  model. Largest likelihoods are bolded. †: Calculated quantities. Sectional PCNA+ve: population of PCNA-positive CMZ RPCs per  $14\mu\text{m}$  cryosection. CMZ annular pop: population of annular CMZ. RPE: retinal pigmented epithelium. CR: cellular retina. Methods in [Section 11.1.2.1](#) and [Section 12.1.7](#). Code in [Section 17.8.9](#).

These results suggest that the organism-level population distribution of eye-level CMZ population counts, as assayed by PCNA immunostaining, is better modelled Log-Normally than Normally. This is true whether we test the primary per-section count measurements, or the estimated whole-annulus population, even though this quantity is calculated using the Normally-distributed lens diameter<sup>1</sup>. The most likely Log-Normal representations of the CMZ populations are about two orders of magnitude more likely than the Normal alternative. Additionally, although Normal models are slightly favoured for describing the population-level distributions of RPE length and cellular retina thickness, the derived cellular retina volume quantity is better modeled by a Log-Normal distribution.

<sup>1</sup>The apparent superiority of the Normal model for lens diameter may be due to the paucity of data at later time points; as described in [Section 12.1.7](#), lenses are difficult to retain in this histological context.

Since the choice of model describing the estimated annular population and retinal volume is critical to the success of later inferences, we used Galilean Monte Carlo-Nested Sampling (GMC-NS) to estimate the Bayesian evidence (the marginal probability of the data over all model parameterisations) for these hypotheses. Evidence estimates and ratios are presented in Table 12.2.

Table 12.2: Evidence favours Log-Normal models of retinal population parameters

Parameter	$\mathcal{N} \log Z$	Log- $\mathcal{N} \log Z$	logZR	$\sigma$ significance
CMZ Population	$-1457.0 \pm 3.7$	<b><math>-1005.0 \pm 3.5</math></b>	$452.1 \pm 5.1$	89.45479335402516
Estimated Retinal Volume	$-2042.2 \pm 3.3$	<b><math>-1525.2 \pm 4.1</math></b>	$517.0 \pm 5.3$	97.87822790648408

$\mathcal{N}$ : Normal distribution.  $\log Z$ : logarithm of  $p(D)$ , the marginal likelihood of the data, or model evidence. logZR: evidence ratio; positive ratios in favour of the log- $\mathcal{N}$  model. Largest evidence values bolded. CR: cellular retina. Methods in [Section 12.1.8](#). Code in [Section 17.8.7](#).

Full estimation of the Bayesian evidence for the Normal vs. Log-Normal hypotheses for our calculated parameters produces confirms the rough calculation of likelihood ratio from the single-fit MAP models, although the extent to which LogNormal models are better justified than Normal differs between the methods.

If between-individual variation in retinal CMZ population and retinal volume are both well-modelled Log-Normally, the question of their independence arises. It seems plausible that the size of the CMZ population would be roughly proportional to the overall volume of the retina, after postembryonic central specification, and establishment of the peripheral CMZ remnant. Moreover, since the growth of retinal volume over the life of the organism is driven primarily by the CMZ<sup>2</sup>, it seems likely that this volume is well correlated with the size of the CMZ. Since the manner in which we model the CMZ differ depending on our assessment of the dependence of these retinal parameters, we performed Bayesian model selection by the [Empirical Bayes](#) method for linear regression, which provides for direct estimation of the evidence for models consisting of linear equations of variables [Bis06].

We find that individual CMZ population and retinal volume estimates are best described by uncorrelated models at in all ages, with the exception of 23.0 dpf, where the evidence weakly favours correlation. It is interesting that the evidence in favour of non-correlation is weakest between 17 and 30 dpf, suggesting the correlation at 23.0 dpf is not spurious, but is related to CMZ activity at this time. These data are displayed in [Table 12.3](#). Of particular importance are the data for 3dpf embryos, as we intend to seed model retinae with CMZ populations and volumes drawn from these distributions. The data for these animals are plotted in [Figure 12.1](#). The general lack of correlation between CMZ population and retinal volume estimates may be due to the loss of information involved in the estimation calculations; on the other hand, it is more plausible that CMZ population should be associated with the rate of retinal volume growth rather than the volume of the retina itself, which suggests that the rate of retinal contribution from the CMZ is probably highest between 17 and 23 days. Because growth rate data are unavailable for single individuals, we cannot make this inference directly.

---

<sup>2</sup>One observes occasional proliferative clusters in the central retina throughout the life of the fish; these are typically ascribed to Müller glial repair processes. I am unaware of any estimate as to the relative contribution of these clusters vs. the CMZ. As we shall see, there is probably more turnover in the specified retina than previously believed. As a result, the relative contribution of these central clusters should probably be subject to statistical estimation; they may be more significant than mere lesion-repair sites.

Table 12.3: Evidence favours uncorrelated linear models of CMZ-population and retinal volume over time

Age (dpf)	Uncorrelated logZ	Correlated logZ	logZR
3.0	<b>-87.651</b>	-91.902	4.252
5.0	<b>-90.0</b>	-92.362	2.362
8.0	<b>-90.918</b>	-96.013	5.095
12.0	<b>-91.183</b>	-99.049	7.866
17.0	<b>-94.818</b>	-96.668	1.85
23.0	-103.386	<b>-103.219</b>	0.167
30.0	<b>-103.511</b>	-104.092	0.581
60.0	<b>-115.025</b>	-118.169	3.144
90.0	<b>-113.533</b>	-122.427	8.894
180.0	<b>-116.778</b>	-124.547	7.769
360.0	<b>-121.016</b>	-128.637	7.621

logZ: logarithm of  $p(D)$ , the marginal likelihood of the data, or model evidence. logZR: evidence ratio; positive ratios in favour of the uncorrelated model. Largest evidence values bolded. Methods in Section 12.1.8, Section 12.1.10. Code in Section 17.8.1.

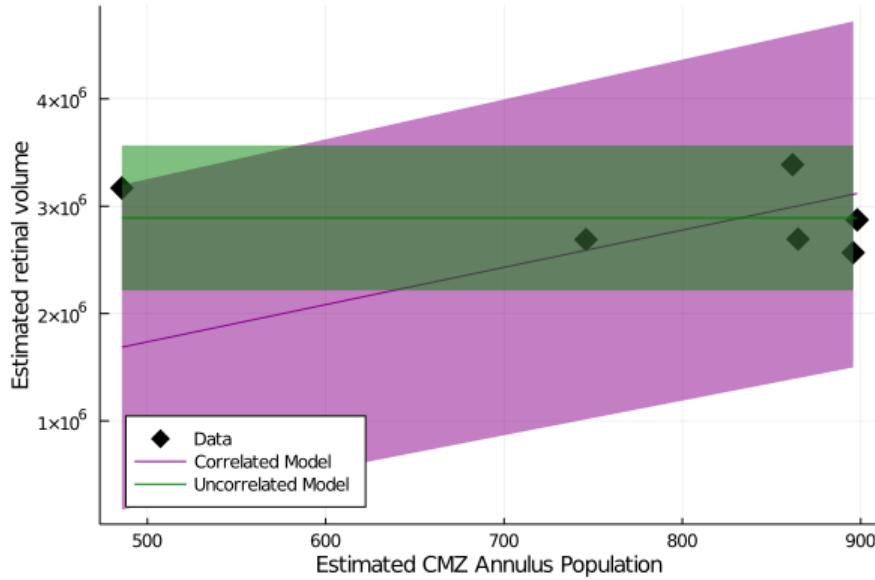


Figure 12.1: CMZ population and retinal volume estimates are uncorrelated at 3dpf  
Individual CMZ population estimate vs retinal volume estimate for 3dpf animals. Uncorrelated and correlated linear models of these variables are plotted as the mean±95% CI of the predictive distribution of the fitted model. Methods in Section 12.1.8, Section 12.1.10. Code in Section 17.8.1.

## 12.1 Materials and methods

### 12.1.1 Zebrafish husbandry

Zebrafish husbandry was performed as described in Section 11.1.1.

### 12.1.2 PCNA Immunohistochemistry

Anti-PCNA immunohistochemistry was performed as described in Section 11.1.2.1.

### 12.1.3 Thymidine analogue histochemistry

Both cumulative thymidine analogue histochemistry presented in Section 4.6, as well as the pulse-chase histochemistry used for the study of neural fate in Section 4.8 were performed by the methods described in Section 11.1.2.2 and Section 11.1.2.3, respectively, except that the data from the whole retinal labelling experiment derives from 14micrometre transverse and coronal cryosections (n=3-6 animals per axis, per age).

### 12.1.4 Lineage tracing immunohistochemistry

Embryos from wild-type AB crosses (staining groups 2 and 3) and *Tg(Isl2b:GFP)* crosses (staining group 1). The *Tg(Isl2b:GFP)* line was the kind gift of the late Dr. Chi-Bin Chien. Embryos were collected and reared as described in Section 11.1.1. Embryos were divided into three groups, to be pulsed with EdU at 3, 23, and 90 dpf and sacrificed after a 7 day chase period. In 3 and 23dpf larvae, animals were pulsed by allowing them to swim freely in a 10mM EdU solution for 2 hours. In 90dpf animals, 10microliters of 10mM EdU was injected intraperitoneally. After 7 days, animals were sacrificed and coronal cryosections were obtained as described in Section 11.1.2.1. These cryosections were divided into staining groups according to their genetic origin, as mentioned above.

Cryosections from all staining groups were allowed to dry briefly at room temperature (RT), before rehydration in PBS for 30 minutes at RT. Subsequently, sections were blocked in 0.2% Triton X-100 + 2% goat serum in PBS for 30 minutes at RT. Primary antibody incubations were as follows for each numbered staining group:

1. Covance rabbit anti-Pax6 primary antibody diluted 1:200 in blocking solution incubated at 4° overnight.
2. Chemicon mouse anti-GS primary antibody diluted 1:500 in blocking solution at 4° overnight.
3. ZIRC mouse anti-Zpr1 primary antibody diluted 1:400 in blocking solution at 4° overnight.

After these overnight incubations, primary antibodies were removed and all sections were rinsed five times in PDT (PBS + 1% DMSO + 0.1% Tween-20). Secondary antibody incubations were as follows for each numbered staining group:

1. Goat anti-rabbit Cy3-conjugated 2° antibody diluted 1:100 in blocking solution at 37°C for 2 hr.
2. Goat anti-mouse Cy2-conjugated 2° antibody diluted 1:100 in blocking solution at 37°C for 2 hr.
3. Goat anti-mouse Cy2-conjugated 2° antibody diluted 1:100 in blocking solution at 37°C for 2 hr.

Staining group 2 was subsequently incubated with Santa Cruz rabbit anti-PKC $\beta$  1° antibody diluted 1:100 in blocking solution at 4°C overnight. The next day, this second primary antibody was removed, and the group sections were rinsed in PDT as above. A further goat anti-mouse Cy2-conjugated 2° antibody diluted 1:100 in blocking sol'n was applied at 37°C for 2 hr.

After each staining group's final secondary antibody was removed, slides were rinsed 3x in PDT, then stained for EdU with an Alexa 647 azide. After another three PDT rinses, 100 ug/mL Hoechst 33258 was applied for 15 minutes at RT. Five PDT rinses, after which all slides were mounted, slipped, and sealed.

### 12.1.5 4C4 immunohistochemistry

The 4C4 antibody was a generous gift of Dr. Pamela Raymond. It has been empirically determined to mark microglia [BB01]. It was used according to the methods described above with a Cy2 secondary antibody.

### 12.1.6 Confocal micrograph acquisition and processing

All confocal micrographs presented in Chapter 4 were acquired on a Leica TCS SP5 II microscope, operated using the LAS AF microscopy software (Leica). Data was assessed by using Bitplane Imaris software (v1.5.1). Specifically, the Imaris watershed segmentation algorithm was applied to the nuclear counterstain channel to segment cellular nuclei. Colocalisation of particular markers (ie. PCNA, EdU, or lineage markers) was assessed by thresholding signal from these channels with the segmented nuclear volumes to produce a subpopulation of colabelled nuclei. The segmented Imaris scenes and the associated settings are available in the thesis HDD archive.

### 12.1.7 Retinal measurements

The developmental morphology study presented in Figure 12.2 was conducted by using the LAS AF software measurement tools on transmitted light data collected in the process of conducting the confocal survey presented in Figure 4.2. Each measurement point is, therefore, likewise the average of the value three central coronal cryosections. Retinal thickness was the linear thickness of the entire cellular retina and its plexiform layers at the center of the section, and the layer thicknesses reflect segments along this line, with the addition of RPE thickness. RPE length was measured in approximating segments from the dorsal to the ventral extrema. Lens diameter was measured in any sections which retained the lens; this was rare at older ages, when the lens sections tend to separate readily as soon as the cryosection is rehydrated. Optic nerve diameter was measured at its thickest point.

### 12.1.8 Estimation of overall CMZ population and retinal volume

The overall CMZ population estimates were derived by the calculations presented in Section 11.1.4. The volume of the cellular retina was crudely estimated by subtracting the volume of two spheres. The circumference of the outer sphere is given by the RPE length plus the diameter of the lens. The radius of the inner sphere is the half total thickness of the cellular retina subtracted from radius of the outer sphere<sup>3</sup>. The difference in volume between the two spheres is calculated; four-fifths this value is taken as the volume estimate in order to partially compensate for the flattening of the eye at older ages.

---

<sup>3</sup>Halving serves as crude compensation for the lack of cellular retina on the lens' part of the sphere, as well as the thinning of the cellular retina toward the periphery

### 12.1.9 Monte Carlo estimation of CMZ population and retinal volume rates of change

In order to estimate the mean daily rate of change of the CMZ population and retinal volume estimates, we performed Monte Carlo difference operations between samples drawn from the estimated LogNormal distributions at subsequent timepoints. This allows us to empirically estimate the uncertainty on these rates, as the difference between two T-distributions is not, itself, T-distributed. The relevant code is found in [Section 17.8.9](#), and uses functions from `NGRefTools.jl`.

### 12.1.10 Evidence estimation by Empirical Bayes linear regression

Evidence comparisons presented without estimated standard deviations ( $\sigma$ ) of significance are produced by the [Empirical Bayes](#) method of linear regression, as presented in Bishop's standard machine learning textbook [[Bis06](#)]. Calculations were performed using the Julia package `BayesianLinearRegression.jl`, which implements Bishop's algorithm. Values are presented in log base  $e$ .

### 12.1.11 Evidence estimation by Galilean Monte Carlo nested sampling

Evidence comparisons presented with estimated standard deviations ( $\sigma$ ) of significance are produced by converging ensembles of [Galilean Monte Carlo](#) sample chains, which collectively constitute a sample from the posterior in rigorous detailed balance [[Ski19](#)]. The [Bayesian evidence](#), or marginal probability of the model over the posterior sample is calculated by the [nested sampling algorithm](#) [[Ski06](#)], and is presented in log base  $e$ . Calculations were performed by the use of the Julia packages `GMC_NS.jl` and `CMZNicheSims.jl`, written for this thesis and presented in [Chapter 7](#) and [Chapter 8](#), respectively. Maximum a posteriori outputs are calculated from the most likely model parameterisation sampled during this process.

### 12.1.12 Estimation of posterior distributions of phase model parameters

Posterior distributions presented in [Figure 4.4](#) and [Figure 4.7](#) were produced by Kernel Density Estimation (KDE) [[Bis06](#), p. 122]. The KDE algorithm used was the implementation available in the Julia Robotics library `KernelDensityEstimate.jl`, described by Sudderth et al. [[SMFW10](#)]. Briefly, this uses multiscale nonparametric Gibbs sampling of supplied weighted “point clouds” to generate the kernel density estimate. Chains from `GMC_NS.jl` were used in this context by weighting the points in parameter space by their estimated evidence. It is important to note that, for this thesis, `GMC_NS.jl` was configured to prioritise the accuracy of evidence estimates over these posterior estimates; as a consequence, samples from around the maximum a posteriori have little weight in the KDE estimates presented herein [[HHHL18](#)]. Alternatives to this scheme are possible; one such algorithm is described by Higson et al. [[HHHL19](#)]

## 12.2 Supplementary Figures

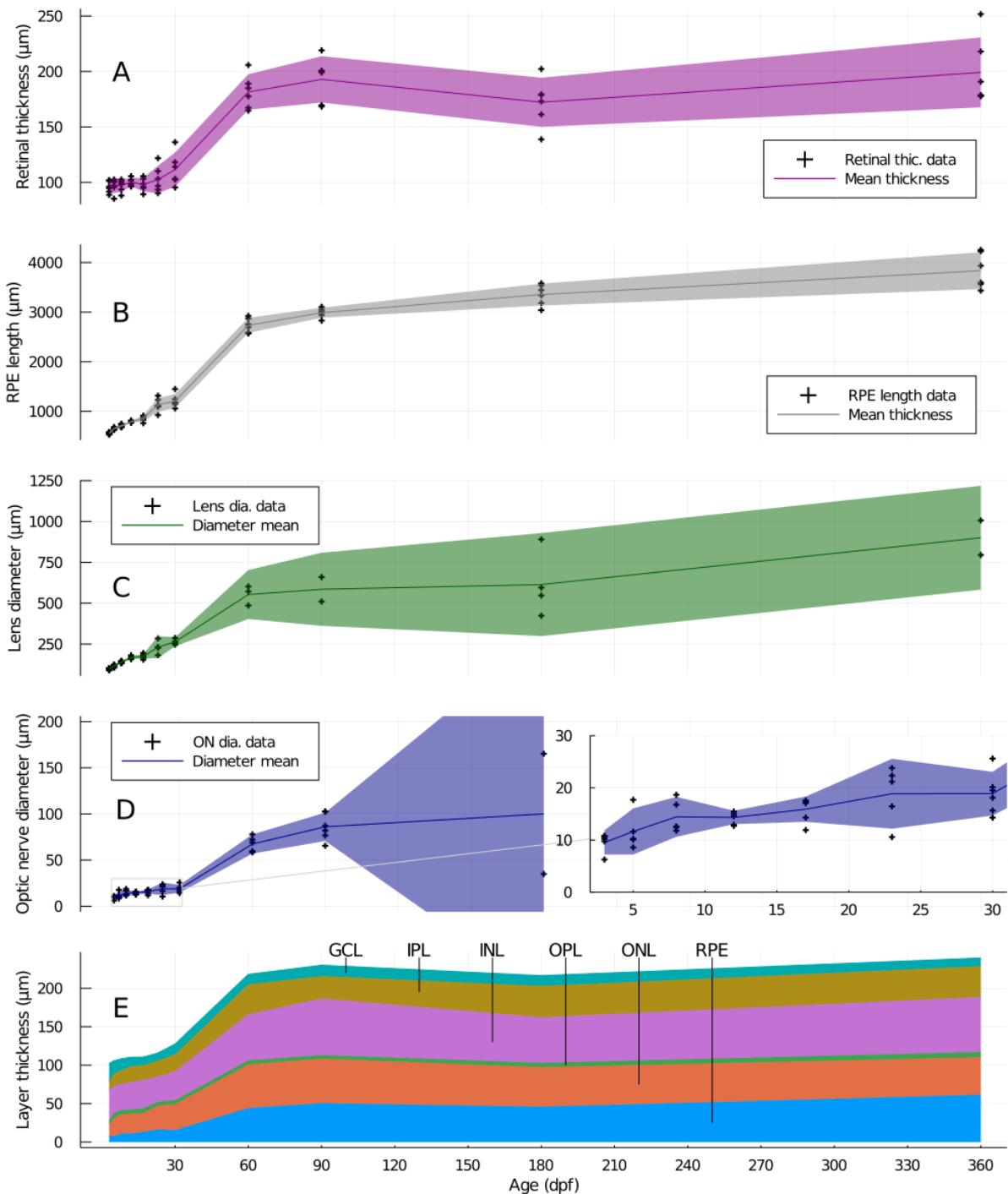


Figure 12.2: Overview of *D. rerio* retinal parameters during morphogenesis

All estimates are presented as the marginal posterior distribution of the mean, from uninformative priors, with 95% credibility intervals. Panel A: Thickness of the retina, measured in its center. Panel B: Length of the retinal pigmented epithelium, measured from one side of the lens to the other. Panel C: Diameter of the lens. Panel D: Diameter of the optic nerve, measured in its center. Panel E: total thickness of the retina, divided by the thicknesses of the layers that compose it. GCL: Ganglion cell layer. IPL: Inner plexiform layer. INL: Inner nuclear layer. OPL: Outer plexiform layer. ONL: Outer nuclear layers. RPE: Retinal pigmented epithelium.

Methods in Section 12.1.7. Code in Section 17.8.6.

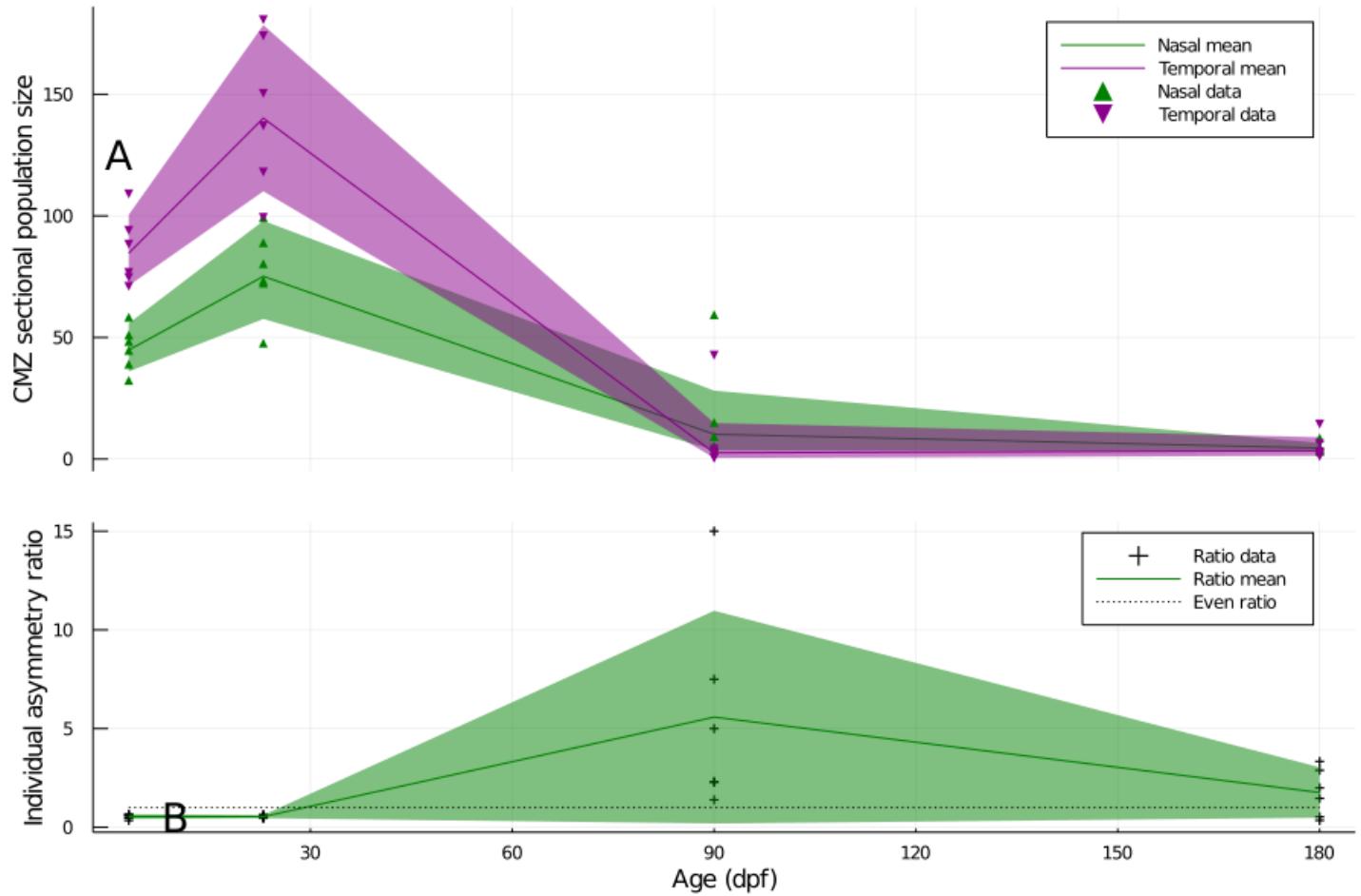


Figure 12.3: **Developmental progression of naso-temporal population asymmetry in the CMZ.**

Marginal posterior distribution of mean nasal (N) and temporal (T) population size in  $14\mu\text{m}$  transverse cryosections (panel A) or intra-individual N/T count asymmetry ratio (panel B),  $\pm 95\%$  credible interval,  $n=6$  animals per age. Data points represent mean counts from three central sections of an experimental animal's eye. Code in Section 17.8.11.

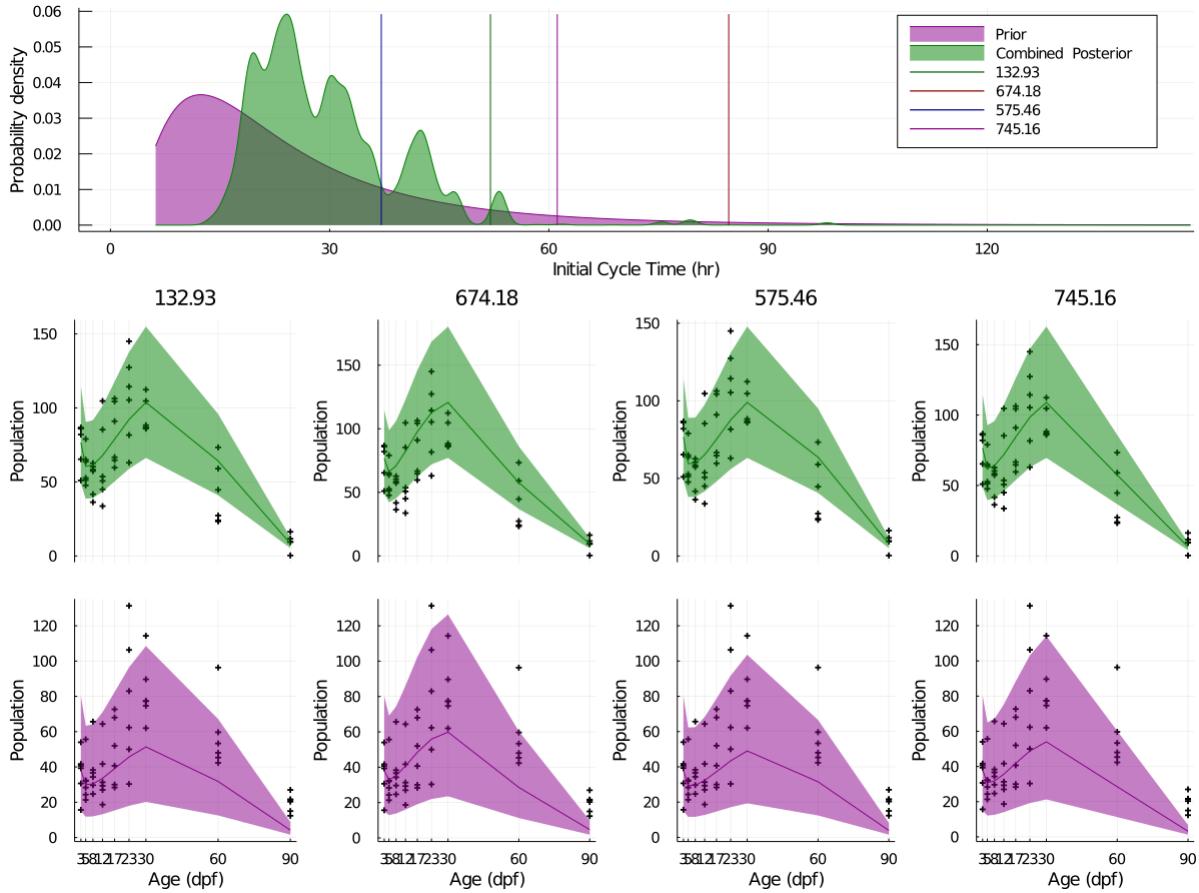


Figure 12.4: **Polymodality of CMZ model MAP estimates**

Four of the six most likely models found in the MAP sample of the combined decay slice model presented in Figure 4.9, located on the marginal distribution of initial cycle time (top panel), and with model output displayed in bottom panels. Note that very different initial CT values produce comparable explanations of the data. Code in Section 17.8.2.

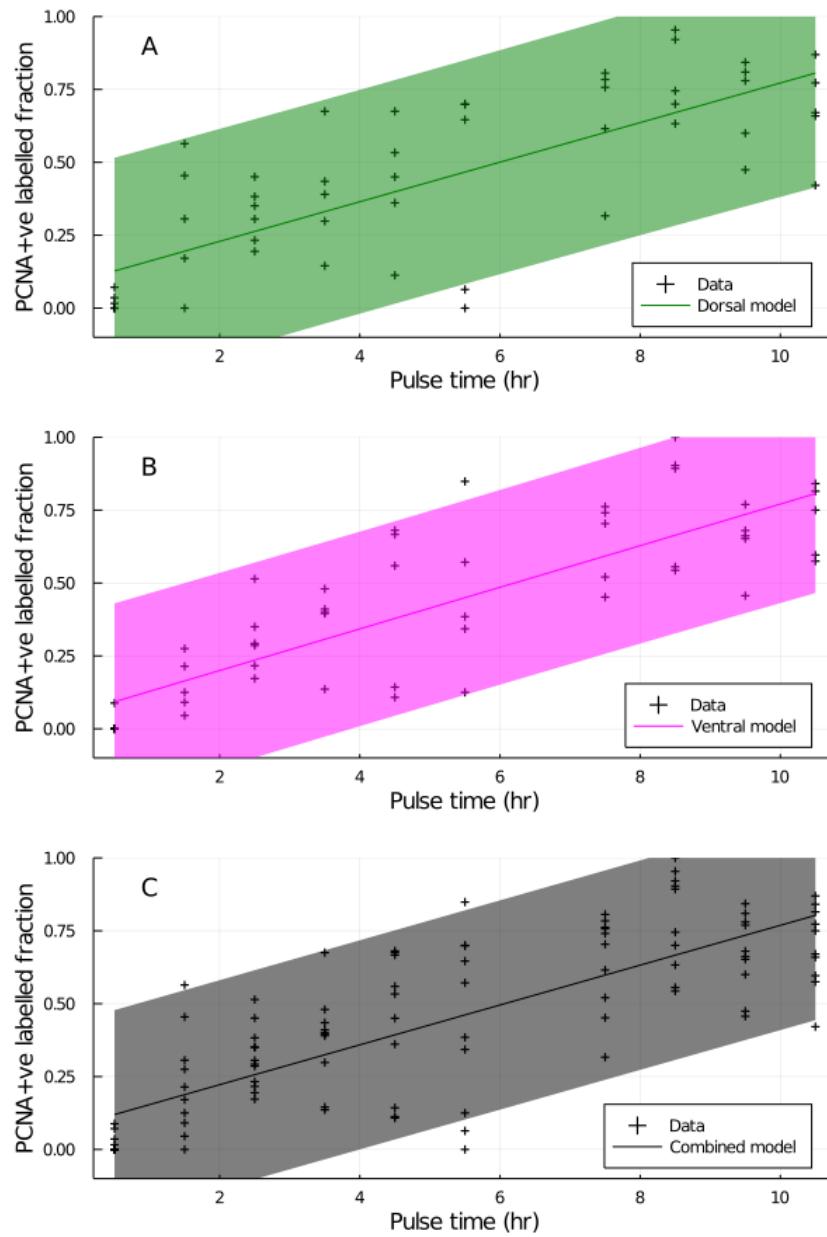


Figure 12.5: Linear regressions performed on cumulative labelling data from dorsal, ventral, and combined CMZ sectional populations

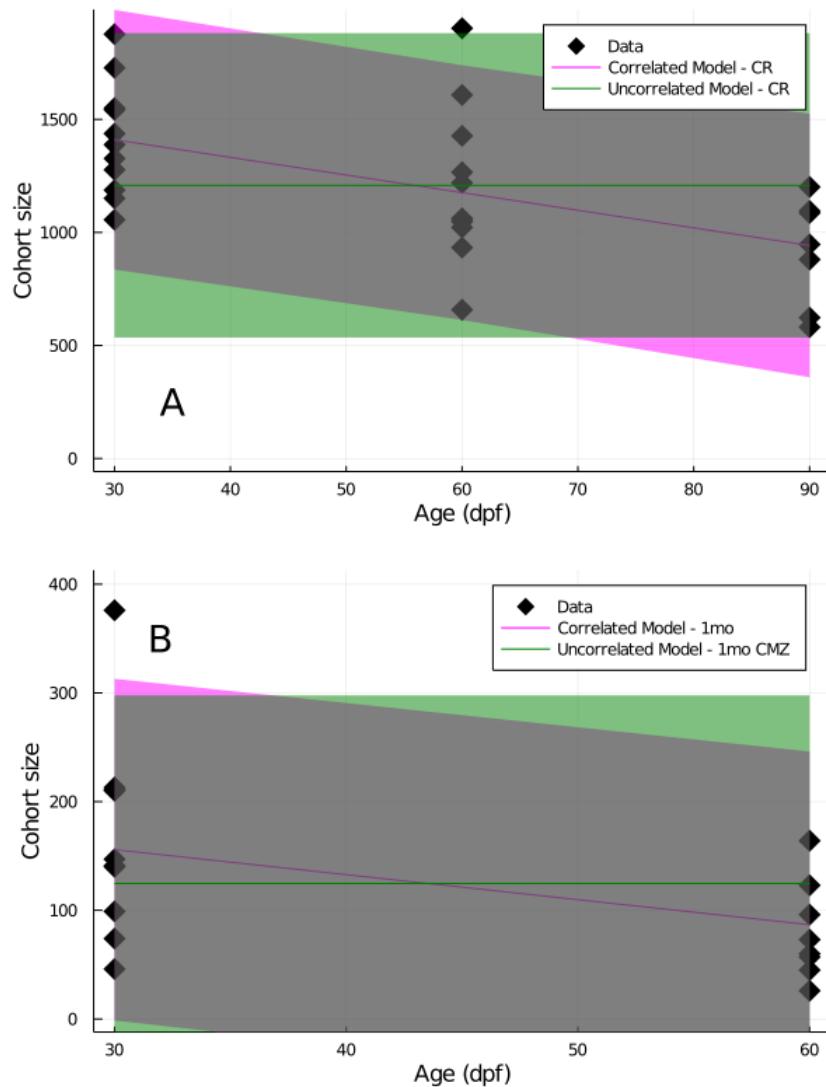


Figure 12.6: Linear regressions of temporally correlated and uncorrelated models of central retinal and 30dpf CMZ-contributed cohorts

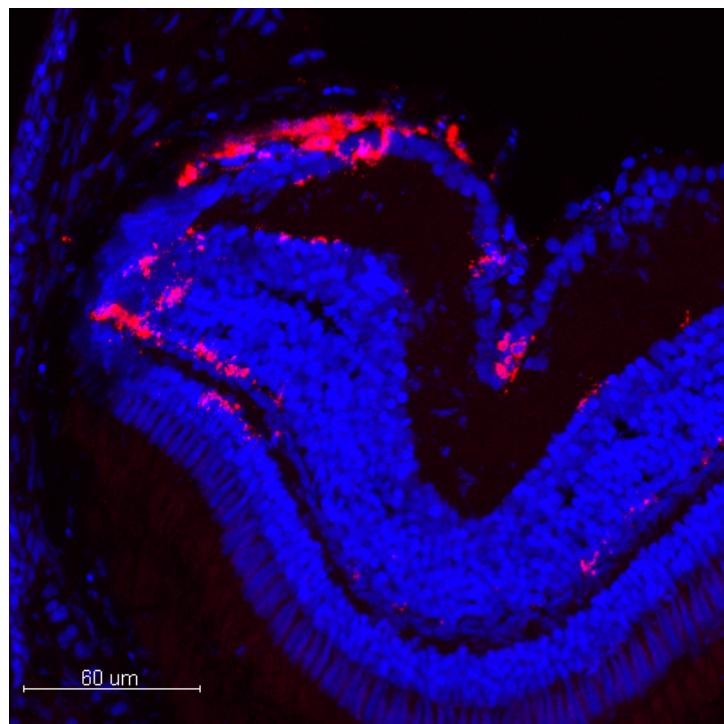


Figure 12.7: **4C4+ microglia are associated with the CMZ**

Representative maximum intensity projection from confocal micrographs of 14 $\mu$ m coronal cryosections through 90dpf zebrafish eyes.

Blue: Hoechst 33258 nuclear counterstain. Red: Microglia labelled with 4C4 antibody. Note extensive presence of 4C4+ microglia around the peripheral CMZ.

## 12.3 Supplementary Tables

Table 12.4: Evidence for Normal vs. Log-Normal models of layer and lineage contribution

Layer	Marker	Cell type	$\mathcal{N}$ logZ	Log- $\mathcal{N}$ logZ	logZR	$\sigma$ sign.
GCL	Cohort	All GCL cells	-65.52 ± 0.66	<b>-30.96 ± 1.0</b>	34.6 ± 1.2	28.9
GCL	Isl2b	RGC	-36.64 ± 0.54	<b>-23.76 ± 0.59</b>	12.88 ± 0.79	16.2
GCL	Pax6	Displaced am.	<b>-40.08 ± 0.71</b>	-40.28 ± 0.46	-0.19 ± 0.85	0.2
GCL	Isl2b/Pax6	RGC subtype	-22.09 ± 0.39	<b>-16.82 ± 0.83</b>	5.27 ± 0.92	5.8
INL	Cohort	All INL cells	<b>-52.65 ± 0.66</b>	-120.1 ± 1.4	-67.5 ± 1.5	44.2
INL	Pax6	Amacrine cell	-23.09 ± 0.39	<b>-9.14 ± 0.46</b>	13.95 ± 0.61	23.0
INL	PKC $\beta$	Bipolar cell	-21.59 ± 0.34	<b>2.81 ± 0.54</b>	24.4 ± 0.64	38.0
INL	GS	Müller glia	-21.45 ± 0.32	<b>10.18 ± 0.6</b>	31.63 ± 0.68	46.4
INL	HM	Horizontal cell	-22.52 ± 0.36	<b>3.02 ± 0.6</b>	25.55 ± 0.7	36.5
ONL	Cohort	All ONL cells	<b>-63.19 ± 0.7</b>	-81.6 ± 1.3	-18.5 ± 1.5	12.3
ONL	Zpr1	Double cones	<b>-31.63 ± 0.6</b>	-26.29 ± 0.6	5.34 ± 0.85	6.3

logZ: logarithm of p(D), the marginal likelihood of the data, or model evidence. Largest evidence values bolded. logZR: evidence ratio; positive values in favour of Log- $\mathcal{N}$  model.

Table 12.5: Evidence for combined vs. separate models of layer and lineage contribution across the dorso-ventral axis

Layer	Marker	Cell type	Combined D-V logZ	Split D-V logZ	logZR	$\sigma$ sign.
GCL	Cohort	All GCL cells	<b>-53.473 ± 0.055</b>	-184.34 ± 0.13	130.86 ± 0.14	911.5
GCL	Isl2b	RGC	<b>-54.58 ± 0.9</b>	-67.72 ± 0.94	13.1 ± 1.3	10.1
GCL	Pax6	Displaced am.	<b>-31.09 ± 0.12</b>	-46.41 ± 0.3	15.32 ± 0.33	46.9
GCL	Isl2b/Pax6	RGC subtype	<b>-43.76 ± 0.76</b>	-109.0 ± 1.3	65.3 ± 1.5	43.3
INL	Cohort	All INL cells	<b>-131.4 ± 1.4</b>	-184.4 ± 1.6	52.9 ± 2.1	25.0
INL	Pax6	Amacrine cell	<b>-15.8 ± 0.024</b>	-52.42 ± 0.69	36.62 ± 0.69	53.3
INL	PKC $\beta$	Bipolar cell	<b>-3.86 ± 0.2</b>	-19.81 ± 0.23	15.96 ± 0.31	51.9
INL	GS	Müller glia	<b>15.78 ± 0.25</b>	28.03 ± 0.41	-12.24 ± 0.48	-25.6
INL	HM	Horizontal cell	<b>10.79 ± 0.41</b>	-3.65 ± 0.19	14.45 ± 0.45	32.2
ONL	Cohort	All ONL cells	<b>-79.87 ± 0.97</b>	-158.2 ± 1.3	78.3 ± 1.6	48.5
ONL	Zpr1	Double cones	<b>-54.66 ± 0.92</b>	-64.27 ± 0.87	9.6 ± 1.3	7.6

logZ: logarithm of p(D), the marginal likelihood of the data, or model evidence. Largest evidence values bolded. logZR: evidence ratio; positive values in favour of combined model.

Layer	Marker	Cell type	$\mathcal{N}$ MLE	Log- $\mathcal{N}$ MLE	lhR
GCL	Cohort	All GCL cells	54.194	<b>55.835</b>	1.642
GCL	Isl2b	RGC	14.439	<b>19.106</b>	4.667
GCL	Pax6	Displaced am.	8.287	<b>11.158</b>	2.871
GCL	Isl2b/Pax6	RGC subtype	20.292	<b>28.139</b>	7.847
INL	Cohort	All INL cells	44.942	<b>45.179</b>	0.237
INL	Pax6	Amacrine cell	<b>23.808</b>	23.178	-0.63
INL	PKC $\beta$	Bipolar cell	<b>27.243</b>	26.119	-1.124
INL	GS	Müller glia	30.847	<b>31.818</b>	0.97
INL	HM	Horizontal cell	27.65	<b>29.431</b>	1.78
ONL	Cohort	All ONL cells	41.488	<b>43.195</b>	1.706
ONL	Zpr1	Double cones	13.256	<b>18.483</b>	5.227

logZ: logarithm of p(D), the marginal likelihood of the data, or model evidence. Largest evidence values bolded. logZR: evidence ratio; positive values in favour of stable model.

# Chapter 13

## Supplementary material for Chapter 5

### 13.1 Materials and methods

#### 13.1.1 Zebrafish husbandry

Zebrafish husbandry was performed as described in Section 11.1.1.

#### 13.1.2 Generation of transgenic *vsx2:eGFP rys* line

Figure 5.9 makes use of coronal cryosections of *Tg(vsx2:eGFP) rys* animals. Transgenesis was performed using reagents from the Tol2kit [KFG<sup>+</sup>07], which uses the Invitrogen Gateway plasmid construction system to assemble three-part constructs from donor plasmids. Briefly, the Gateway system uses BP clonase to introduce fragments of interest into sets of 3 entry vectors, which are recombined in predictable order by LR clonase. Tol2kit consists of a numbered, standardized set of preconstructed entry vectors intended for use in the construction of transposable Tol2 transgenesis vectors.

The construct of interest was built from a ~1.1kb fragment of the 5' promoter sequence of *vsx2* in the 5' entry vector, an eGFP expression cassette in the middle entry vector (Tol2kit #383), and a polyadenylation signal in the 3' entry vector (Tol2kit #302), recombined into pdestTol2pA2 (Tol2kit #394).

The *vsx2* promoter fragment was isolated by PCR amplification from the BAC (). BP clonase was bypassed in favour of restriction endonuclease sites inserted into the fragment's primers; this was necessitated by the instability of the ccd cassette in the BP vectors. The promoter fragment was amplified with a XhoI site 5' and a AgeI site 3'; the Tol2kit 5' Multiple Cloning Site entry vector (#228) was digested with SalI and XmaI in order to produce compatible sites for ligation. All restriction enzymes and the Quick Ligation kit were from NEB.

Injections into 1-2 cell embryos were conducted by standard techniques [Wes00]. Animals with stable expression of *vsx2* in postembryonic peripheral RPCs and in central retinal bipolar cells, as expected, were identified and bred out. These animals were subsequently introgressed into *rys* in order to produce *Tg(vsx2:eGFP) rys* animals.

### 13.1.3 Morpholino injections

Morpholino injections were performed by Maria Augusta Sartori by methods previously described [WPMT15].

### 13.1.4 Morphological photography

The photographs presented in Figure 5.1 were obtained with the transmitted light optics of a Zeiss fluorescent stereoscope (SteREO Lunar.V12), using ZEN Imaging software.

### 13.1.5 PCNA and EdU proliferative histochemistry

PCNA and EdU proliferative histochemistry giving rise to the data presented in Figure 5.3 was performed as described in Section 11.1.2.1 and Section 11.1.2.2, except that the length of the EdU pulse was 24 hours, and was not administered to 5dpf animals, which were set aside for the nuclear morphology study presented in Figure 5.6.

### 13.1.6 BrdU pulse-chase assay

The BrdU pulse-chase assay presented in Figure 5.4 was conducted by soaking 3dpf *rys* larvae in 10mmol BrdU for 8 hours. After a subsequent 7 day chase period, the 10dpf larvae were sacrificed and processed as described in Section 11.1.2.3.

### 13.1.7 Cumulative EdU labelling assay

The cumulative EdU labelling experiment which was used to estimate cell cycle parameters in Figure 5.5 was conducted as described in Section 11.1.2.2.

### 13.1.8 Analysis of *rys* nuclear parameters by Galilean Monte Carlo Nested Sampling

The estimation of evidence for separate and combined mutant and sibling models of nuclear parameters presented in Table 5.1 was produced using the Julia package GMC\_NS.jl, presented in Chapter 7.

### 13.1.9 Progenitor identity marker immunohistochemistry

Pax6 immunohistochemistry was performed as described in Section 12.1.4. In situ hybridization for npat transcript was performed by Monica Dixon by standard methods [Wes00].

### 13.1.10 Caspase-3 immunohistochemistry

Caspase-3 immunohistochemistry was performed by Monica Dixon, using histochemical methods described in Section 12.1.4, substituting the appropriate anti-caspase-3 primary antibody.

### 13.1.11 Electron microscopy

Images presented in Figure 5.7 were produced from ultrathin sections for transmission electron microscopy, prepared as described previously [LDT12] by Monica Dixon and Audrey Darabie. These

sections were imaged on a Hitachi H-7000 transmission electron microscope using AMT Image Capture Engine software.

### 13.1.12 qPCR analysis of *rys npat* expression

The RT-PCR analysis of *rys npat* expression presented in Figure 13.6 was conducted by Monica Dixon by standard methods [SFM89].

### 13.1.13 qPCR analysis of *rys histone* expression

The qPCR analysis of *rys histone* expression presented in Figure 5.13 was conducted by Maria Augusta Sartori. Briefly, total mRNA was extracted from groups of sib and *rys* embryos at the appropriate age. Two separate pools of cDNAs were generated from the isolated mRNAs, primed with random hexamers and oligo-dTs. These cDNA pools were used as the templates for the amplification of total histone transcript and polyadenylated transcript, respectively. This was done using primers degenerate for the named class of histone; individual transcripts are not identified. A Bayesian analysis of these data was performed in Section 17.8.23.

### 13.1.14 *rys* MNase digestions and NGS

Two 10-animal pools each of 5dpf *rys* sib and *rys* animals had fresh nucleosome-protected MNase fragments produced using the Takara Biotech micrococcal nuclease kit. These digests were submitted to the SickKids TCAG facility for Illumina HiSeq 2500 NGS, with two lanes run per pool.

### 13.1.15 Nucleosome position calling

The nucleosome position calling pipeline used in Chapter 5 was automated using the Java machine learning framework KNIME [DB16] workspace incorporating KNIME4NGS nodes [HJH<sup>+</sup>17]. Node settings are largely default and are available in the `knime_workspace` folder of the thesis archive.

Briefly, FastQC (v0.11.8) [And18] was used to characterise the quality of the fastq sequence files describing the nucleosome-protected fragment pool from the MNase digestions described above. No sequence from any of the two duplicates of the two pools was rejected due to low quality. Sequences were found to be of uniformly high quality, with low adapter content. These results are available in the `fastqc` thesis git archive. TrimGalore (v0.5.0) [And18] was used to remove trailing adapter bases, using Adapter sequence: 'AGATCGGAAGAGGC' (Illumina TruSeq, Sanger iPCR). The trim reports are available in the thesis git archive. Bowtie 2 [LS12] with default settings was used to map reads to the zebrafish genome (GRCz11). Nucleosome positions were called from the resulting sequence alignment map (SAM) files by using DANPOS2 (v2.2.2) [CXP<sup>+</sup>13]. As DANPOS2 is no longer supported, and the code contained errors preventing execution, a fixed version of this code has been uploaded to <https://github.com/mmattocks/DANPOS2fix> and filed in the thesis HDD archive, along with the called output positions.

The DANPOS nodes of the KNIME workflow are python nodes which execute DANPOS2 itself, as well as the nucleosome position analyses described below.

### 13.1.16 Analyses of *rys* nucleosome position disposition

Figure 5.16 was produced from the DANPOS2-called positions for the pooled *rys* sibling and mutant MNase-protected fragment datasets. For each numbered *D. rerio* chromosomal scaffold (1-25), as well as for scaffolds not assigned to any chromosome (NC), the number of sibling positions was divided by the total length of the scaffold. These values were compared to the naively expected number of positions per scaffold, determined by dividing the total number of positions per kilobase of genomic material, then multiplying the particular scaffold length. The per-chromosome relative number of positions, as well as fold-differences from the naive expected value, are displayed in pie chart format in Panel A. The Panel C chart displays the result of normalizing these position values by relative occupancy, as determined by DANPOS2.

#### 13.1.16.1 Background Hidden Markov modelling of *D. rerio* genomic sequence emission

The Julia package `BioBackgroundModels.jl` was used to perform background Hidden Markov Model selection on samples of *D. rerio* genomic material, in order to serve as a model of genomic noise for subsequent Independent Component Analysis (ICA) modelling of nucleosome positions, described below. The general strategy pursued was to first train a zoo of 3 replicates each of all HMMs with 1-6 states, emitting 0<sup>th</sup>-2<sup>nd</sup> order DNA kmers, on each of three broad partitions of the GRCz11 zebrafish genome, as described in Chapter 9, and initializing the zoo's EM chains using the default `autotransition_init` function, which samples from an uninformative prior on state emission vectors, but a transition matrix prior heavily favouring state autotransition. This zoo was converged to an EM step likelihood delta of 1e-3 on 2e6 bp sampled without replacement from each of these partitions, then evaluated by testing against a further 2e6 bp sampled as a test set. The best model triplicate for each partition, in every case the 6 state, 0th order triplicate, was selected for further refinement by EM training on a new 8e6 bp sampled from the appropriate partitions. After this refinement process was complete, the triplicate chains were inspected using `BioBackgroundModels.jl`'s reports. The convergence of the triplicate on the same region of the parameter space, and the stationarity of the chains were verified before selecting the most likely models of the refined triplicate to represent *D. rerio* genomic background noise in the `BioMotifInference.jl` ICA models.

The code to sample GRCz11 for the initial zoo training task is available in Section 17.8.28, for the refinement task in Section 17.8.27. EM execution code is found in Section 17.8.29 and Section 17.8.31. Analysis code is given in Section 17.8.30 and Section 17.8.26.

### 13.1.17 Evidence and maximum a posteriori estimation of ICA models of *rys* nucleosome position sequence emission

The Julia package `BioMotifInference.jl`, presented in Chapter 10 was used to perform evidence and MAP estimation of Independent Component Analysis PWM source models of nucleosome position emission. The ICA models used had 8 PWM sources ranging from 3 to 10 positions in length. Model ensembles were initialized by sampling from uninformative source priors and an mixing prior of .07 (ie. for each source, mean 7% of observations will begin mixed with that source).

Observation sets comprised samples of 7092 nucleosome positions from each of the *rys* mutant and sibling differential position sets, corresponding to approximately 1 Mbp for each of the mutant and sibling sets, and 2 Mbp for the combined set.

Ensembles were converged to within 125 orders of magnitude likelihood compression.

The code used to produce the differential position set samples is available in [Section 17.8.34](#). Ensembles are assembled in [Section 17.8.32](#), and converged in [Section 17.8.33](#).

## 13.2 Supplementary figures

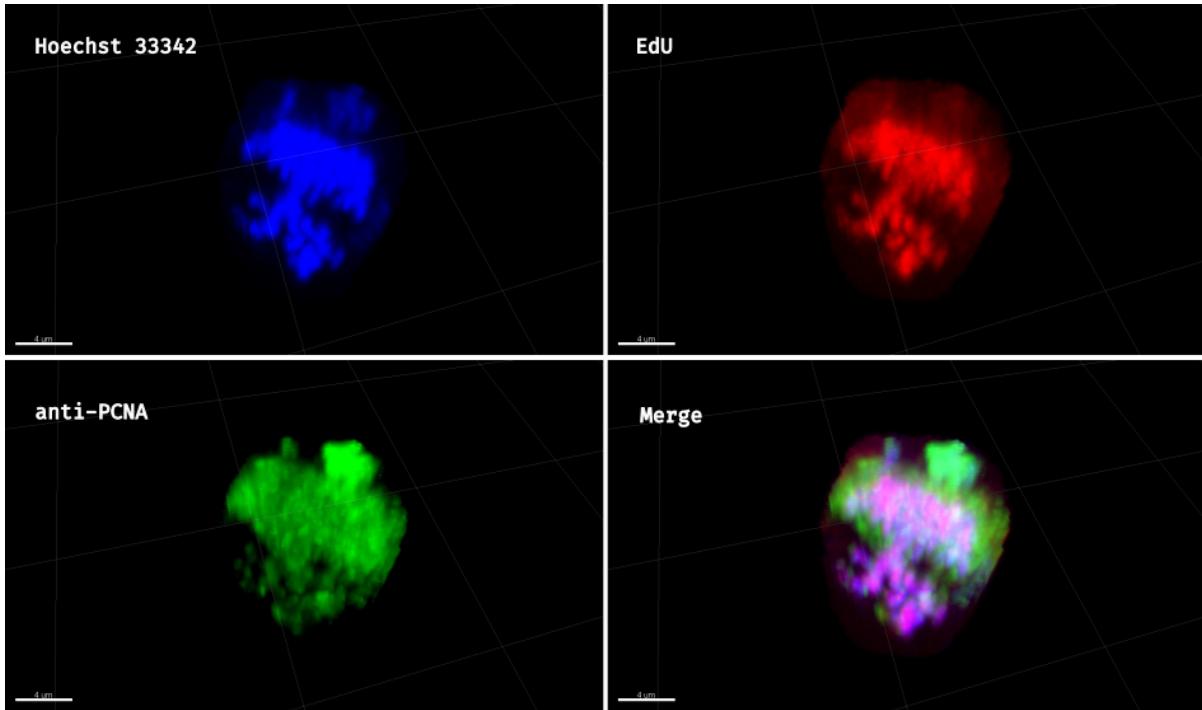
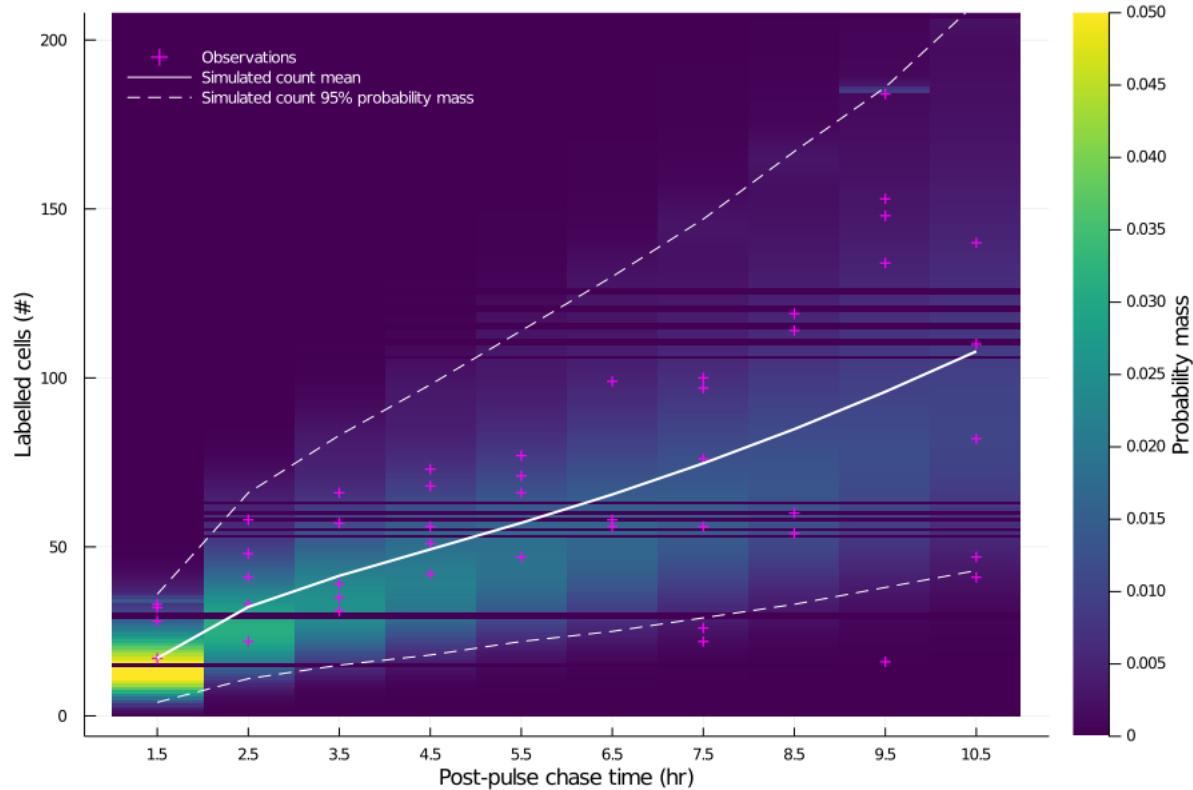


Figure 13.1: 10dpf mutant *rys* RPCs are mitotic

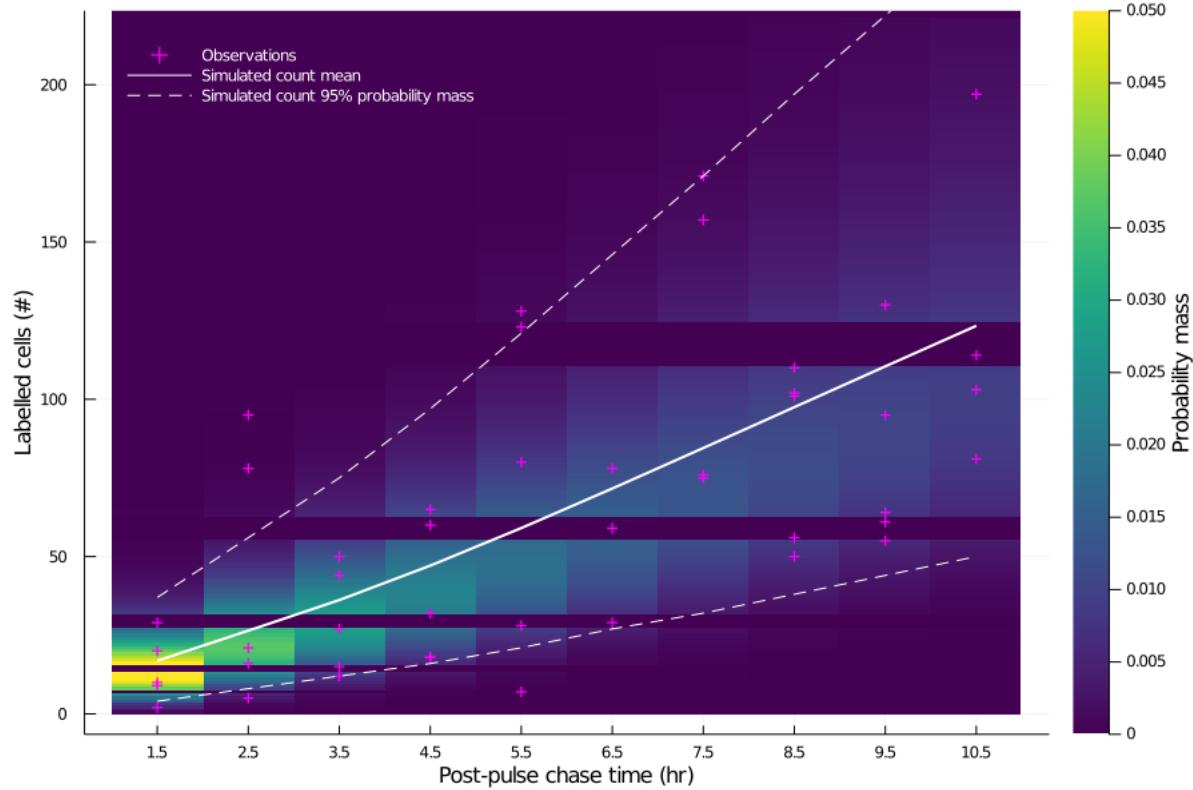
3d volume reconstruction of a complete mitotic figure in a *rys* mutant CMZ RPC at 10dpf, from a confocal micrograph of a 14 $\mu$ m coronal cryosection. Blue channel: Hoechst 33258; Green channel: PCNA; Red channel: EdU. Scale bars: 4  $\mu$ m.



**Figure 13.2: MAP model output and observations for *rys* sibling thymidine slice model of 5dpf cumulative EdU labelling**

Count of EdU-positive cells observed in 14 $\mu$ m coronal cryosections through *rys* mutant CMZs at indicated chase times during a 10 mM EdU pulse (magenta crosses), overlaid with MAP model output. Probability mass distribution of the discrete non-parametric output is shown by color scale; yellow values include counts with >.05 mass. Output mean and 95% mass are indicated by solid and dashed white lines.

Methods in Section 13.1.7. Code in Section 17.8.16.



**Figure 13.3: MAP model output and observations for *rys* mutant thymidine slice model of 5dpf cumulative EdU labelling**

Count of EdU-positive cells observed in 14 $\mu$ m coronal cryosections through *rys* mutant CMZs at indicated chase times during a 10 mM EdU pulse (magenta crosses), overlaid with MAP model output. Probability mass distribution of the discrete non-parametric output is shown by color scale; yellow values include counts with >.05 mass. Output mean and 95% mass are indicated by solid and dashed white lines.

Methods in Section 13.1.7. Code in Section 17.8.16.

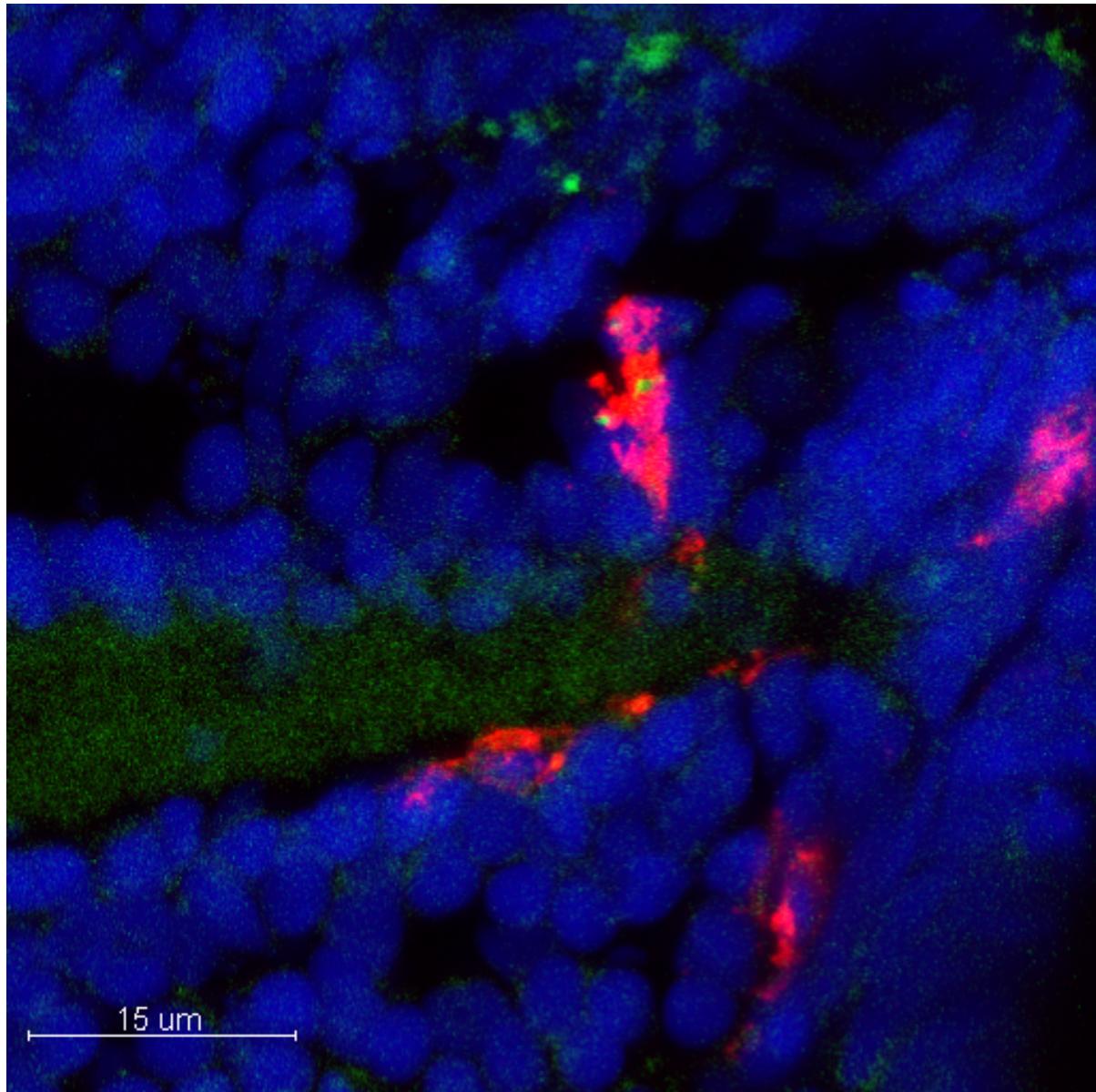
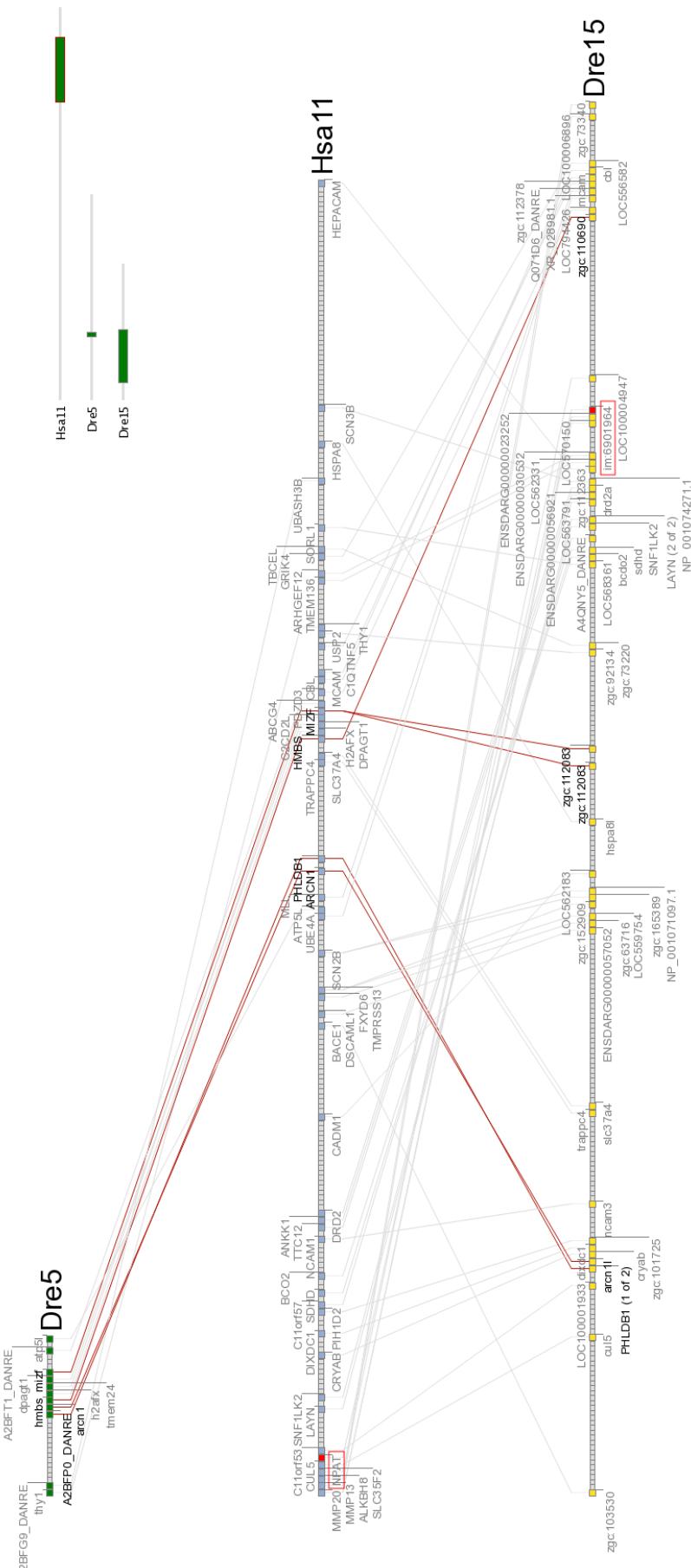


Figure 13.4: **4C4-positive microglia engulf rys mutant RPCs**

Maximum intensity projection of *rys* mutant retina at 5dpf, from a confocal micrograph of a 14 $\mu$ m coronal cryosection. Labelled in red, a 4C4-positive microglia engulfs TUNEL-positive apoptotic fragments in the vicinity of the CMZ. Blue channel: Hoechst 33258; Green channel: TUNEL; Red channel: 4C4.



**Figure 13.5: Synteny Database output for the syntenic region containing *D. rerio* npat**

Dre#: *Danio rerio* chromosome # Hsa#: *Homo sapiens* chromosome #

The relative position of the displayed genomic neighbourhoods on their chromosome scaffolds is displayed inset, top right. *D. rerio* npat is highlighted with a red square, visible on the right of the selected Dre15 region, annotated “im:6901964”. *H. sapiens* NPAT is highlighted similarly on the left of Hsa11. Landmarks of the duplication and rearrangement event that brackets the position of npat on Dre15 are highlighted with red lines.

On the scaffold of *H. sapiens* chromosome 11, NPAT occurs upstream of the highlighted duplication cluster bracketed by ARCN1 and MIZF. Zebrafish npat is located in the midst of this rearranged, duplicated cluster, but does not appear to have been duplicated itself; at least if it was, no parologue can be found by synteny or similarity analysis.

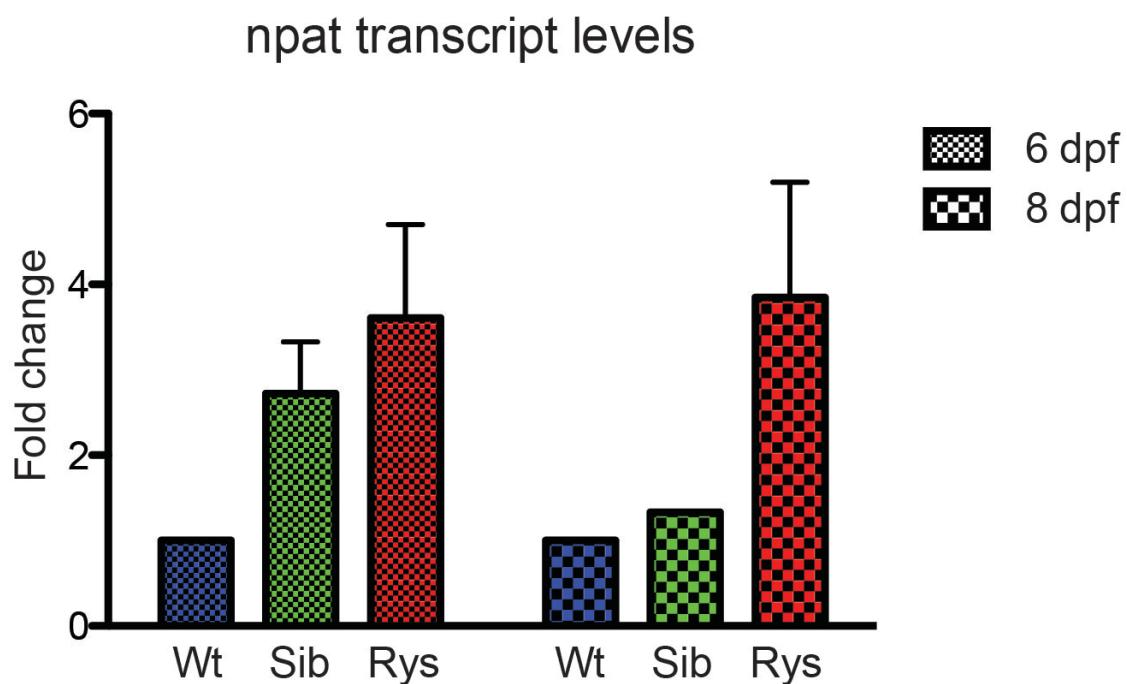


Figure 13.6: **npat is overexpressed in *rys***  
qPCR data for npat transcript amplified from pools of *rys* mutant (red), sibling (green), and wild-type control (blue), at 6dpf (left) and 8dpf of age (right). Error bars indicate standard deviation.

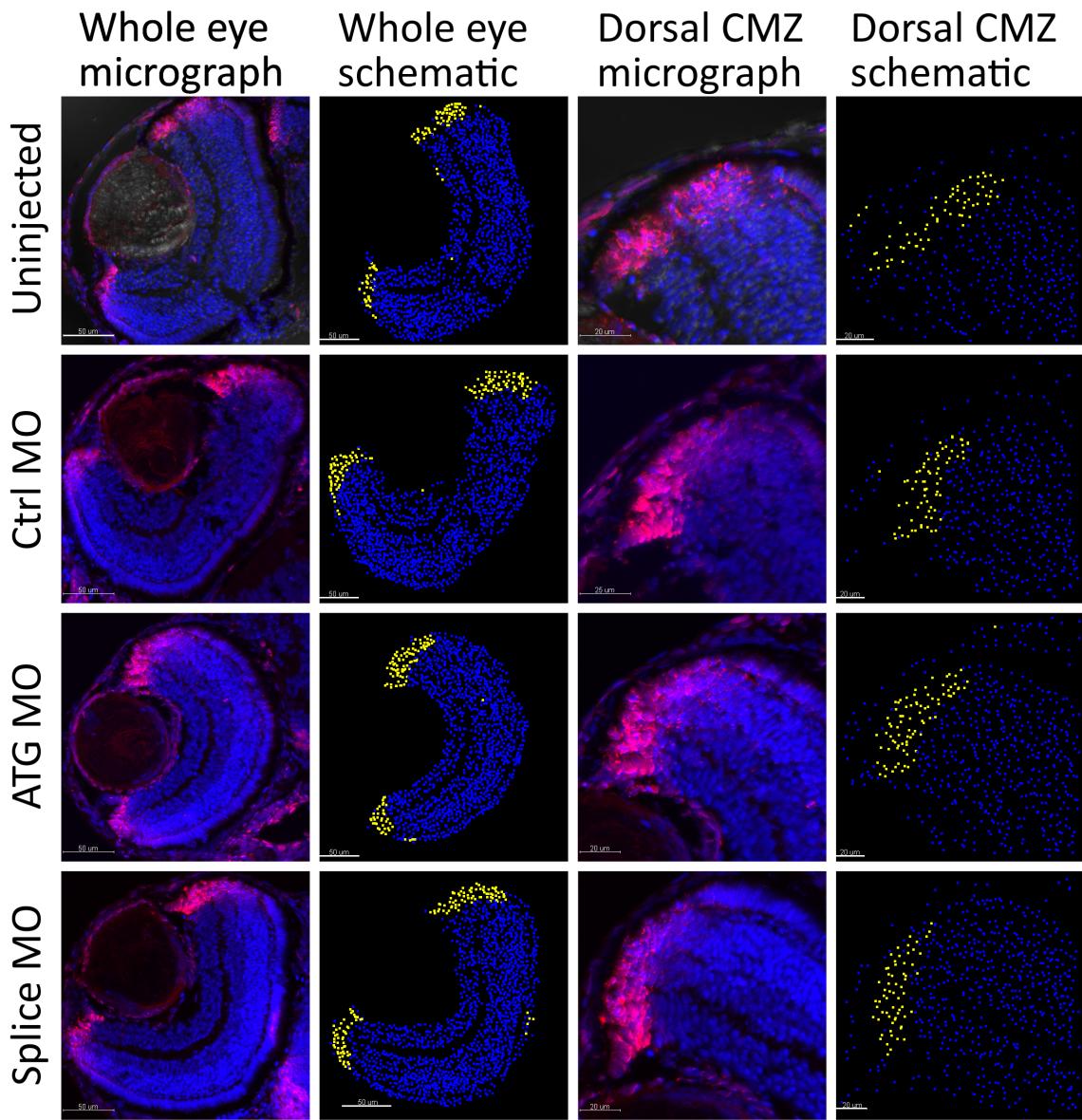


Figure 13.7: *npat* morpholino injectants do not recapitulate *rys* retinal disorganisation phenotype

Representative 14 µm central coronal cryosections through eyes of zebrafish embryos injected at the 1-2 cell stage with control, ATG- and splice-directed *npat* morpholinos, alongside uninjected siblings. Maximum intensity projection micrographs, with red staining of PCNA+ve nuclei against a blue Hoechst 33342 counterstain, are presented alongside dot schematics, displaying the positions of nuclei in blue, with PCNA+ve CMZ nuclei highlighted in yellow.

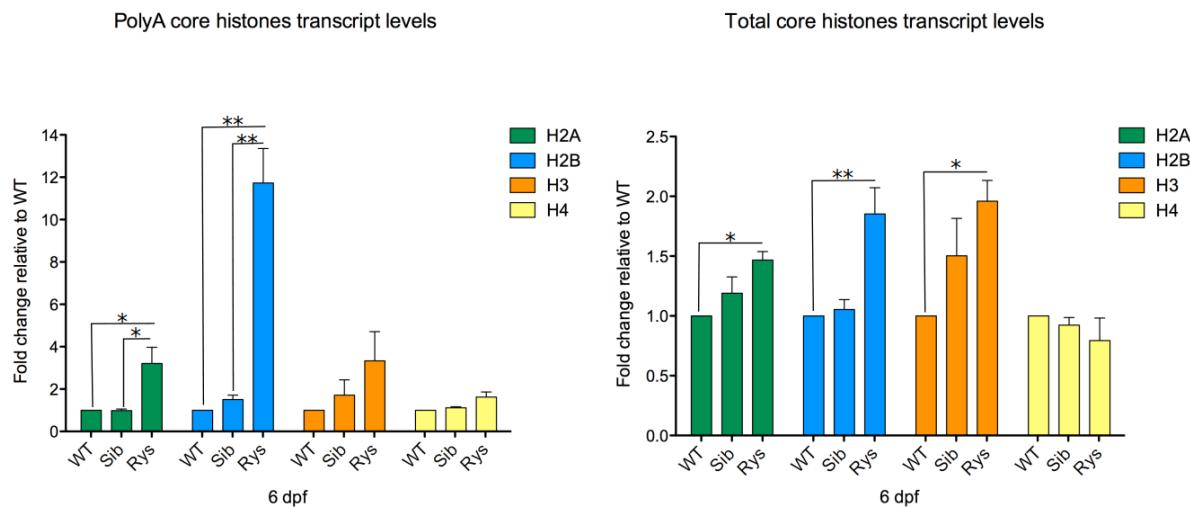


Figure 13.8: **Morpholinos directed to npat result in histone transcript overexpression similar to rys**

qPCR analysis of fold changes in histone family transcript expression, both polyadenylated (left panel), and total (right panel), after injection at the one-cell stage with morpholino directed to the npat start codon.

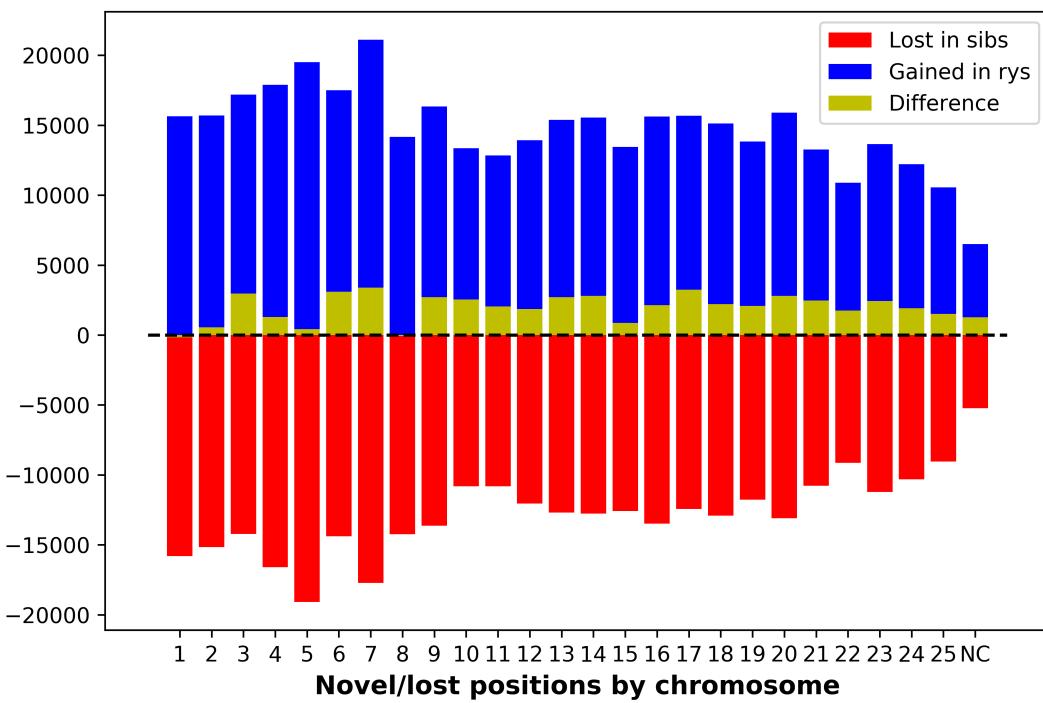
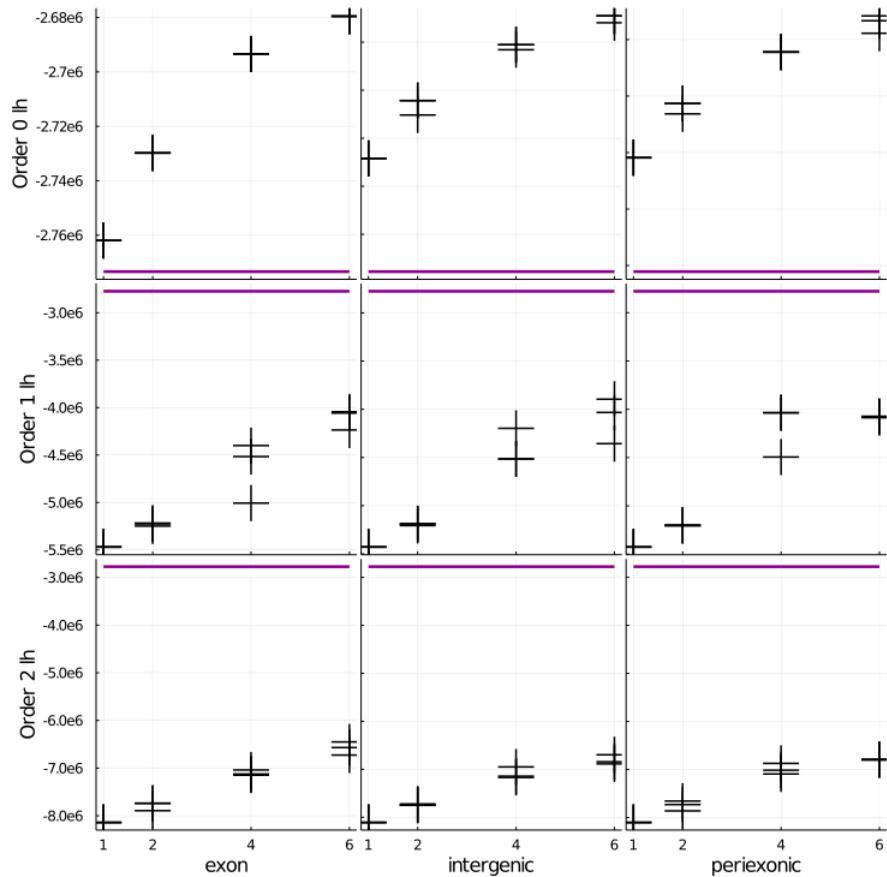


Figure 13.9: **Novel nucleosome positions in *rys* occur in similar numbers to those lost from sibs.**

Counts of positions found in sib but not *rys* (red bars, represented as negative numbers, as these are ‘lost’ in *rys*) and those found in *rys* but not sib (blue bars, ‘gained’). The magnitude of the difference between the counts is represented with a yellow bar.



**Figure 13.10: Test likelihoods for background HMMs trained on *D. rerio* genome partitions**  
 Panels are gridded by genomic partition (left-to-right, exon, intergenic, and periexonic sequences) and emission symbol order (top-to-bottom, 0th order, 1st order, 2nd order). Within each panel, increasing state number is represented on the x axis, while test likelihood is on the y axis. Each cross gives the likelihood of one of the three replicates optimised by EM for each order/state/partition combination. Magenta line indicates the likelihood of a naive model (1-state, equiprobable emission of each base). Only 0th order models prove to be more plausible than this naive control.

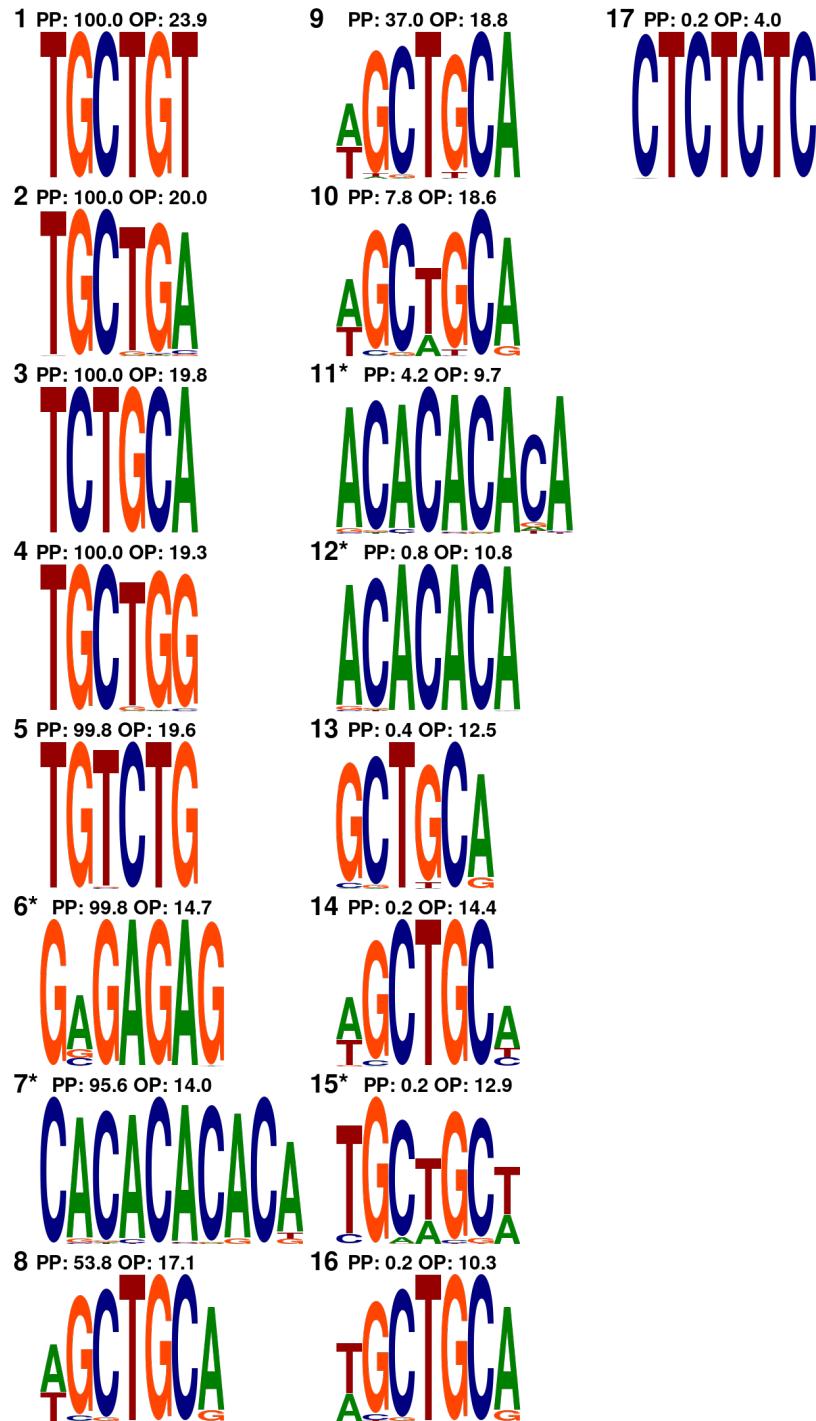


Figure 13.11: **PWM sources detected in combined sib and rys differential sources.**  
PWMs presented as sequence logos with letter height proportional to position informational contentemission probability.

PP: Posterior prevalence; proportion of models within the compressed maximum a posteriori ensemble which have this PWM as a signal source

OP: Observation prevalence; mean proportion of observed position sequences this sequence is used to explain, over all posterior samples

### 13.3 Supplementary tables

Table 13.1: Evidence for separate models of retinal parameters in anti-npat morpholino-injected animals

Morpholino	Measurement	Separate logZ	Combined logZ	logZR	$\sigma$ sign.
ATG	CMZ pop.	-236.4 $\pm$ 2.3	<b>-198.3 <math>\pm</math> 2.2</b>	-38.1 $\pm$ 3.2	12.1
ATG	Total sectional pop.	-412.4 $\pm$ 7.8	<b>-225.3 <math>\pm</math> 4.9</b>	-187.1 $\pm$ 9.2	20.3
ATG	Sectional pop. per CMZ cell	<b>-70.18 <math>\pm</math> 0.86</b>	-78.0 $\pm$ 0.86	7.8 $\pm$ 1.2	6.4
ATG	Dorsal CMZ pop.	-169.5 $\pm$ 1.4	<b>-151.75 <math>\pm</math> 0.8</b>	-17.8 $\pm$ 1.6	11.2
ATG	Ventral CMZ pop.	-223.1 $\pm$ 1.5	<b>-165.3 <math>\pm</math> 1.0</b>	-57.8 $\pm$ 1.8	32.5
ATG	Nuclear volume	-290.6 $\pm$ 1.8	<b>-192.7 <math>\pm</math> 1.8</b>	-97.9 $\pm$ 2.5	39.6
ATG	Nuclear sphericity	<b>-186.13 <math>\pm</math> 0.79</b>	-215.66 $\pm$ 0.57	29.53 $\pm$ 0.97	30.3
Spl	CMZ pop.	-272.6 $\pm$ 2.4	<b>-197.0 <math>\pm</math> 2.6</b>	-75.7 $\pm$ 3.6	21.2
Spl	Total sectional pop.	-455.9 $\pm$ 8.3	<b>-205.6 <math>\pm</math> 4.7</b>	-250.3 $\pm$ 9.5	26.3
Spl	Sectional pop. per CMZ cell	-87.04 $\pm$ 0.17	<b>-66.22 <math>\pm</math> 0.29</b>	-20.82 $\pm$ 0.34	62.0
Spl	Dorsal CMZ pop.	-226.2 $\pm$ 1.7	<b>-171.0 <math>\pm</math> 1.1</b>	-55.2 $\pm$ 2.0	27.7
Spl	Ventral CMZ pop.	-234.97 $\pm$ 0.81	<b>-222.3 <math>\pm</math> 1.7</b>	-12.7 $\pm$ 1.9	6.8
Spl	Nuclear volume	-277.5 $\pm$ 1.6	<b>-223.2 <math>\pm</math> 2.1</b>	-54.3 $\pm$ 2.7	20.4
Spl	Nuclear sphericity	-94.3 $\pm$ 1.1	<b>-67.71 <math>\pm</math> 0.39</b>	-26.6 $\pm$ 1.1	23.2

logZ: logarithm of  $p(D)$ , the marginal likelihood of the data, or model evidence. Largest evidence values bolded. logZR: evidence ratio; positive values in favour of stable model.

Table 13.2: *rys* mutant differential nucleosome positions are less typical of the genome than those mapped to sibling positions

Sequence set	<i>rys</i> naive LH	sib naive LH	LH ratio
Differential	-6.3402e7	-5.271e7	-1.0692e7
Mapped	-2.2921e8	-2.2617e8	-3.0407e6

naive LH: log-Likelihood of sequences, given genomic background noise models alone. Higher values are more likely a priori, indicating sequences more typical of the genome. LH ratio given with *rys* mutant sequences as the denominator. Note that the ratio of the differential sequences is much greater than that of the mapped sequences, indicating that the *rys* differential position set is much less typical of the genome than mutant sequences which are found overlapping sibling positions. Code in Section 17.8.36.

# Chapter 14

## Supplementary materials for all chapters

### 14.1 Thesis git and HDD archives

The resources used to compile and typeset this document are available at <https://github.com/mmattocks/Thesis>. These include most observational datasets referred to in figures. Most of the code used to generate figures, available in Section 17.8, can be executed without additional resources, beyond the package dependencies included by the script. These are easily installed in Julia, as one may `]add` packages from the interpreter. The scripts may then be `include()`d, with appropriate changes to paths where required (normally declared at the top of the script). Not included in the git archive are raw confocal microscopy stacks from any chapter, nor the bioinformatic data from Chapter 5), due to the large size of these datasets. The fully calculated ensemble posterior samples are also not included, although they can be replicated from the analysis scripts. These larger datasets are available in the HDD data archive, available in the Tropepe lab at the Department of Cell and Systems Biology, kept with Dr. Vince Tropepe. The git archive will host the most up-to-date and accurate version of the thesis.

### 14.2 Code notes

Most of the code used in this thesis is presented in Chapter 17. Three languages are represented: C++ for the CHASTE simulators presented in Chapter 2, Python for the SPSA optimization algorithm and general scripting in Chapter 2, as well as for the generation of Figure 5.16 in Chapter 5, and Julia for everything else. The Julia code is by far the easiest to execute. We have found it to be a productive, performant, and educational language to code, and the lack of a well-developed cell-based simulation system like CHASTE is the primary reason the thesis is not entirely Julian.

The C++ code requires the CHASTE framework [MAB<sup>+</sup>13] to compile. If replication of the simulation pipeline in Chapter 2 is required, CHASTE can be downloaded from <https://www.cs.ox.ac.uk/chaste/index.html> and used to compile the simulators in the SMME/apps folder of the SMME repo, available in Section 17.1. The simulators are managed from the python script fixtures available in SMME/python\_fixtures. Python used for the SMME work is python3.

We also used the Java-based KNIME scripting framework to manage the execution of the bioinformatics pipeline for calling nucleosome positions in Chapter 5. This made use of the KNIME4NGS package [HJH<sup>+</sup>17]. The KNIME workspace used is available in the HDD archive at `/knime_workspace`. This workspace includes a THiCweed analysis pipeline [ASNS17], which is not presented in this thesis, and can be deleted or ignored. The pipeline requires both python2 and python3 environments, because danpos, the nucleosome position calling algorithm we used, is written in python2. danpos [CXP<sup>+</sup>13] is no longer available in functioning form from the author's archive. We have uploaded a version that fixes execution errors to <https://github.com/mmattocks/danpos-fixed>.

## 14.3 Computational cluster description and discussion

The computational cost of the model sampling performed in this thesis varies widely. The ICA models estimated in the *rys* work of Chapter 5 are comparatively enormous problems with expensive solutions, and require more power to solve in a reasonable timeframe than is typically available in a molecular biology lab. On the other hand, the CMZ slice models used in Chapter 4 can be estimated in a day on typical higher-end PCs used for microscopy acquisition and the like. This thesis used two machines as the local cluster for the non-ICA problems. None of the problems made use of GPU processing, although some of them could possibly benefit from the substitution of CUDA arrays.

The first machine was an obsolete watercooled 4-core i5-4670K overclocked to 4.4 GHz, with 16 GB RAM. It has good thermal characteristics and maintains full clock under 100% load at 60°C. This machine is adequate for estimating most of the `CMZNicheSims.jl` models, on the datasets of this thesis, in less than a day. The CHASTE/SPSA approach presented in Chapter 2 is slow; it takes 7-10 days to execute the required 15000 iterates on this machine. The comparable model in `CMZNicheSims.jl` is the `Thymidine_Model`, which is the slowest of the simulators presented in Chapter 8. Generally, this machine will complete about 1500 iterates a day using this purpose-coded model in Julia. Direct comparison between SPSA and GMC estimation processes is difficult, but this suggests that the overhead from using the object-oriented spatial modelling framework CHASTE does not dominate computational cost. Still, given Julia's C++-equivalent performance and much higher coding productivity, it seems the purpose-built, functionally-coded model will usually be faster and cheaper to develop and execute, overall. In any case, for the types of relatively simple observational datasets collected from microscopy studies, lab machines will suffice; for larger bioinformatic problems like those tackled in Chapter 5, this machine is best used as a combined master and fast calculation node in a cluster of 50+ dedicated calculation nodes. A particular limitation of older machines is the relatively low cap on the size of available RAM DIMMs. Generally, this prevents maximum parallelisation of `BioBackgroundModels.jl` models across cores. Parallelisation is easier to achieve on more modern machines with more cores/threads and RAM.

The second machine was a more modern 12-threaded i7-9750 in a Tongfang mobile chassis, clocked to 4.0 GHz, with 32 GB RAM. This machine is strongly thermally limited<sup>1</sup>, typically running at 3.4GHz and 90°C under full load. Still, the additional threads and RAM make higher-end laptops a useful calculation node. Machines like this can expect to estimate two or more of the `CMZNicheSims.jl` models simultaneously in less than a day, given similarly sized datasets. They can expect to supply about three 4-threaded workers for distributed `BioBackgroundModels.jl` or `BioMotifInference.jl` jobs,

---

<sup>1</sup>This machine cooked its own motherboard during the thesis work, requiring replacement with a revised board less vulnerable to overheating. This can be partially overcome in Canadian winters by calculating outside.

with significantly better performance than the AWS nodes described below.

Both BBM.jl and BMI.jl assume that the user will want to distribute the job across a cluster of dissimilar workers. Because Julia has an extremely elegant Distributed package as part of its Base distribution, any type of julia **ClusterManager** with master-worker topology can be used. For this thesis, we chose to purchase compute time on the spot market at Amazon Web Services (AWS). A small module with helper functions for purchasing node time on this market this is supplied in [Section 17.7](#). The AMI image used in this thesis is available in the HDD archive. We used **c5a.24xlarge** nodes. These have 48 cores per socket, clocked at 2.8GHz, and 196 GB of RAM. Most AWS instances are slower than enthusiast hardware, but the sheer number of available cores and RAM makes them an effective way to quickly assemble a large worker pool. We usually used both BBM.jl and BMI.jl with 4 cores per worker, as the threading overhead decreases efficiency beyond this point. Some CNS.jl simulations can usefully use 24-36 threads, but these should be benchmarked. Thymidine\_Ensembles do noticeably poorly with large thread numbers, for instance. The model comparison performed in [Chapter 5](#) required five **c5a.24xlarge** instances, added to the two machines described above. These ran for approximately two weeks (half a week for each of the sib and *rys* observations sets and a further week for the combined set). Typical spot prices for these nodes are about USD  $1.\text{hr}^{-1}$  at the time of writing, giving an approximate cost of USD 1700 for a problem of this size. On-demand pricing is typically 4-5 times this cost; obviously, it is preferable to use the spot market.

This presents the risk that nodes may be stopped at any time, either because the spot price exceeds the submitted maximum bid, or because AWS requires the compute power for the on-demand market<sup>2</sup>. As a result, all of Julia algorithms available in this thesis back up to disk and are robust to dropped workers, power outages, and other kinds of interruption. To resume an interrupted calculation performed from one of the Julia analysis scripts in this thesis, simply execute it again.

Users of BBM.jl and BMI.jl clusters should be aware that observations data transfer to workers can take a very long time, particularly with the relatively slow connections provided by AWS. This presents an additional complexity to budgeting for the spot market: interruptions require the re-transfer of observations to each worker in turn. BMI.jl waits for observations to finish transferring to each worker before starting the next; this prevents the situation where attempting to transfer to all nodes simultaneously on the same connection prevents any of them from starting on the problem. Probably, for larger datasets and larger models, the master node should be one of the cloud devices, to allow the observations to be local to those machines.

---

<sup>2</sup>High Netflix viewing times seem to be associated with this.

# Chapter 15

## Theoretical Appendix A: Model theory and statistical methods

### 15.1 Model Theory

This thesis is concerned with a particular kind of scientific model: those that are subject to mathematical analysis, and tractable subjects for numerical simulation techniques. Most biological models do not fit this description, but increasing computing power has expanded the potential for, in particular, cellular and macromolecular explanations to be formalized and tested in this way. While doing so is sometimes derided as “physics envy”, the reality is more complex. Simple heuristics are, sometimes, adequate biological explanations; much of differential diagnosis consists of reliable descriptive relations between qualitatively characterised signs in patients, and the presence of a pathogen, for instance. More often, the complexity of biological systems, and the sophistication of the instruments with which we probe them, give rise to observations which support more than one plausible causal explanation for the features of the data. The question of model comparison then immediately arises. Without numerical analysis, we may only make reference to the use of the model in applied domains where the model failure has commercial consequences. Because biological models are rarely required to succeed as engineering tools, for the prediction and control of outcomes of practical significance, an inability to analyse them numerically means that we are left only with rhetoric to assess their relative explanatory value. This approach to “model comparison” necessarily devolves into the anarchic Feyerabendian discursive chaos discussed in Section 16.2.1. Feyerabend correctly points out that scientific models can only be compared against one another, and that the selection of comparative criteria can never be “objective” in the sense of being fully independent of the contingent situation and goals of the observer. Nothing can be done about this; either we look for relatively-better criteria and relatively-better models as judged by those criteria, or we remain in the realm of pure metaphysical rhetoric, without investigating their implications, what Feyerabend describes as the realm of the “crank”:

[T]he distinction between the crank and the respectable thinker lies in the research that is done once a certain point of view is adopted. The crank usually is content with defending the point of view in its original, undeveloped, metaphysical form, and he is not prepared to test its usefulness in all those cases which seem to favor the opponent, or even admit that there exists a problem. It is this further investigation, the details of it, the knowledge

of the difficulties, of the general state of knowledge, the recognition of objections, which distinguishes the ‘respectable thinker’ from the crank. The original content of his theory does not. [Fey81, p.199]

The support of complex biological systems for multiple explanations gives rise to calls for explanatory pluralism [Bri10]. Very differently parameterised models can be adequate explanations for the same system. This is common for explanations at different descriptive depths. For instance, the models used in Chapter 4 do not descend below cellular level, and make no reference to macromolecules. We would still like to have macromolecular explanations of RPC function, particularly because these may supply us with means to intervene onto RPC proliferative and lineage outcomes, which is a reason to have different models for investigating the molecular phenomena of *rys* in Chapter 5.

In other plural explanations, adequate, differently-parameterised models of the same system may be describing different aspects of the same level of organization. Pharmacological kinetic studies of G-protein association with receptors, paired with crystallographic studies of the same phenomenon, supply a good example. These models allow experimenters to pursue different descriptive and interventional objectives. Indeed, as Nicholas Rescher has noted, attempts to synthesize many models into a single overarching explanation often result in descriptive chaos [Res00, p.65-6]. Rescher argues the adequacy of some model in its domain usually requires the blurring out of at least some pertinent details of the system that could have been included; the computational tractability of the model requires the same.

If we all accept these forms of pluralism, we are still left with the cases where models are making contradictory claims about reality<sup>1</sup>. Pluralism cannot coherently extend to abandoning the fundamental logical law of non-contradiction, without compromising the entire endeavour of rationalism. We cannot accept logically contradictory notions about the structure of reality without precluding a broadly consistent view of the way the world works, what Rescher refers to as “Cognitive Harmony” [Res05]. Given the complexity of biological systems, we have no option except to express models formally, and to test them rigorously against one another. This is the task of model selection.

Model selection requires that we be able to score models against observations that represent phenomena, to measure their quality. This requires the selection of a quality function. Loss functions, like AIC, used in Chapter 2, express the relative amount of information in the dataset lost by the model, given a set of parameters. Likelihood functions, used elsewhere, express the credibility of a parameterised model, given the observations. The parameters of the model define an n-dimensional “parameter space”; the quality function expresses a hypersurface within this space. By sampling within this space, we may estimate the shape of the surface. This sampling information can be used in three ways: we may propose new, higher quality parameter space locations to sample from, in search of the highest quality parameterisation of the model (model optimisation); we may derive marginal likelihoods for particular parameter values (parameter estimation), or we may estimate the quality of the model over all parameterisations (evidence estimation). These topics are discussed below.

### 15.1.1 Bayesian Epistemological View on Model Comparison

The analyses presented in the data chapters express two views on model comparison. The first, in Chapter 2, is drawn from information theory, while the second, taken up in Chapter 4, is a Bayesian view, albeit with more sophisticated tools and more computing power than has been available to Bayesians in

---

<sup>1</sup>That is, they have incompatible metaphysical content; they are therefore subject to counterinduction as mentioned in Section 16.2.1

the past. In the same way that frequentist analyses may be expressed as a subset of Bayesian analyses (e.g. they seek maximum a priori model parameterisation and likelihood from uninformative priors), informational theoretical approaches to model comparison can be expressed as a subset of Bayesian model comparison theory. In fact, the loss function used in Chapter 2, Akaike Information Criterion, has been adapted to refer to a prior distribution, as the Bayesian Information Criterion [PB04]. The intent of these criteria is to take Occam’s Razor to the maximum-likelihood approach, by penalizing the maximum-likelihood value (or maximum a priori score, in the case of BIC) by the number of free parameters.

The general approach of optimizing a model for a quality against a dataset, then penalizing the best model quality by the parameterisation of the model, allows us to overcome an important problem with many maximum-likelihood approaches to model selection: the requirement for models to be parametrically nested. Model nesting precludes counter-induction; we cannot compare models which express fundamentally different views of how the described system is parameterised, only whether adding more parameters improves a particular view. Escaping this limitation allows us to compare stochastic and deterministic mitotic mode models in Chapter 2; these models express different views of how reality is organised. Still, in important ways, this approach shares the basic problem of simply calculating the MLE: the score does not summarize the robustness of the model fits. That is, a model which is a terrible description of a dataset over most of its plausible parameter space, but an excellent one in a tiny region, can appear to be a better explanation than a model which is a broadly good description over the whole parameter space. Simply using the number of parameters to penalize the best model found fails to capture how justified inclusion of those parameters is, by improving the overall likelihood of sampled models.

This leads us to what may be regarded as the completion of the Bayesian view on model selection<sup>2</sup>, John Skilling’s system of Bayesian inference, nested sampling [Ski06, Ski12, Ski19]. By rearranging the usual presentation of Bayes’ rule, Skilling demonstrates how it specifies computational inputs and outputs associated with the activities of model sampling, parameter estimation, and evidence estimation. Bayes’ rule is typically written as follows, where  $x$  is a logical proposition about the data (e.g. specific values for the cell cycle length and exit rates of the CMZ), and  $Pr(a|b)$  denotes the probability of a given  $b$ :

$$Pr(x|data) = \frac{Pr(data|x)Pr(x)}{Pr(data)}$$

This can be read aloud as “the posterior probability of the proposition, given the data,  $Pr(x|data)$ , is equal to the likelihood of the data given the proposition,  $Pr(data|x)$ , multiplied by the prior probability of the proposition,  $Pr(x)$ , and divided by the marginal probability of the data over all such propositions,  $Pr(data)$ .” This gives the impression that the principal task in statistical analysis is the calculation of the marginal posterior distributions over model parameters, and gives rise to the treatment of the marginal probability,  $Pr(data)$ , as a mere normalizing constant. Skilling rearranges this to put computational inputs on the left, and outputs on the right:

$$Pr(x)Pr(data|x) = Pr(data)Pr(x|data)$$

---

<sup>2</sup>If nested sampling is not *the* completion of the Bayesian system, it is at least *a* complete Bayesian system.

This shows us that the evaluation of a model consists of supplying a prior probability for the proposition,  $Pr(x)$ , and a likelihood function, to assess the probability of the data given that proposition,  $(Pr(data|x))$ . As computed output: (over many propositions  $x$ ), the total evidentiary mass of the data for this model,  $(Pr(data))$ , as well as the posterior parameter estimates,  $Pr(x|data)$ . Sample model parameters are drawn from the prior; the likelihood of these propositions about the data are calculated. By accumulating many such samples, the marginal probability of the model over all of these propositions (the evidence for the model) can be estimated. Because these samples may be weighted by their calculated likelihoods and position on the prior, we may also use them to estimate the marginal posterior probabilities of parameters of interest.

This encapsulates both the numerical procedures involved in model analysis, as well as the epistemological view implied by Bayesian statistics. That is, a model analysis is the joint product of the model and the data, which expresses our belief about the overall evidence for the model  $(Pr(data))$ <sup>3</sup>, as well as allowing us to estimate the distribution of credibility we should assign to various values for parameters of the model (propositions),  $Pr(x|data)$ . These are the two fundamental levels on which quantitative measurements of natural systems allow us to make inferences. We may distinguish between models of the systems in a general sense by their evidence, when applied to the same overall dataset. This allows us to counterinductively test contradictory descriptions of the structure of the phenomenon against one another, inferring which is a better map to the territory. The second level of inference is the ranking of propositions for the parameterisation of models achieved by the sampling procedure. This allows us to determine which particular propositions about the system are supported, given the model and observations. Because the posterior distributions need not be unimodal, we can evaluate these modes as separate hypotheses that are supported to varying degrees by the data. An extensive review of the Bayesian approach to model evidence and model selection has been provided by Knuth et al., covering other methods of estimation not touched on here [KHM<sup>+</sup>15].

This view dispenses with interpretations of model selection as being about finding the “true model” of reality, or of estimating the actual, objective probabilities inhering in things or processes (a view disputed in Section 16.1). Instead, we are guided to focus on the relative quality of models in explaining all of the relevant data we can gather; we may then evaluate the relative quality of specific propositions about the system within those models (i.e., the posterior distributions on model parameters) as we see fit.

### 15.1.2 Model sampling and optimization

There are many techniques used to sample the parameter space of a statistical model, given a quality function which scores the model. These techniques have a variety of purposes. Commonly, one wishes to “optimize” a model by finding the parameter vector which produces the best objective function result, given some dataset. This is often referred to as Maximum Likelihood Estimation, the product being a Maximum Likelihood Estimate, with MLE used to refer to these interchangeably. SPSA is used in this thesis to optimize cell-based models of RPC activity in Chapter 2. The likelihood scores are penalized by the number of model parameters to produce an AIC score. MLE methods are broadly useful for many applications, with some tuning. SPSA is used in control systems, to maintain the consistency of modelled processes by estimating the inputs most likely to achieve a setpoint, for instance [ZZLG08].

---

<sup>3</sup>ie. a better model gives higher marginal probability to observations than a worse one

More sophisticated techniques, like [Nested Sampling](#), involve the estimation of the distribution of quality values over the parameter space, rather than focusing solely on estimating the optimal value.

The process of model sampling requires an algorithmic means to generate proposed parameter vectors to be supplied to the quality function. Most effective sampling methods operate by performing an initial sample from some type of prior information<sup>4</sup>, and then iteratively generating new proposals from these initial ones. This iterative proposal generation forms a chain of linked positions within parameter space. Generally, new proposals are made considering information related only to the last accepted proposal<sup>5</sup>. Because the process is “memoryless” in the Markovian sense, and involves the iterative generation of models, it is often called Markov Chain Monte Carlo (MCMC).

Many algorithms for proposal generation exist, and the accuracy and efficiency of sampling procedures depends on their properties. Most offer a physical interpretation of the quality surface in parameter space. Random-walk proposal generation, like the original Metropolis-Hastings implementations of MCMC, is often too inefficient to be usable for high-dimensional parameter spaces, and has broadly been replaced by these physical sampling methods. [SPSA](#), for instance, is a simple gradient method. New proposals are generated by bracketing an existing sample with a pair of samples that define an  $n$ -dimensional “slope” at the current location. By proposing a new location down-slope, we receive a new sample which is closer to the local optimum, and we may inexorably approach it in this way. Other, more sophisticated interpretations seek to improve on random-walk proposal generation by mechanical interpretations of proposal movement through parameter space, like Hamiltonian Monte Carlo, or thermodynamic interpretations, like simulated annealing [\[ADH10\]](#). These methods are better suited to the estimation of the distribution of objective function values than an algorithm like SPSA.

In order to ensure the accuracy of any such estimated distribution across parameter space, the algorithm should, ideally, produce proposals in “detailed balance”, which refers to the physical concept of reversibility found in classical mechanics. The purpose of constraining proposal generation in this way is to guarantee that particular areas of the parameter space are not over- or under-sampled, distorting the final estimate, as a consequence of the manner in which the sampling algorithm generates proposals. [Galilean Monte Carlo](#) has the property of being in detailed balance, although, as [Chapter 7](#) demonstrates, this is not a guarantee of accurate evidence estimation.

### 15.1.3 Overfitting

Overfitting is a common problem in, particularly, frequentist statistical modelling [\[Bis06, p.9-11\]](#). It can arise from an unjustified excess of model structure relative to the information present in the data. This excess structure allows the model to capture measurement noise in the dataset the model is being scored against. This produces models which appear highly explanatory, but which reflects the particular structure of the measurement errors in the training data better than the overall phenomenon. This problem will typically become apparent in cases of sequential inference; an overfit model will fail to explain new observations as well as it explained previous ones.

A second source of overfitting is the failure to properly account for uncertainty and prior information.

---

<sup>4</sup>In the SPSA optimization performed in [Chapter 2](#), this took the form of the original, poorly optimized fit for the stochastic model and a best guess for a related vector for the deterministic model. In subsequent nested sampling analyses, initialization is performed by sampling randomly from a defined prior distribution.

<sup>5</sup>Proposal rejection can occur for a variety of reasons, depending on how the parameter space is interpreted. In Galilean Monte Carlo, this occurs if proceeding along the specified velocity vector would produce a proposal less likely than the last one.

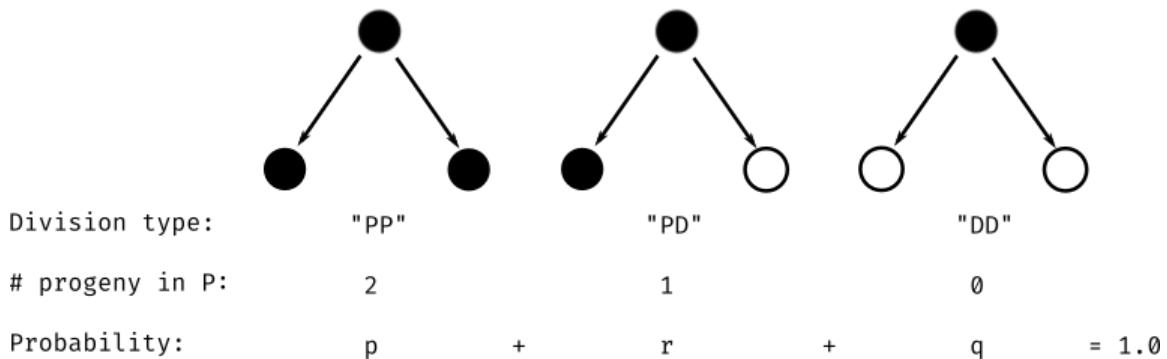


Figure 15.1: **Simple stochastic stem cell model, representing probabilities of cell division events.**

Black circles denote proliferative cells, while white and grey circles denote different types of postmitotic offspring. “Number of progeny in P” is the number of mitotic offspring produced by each type of division. The probability of each division type must sum to 1, as all possibilities are represented, granting that the division types are defined by the postdivisional mitotic history of the offspring.

For instance, if unidimensional measurements are modelled with a Normal Gaussian distribution, the MLE Normal Gaussian (whose parameters arise from calculating the mean and standard deviation of the measurements, given the assumption of Normality) can be an overfit to the data [Bis06, p.28]. In this simple example, this usually means that the MLE Normal underestimates overly influenced by “outliers”. This is one case of the general overfitting problem seen in MLE methods [Gre17]. Accounting for available prior information is the best solution to this problem. As this thesis demonstrates, appropriate accounting for our uncertainty about the parameters of the Normal model, and comparison of the Normal case with appropriate alternatives that might better reflect the underlying causal structure that produces the “problem” data (in this case, Log-Normal models), both help in producing more accurate estimates of model parameters.

#### 15.1.4 Simple Stochastic Models

The Simple Stochastic Model is schematised in Figure 15.1. This is the basic structure of the majority of formal models in the stem cell literature, derived from post-hoc analyses of populations taken to include stem and progenitor cells. The population-level approach is usually explicit, as no differentiation is made between types of proliferating cell; in general, no particular cell is identified with a stem cell, nor can any be identified from the necessarily retrospective population data used to infer the parameters of the model.

The central concept of the model is that mitotic divisions can be categorised by the number of progeny which remain mitotic after the division. This categorisation is necessarily retrospective. This must be kept in mind when considering models of this type, as this categorisation does not necessarily imply that there is some mechanism by which the cell specifies the fate of offspring *at the time of mitosis*, although there is extensive evidence for the coupling of mitotic and specification processes at the molecular level [TFC13, TR14, Dal15, DD17].

In effect, the model compresses the process of fate specification into individual mitotic events. Since the primary distinction between cells in the model is simply whether they are proliferating or not, the model also elides any heterogeneity within the proliferating population. Beyond not identifying partic-

ular cells as “stem cells”, this may make models derived from the SSM inappropriate for proliferative populations with a large degree of heterogeneity. The classic idea of a small number of slowly proliferating “true” stem cells and a larger population of rapidly dividing “transit amplifying” progenitors can only be represented by compartmentalised, independent SSMs (as implemented in Chapter 4).

As Fagan notes, in the SSM, “relations among p, r, and q values entail general predictions about cell population size (growth, decrease, or ‘steady-state’), and equations that predict mean and standard deviation in population size, probability of [lineage] extinction, and features of steady-state populations are derived.”<sup>6</sup> [Fag13, p.60]

Typically, this type of model has been employed to describe population dynamics of proliferating cells in assays generating clonal data, where a “clone” here refers to the population constituted by all offspring descended from some particular (usually “initial” and sometimes therefore taken for “stem”) proliferating cell. This population is the *lineage* generated by the cell. The mitotic events which give rise to a lineage are summarised as a Galton-Watson branching process, a statistical model originally intended to represent the lineage extinction of surnames. In the case of branching process models applied to proliferative cells, the random variable determines the mode of division of mitosing cells, defined by the proliferative state (construed in the model as being either mitotic or postmitotic) of progeny. For any given division, a cell may produce two mitotic, one mitotic and one postmitotic, or two postmitotic progeny, and each of these division modes is given a defined (often, but not always, static) probability. Given these values, the history of a cell lineage may be simulated; the output of many of these simulations pooled together, in Monte Carlo fashion, allows the statistical properties of the dynamics of population of simulated cells to be estimated.

### 15.1.5 Systems of difference equations

Models defined by difference equations are used in Chapter 4. A difference equation is the discrete counterpart of a continuous differential equation. Both types of equations can be used to describe the time-evolution of a system, because they can be recursively applied to calculate the value of model outputs at a time  $t$ , given the value of the outputs at  $t - p$ ,  $p$  time units in the past [KP01, p.1]. In the models presented in this thesis, the simulation’s unit of resolution is the day, which motivates the use of the algorithmically simpler analytic solutions to difference equations. As implied above, the time-dependent value of a model output may depend on a output of another equation. This is the case for the model of estimated CMZ annulus population and retinal volume presented in Figure 4.2, where the volume estimate depends on resolving the value of the day’s starting population before evaluating the volume contribution of that population, given the exit rate. The slice models (see Slice\_Model) used to interrogate phase transition timing in Figure 4.6 are not fully systems of difference equations, since the population difference equation depends on evaluating a continuous power law model of lens growth. Moreover, the slice model of decaying cycle rate (see Decay\_Model) depends on the daily evaluation of the continuous exponential cycle time function.

---

<sup>6</sup>While Fagan refers to “stem cell” extinction, the model does not specifically define stem cells, nor does it imply intergenerational continuity, such that a particular intergenerationally identified stem cell should be said to have become extinct. The unit which survives or is made extinct is the lineage derived from some particular proliferative cell.

### 15.1.6 Independent Component Analysis models of sequence emission

The models of *rys* nucleosome position sequence emission presented in Chapter 5 are products of Independent Component Analysis (ICA), a technique which is used to infer the independent contributions of multiple signal sources on a single multiplex channel; it can be thought of as an application of information theory [LGBS00]. The most common example offered to illustrate this is the separation of the individual streams of speech produced by multiple speakers conversing in a noisy room [JHTS00]. ICA can be used, in this context, to model the process by which a listener separates the independent signals of the speakers from background noises. In this thesis, the *D. rerio* genome is treated in this fashion. In effect, any given sample of genomic sequence may be treated as a multiplex channel produced by a host of independent causal processes. In this sense, each nucleosome position is a “noisy room”, with its own local structure of background noise and conversational participants.

The distinction between background noise and signal in an ICA model is made both conceptually, and in the numerical representation of these parts of the model. I have closely followed the early, pioneering example of Down et al. [DH05]. Down et al. use Hidden Markov Models, optimized on promoter sequences, as background models for promoter sequences in general. Against this, a foreground of Position Weight Matrix signals, representing binding motifs within the promoter region, is inferred by nested sampling. Similarly, in Chapter 5, separate HMMs were used to represent genomic sequences from the exonic, “periexonic” (introns and  $\pm 500$ bp intervals around the CDS), and intergenic sequences, against which the foreground of repetitive PWM structural signals is inferred, representing characteristic preferred contact points between the core nucleosome and genomic DNA. The HMM background models and  $s$  PWM signal models are linked in their description of a set of  $o$  observations by an  $o \cdot s$  mixing matrix. While many implementations of source and background mixing are possible, in this case, Down et al.’s simple binary implementation is retained. Each observation  $o$  is associated with a “mix vector” of  $s$  binary values, indicating whether or not  $s$  source is to be scored in that observation.

The likelihood scoring function for this model is numerically equivalent to Down et al., and is available in full in Section 17.6.3. Briefly, for every observed sequence whose mix vector indicates a source is present, a score matrix is produced for that source and observation. This involves “scanning” the PWM down the length of the observation, offering a score for the entire PWM at each base (in the default reverse-complement operation of BioMotifInference.jl, this is repeated with the reverse complement of the motif, for motif detection on the reverse strand). This feature of likelihood function confers the “multiple uncounted” nature of the motif representation in these models. This means that multiple occurrences of a PWM source motif in a sequence can be detected, and that no particular number of occurrences must be specified. These features make this model particularly well suited to the detection of the repetitive structural signals inferred in Chapter 5. The PWM source score matrices are subsequently “woven” by combination with background scores, an expectation value for motif observation, given the length of the sequence, and a penalty for every source included in the observation, which prevents overfitting.

### 15.1.7 Position Weight Matrices

Position weight matrices (PWMs) were used to model signals arising from nucleosome contacts with their DNA positions in Chapter 5. For a sequence signal of length  $\lambda$ , the PWM which defines it is a  $\lambda \cdot 4$  matrix, with each of the four columns representing the categorical probability weights of A, C, G,

and  $T^7$  base emission at the position represented by row  $\lambda$  [SSGE82]. The PWM is typically used in a frequentist context, without prior distributions over its parameters. However, in a Bayesian context, the PWM is easily interpreted as a  $\lambda$ -base length vector of discrete categorical distributions over base emission frequencies<sup>8</sup>. A computationally useful conjugate prior distribution for discrete categoricals is the Dirichlet distribution [Min00]. Vectors of Dirichlets have been used as the priors for signals in BMI.jl models. Because of their conjugacy, Dirichlet priors can theoretically be updated to produce full posterior distributions over the PWM’s parameters. BMI.jl does not provide for this, for reasons related to the ICA PWM model implementation discussed in Chapter 10. The 4-parameter Dirichlet over the 4-category probability simplex can be usefully thought of as a density cloud in a 3-dimensional tetrahedron (i.e., an ordinary one with 4 vertices). The relative density within the tetrahedron expresses the prior (or posterior) probability of a particular categorical distribution at base  $\lambda$ . The points of the tetrahedron represent categorical distributions with unit probabilities for one of the four bases, while points in the center of the tetrahedron represent categoricals with identical .25 probabilities of emitting each base.

### 15.1.8 Hidden Markov Models

Hidden Markov Models (HMMs) were used to model *D. rerio* genomic background noise in Chapter 5. An HMM is a state machine; it can be in one of a finite number of states at any given time. Each state is associated with a probability distribution on the model’s emission of symbols. The HMM generates a sequence of such symbols by transitioning from state to state (or remaining in the same state, called autotransition), emitting after each transition. The states are “hidden” in the sense that their presence is inferred from the sequence of data upon which the model is trained, and not from direct observations of the states themselves. HMMs are, therefore, well-suited to modelling sequences of outcomes with obscure or irrelevant causal structures. A  $k$ -state HMM that emits  $s$  symbols is defined by an initial state probability vector of length  $k$ , a  $k \cdot k$  state transition probability matrix, and the outcome probability vectors for each state [Rab89]. HMMs can be used in a variety of contexts, each of which is associated with different algorithmic procedures. For instance, the Viterbi algorithm can be used, given an HMM and a sequence of observations, to infer the most likely state the HMM would have resided in at each position in the sequence, had it been generated by that HMM. An example of this use is the inference of exon and intron features from an HMM trained on annotated CDSes [HSF97]. In this thesis, HMMs are optimized by an Expectation Maximization algorithm, and are subsequently used to score the likelihood of *ryst* sibling and mutant nucleosome position sequences, given their genomic context.

## 15.2 Statistical Methods

### 15.2.1 Akaike Information Criterion scoring

An Akaike Information Criterion (AIC) score for a model depends on the number of free parameters in the model,  $k$ . AIC differences between models can be readily calculated from the residual sum of squares (RSS), which is used as the quality function for measuring model output divergence from observations.

---

<sup>7</sup>This base ordering is computationally significant, since it allows for trivial reverse complementing by reversing the matrix values across both dimensions.

<sup>8</sup>That is, a vector of nonparametric categorical distributions, themselves on a support of ACGT vector of the 4 nucleotide base emission frequencies. BMI.jl encodes the support as the integers 1234 for computational efficiency

For  $n$  observations and model parameters  $\theta$ ,  $RSS = \sum_{i=1}^n (y_i - f(x_i, \theta))^2$ . For model comparison, AIC is calculated as:

$$mAIC = 2k + n \cdot \ln(RSS) \quad [\text{BA02}]$$

This is not the “true” value of AIC but assumes that a constant associated with the same data cancels out during model comparison (i.e. by subtraction). AIC is a flawed model comparison criterion. It takes no account of uncertainty in parameters, and tends to be biased toward simple models [Bis06, p.33]. AIC does not fare well for the evaluation of models whose parameters are not well-constrained by the data [Bis06, p.217]. AIC calculations are implemented in Section 17.1.9.

### 15.2.2 Monte Carlo simulations

Monte Carlo refers to the repeated sampling of a statistical model’s output to estimate quantities related to it. Many of the simulations presented in this thesis involve two distinct uses of Monte Carlo techniques. The first use is the exploration of parameter spaces by Markov Chain Monte Carlo (MCMC), with sequential sampling of locations in parameter space, such that the probability of a proposed new location depends only the current position in parameter space (ie. it is a Markovian trajectory). These proposals can be generated by various means; in this thesis, Simultaneous Perturbation Stochastic Approximation, Galilean Monte Carlo, and an ad-hoc permutation routine are used. The second use of Monte Carlo is the estimation of model outcomes distributions, which involves repetitively sampling the same position in parameter space to empirically determine a distribution, against which observations are scored. This is how the majority of the models in the thesis are scored.

### 15.2.3 Simultaneous Perturbation Stochastic Approximation (SPSA)

SPSA is a model optimization technique. It is a reliable gradient-descent method for finding the local optimum in parameter space, given an objective function and a sample starting position. It has been proven useful in a wide variety of engineering applications [KHI97, ZZLG08], not least because it is tolerant to measurement noise in the objective function. This makes it useful for applications where exact determination of the measurement function’s value is not possible; these include on-line intervention into systems with real measurement noise [ZZLG08], as well as the calculation of model likelihoods by Monte Carlo.

The implementation of SPSA in Section 17.1.9 follows the example of Spall exactly [Spa98], extended with the modifications of Sadegh for bounded parameter spaces [Sad97], as suggested by Spall. As noted above, SPSA is a gradient-based method. It moves a single model-particle through parameter space. At each iterate, the local quality gradient is estimated by taking two samples on either “side” of the particle, given by adding and subtracting a perturbation vector to/from the current location. The perturbation vector is composed by random selection of positive and negative magnitudes in each model dimension. This is “Simultaneous Perturbation”, the alternative being to perturb dimensions one at a time, which the technique Finite Difference Stochastic Approximation (FDSA) does, instead. SPSA is more accurate and efficient, especially for high-dimensional problems [Chi97]. Given the quality estimates at the two bracketing samples, the model-particle’s position is updated so as to move it downslope (in the case of AIC), toward the local optimum. In order for ever-smaller, higher-quality regions of the parameter space to be consistently found, the distance sampled from the particle for the gradient estimation, and for moving the particle itself, must both consistently decline. This is achieved by tuning coefficients,

from which gain sequences are derived. Spall provides useful guidelines on how these may be selected, even automatically [Spa98]. Experience with the SMME models suggests that a degree of hand-tuning is probably usually necessary.

### 15.2.4 Bayesian treatment of Normal models of unknown mean and variance

As noted in the section on overfitting, the usual MLE method for characterising a variable population with a Normal gaussian (calculating the mean  $\mu$  and variance  $\sigma^2$ ) is susceptible to overfitting. In particular, these methods fail to convey the wide uncertainty that usually obtains on the variance parameter of the Normal model [Mur07]. Normal models figure heavily in the description of variable population-level data. While these models are well-justified both by their ubiquitous success in parameter estimation, and by information theoretic considerations [JBE03], and need not reflect the actual distribution of population measures, we use them in this thesis to model these parameters directly. In the main, this is done using LogNormal models, which are Normal gaussians fitted to the logarithm of the data. This change of variable allows more accurate modelling of population data with long right tails, which is common for the outcomes of growth processes. As Heath notes, long right tails are to be expected for values on ranges that are closed at the 0 minimum and open at the high end [Hea67]. Heath suggests that the Log-Normal distribution should be preferred by default for data of this type. We instead measure the evidence for LogNormal against Normal models for the distributions of our measurements in Section 12.0.1.

A valid Bayesian means of modeling the credibility of mean and variance parameters on Normal gaussian distributions is the use of Normal-Gamma (NG) prior distributions. This assigns normally distributed uncertainty on  $\mu$  and gamma distributed uncertainty on precision,  $\lambda$ , giving rise to a joint normal-gamma (NG) distribution:

$$p(\mu, \lambda) = NG(\mu, \lambda | \mu_0, \kappa_0, \alpha_0, \beta_0) \stackrel{\text{def}}{=} \mathcal{N}(\mu | \mu_0, (\kappa_0 \lambda)^{-1}) Ga(\lambda | \alpha_0, rate = \beta_0) \quad [\text{Mur07}]$$

In some cases, we have made use of the more convenient Normal-Inverse-Gamma (NIG) distribution, which takes an inverse gamma distribution over the variance  $\sigma^2$  rather than a gamma distribution over the inverse of variance,  $\lambda$ :

$$p(\mu, \sigma^2) = NIG(\mu, \sigma^2 | \mu_0, V_0, a_0, b_0) \stackrel{\text{def}}{=} \mathcal{N}(\mu | \mu_0, \sigma^2 V_0) IG(\sigma^2 | a_0, b_0) \quad [\text{Mur07}].$$

These prior distributions have useful properties. One of these is conjugacy; posterior distributions in the same form as the NG or NIG prior may be calculated directly, allowing for sequential inference, although this is not performed in this thesis. Another useful property is that the marginal credibility distribution over the normal model's mean is given by a T distribution, being a weighted sum of an infinite series of Normal distributions with different variances. The marginal distribution of the model mean, calculated from an uninformative NG or NIG prior, is identical to the frequentist Standard Error of the mean [Mur07].

We use the Julia package `ConjugatePriors.jl` for the NG and NIG models. However, this package is not provided with functions for fitting these distributions from uninformative priors, nor for calculating marginal distributions. We provide a series of utility functions for these and related tasks in the `NGRefTools.jl` package, included in Section 17.2.

### 15.2.5 Empirical Bayes linear regression

Linear regression models are those that model the response of some dependent variable to a series of independent variables, which are assigned weights. The Empirical Bayes method of linear regression used in Chapter 4 allows the estimation of posterior distributions on the weights, and on the noise of observations about the mean, by specifying the maximum likelihood prior hyperparameters, given the data. Determination of prior distributions from observations is what gives the Empirical Bayes family of methods its name [BSB00, p.373]. Normally, priors are fixed without reference to the dataset to be examined.

We used the iterative solution to likelihood maximization provided by the analysis of Bishop [Bis06]. Briefly, this consists of a series of steps in which  $\alpha$  distributions on weights and  $\beta$  distributions on the response variable's noise are estimated from the data. This process maximizes the probability of the data  $y$  given the  $\alpha$  and  $\beta$  distributions, over possible values for weights  $w$ :

$$p(y|\alpha, \beta) = \int p(y|w, \alpha, \beta)dw$$

Bishop's method naturally produces an estimate of the model evidence,  $\log Z$ , similar to the more general [nested sampling](#), albeit without an estimate of the error in the evidence.

### 15.2.6 Galilean Monte Carlo

Galilean Monte Carlo (GMC) is an algorithm for the [sequential Monte Carlo](#) generation of new sample proposals within a parameter space. While GMC was introduced in 2012 [Ski12], the algorithm used in `GMC_NS.jl`, presented in Chapter 7, is Skillings' 2019 revision [Ski19], which prevents model-particles from leaving the likelihood contour in order to find a reflection, and adds an additional reflection direction.

GMC operates by assigning model-particles (ie. a particular parameterisation of that model at some point in parameter space) a velocity vector, chosen isotropically (ie. uniformly, without preference for any direction). New proposals are generated by moving the particle along the velocity vector. A proposed position's likelihood is calculated, and the proposal is accepted if this value is greater than or equal to the contour. Equality is required to allow particles to search flat likelihood "plateaus", which are prominently seen in the models of `CMZNicheSims.jl`, presented in Chapter 8, due to the discretised (timepoint-bound) nature of the observations.

If a proposal is rejected, its current vector crosses the likelihood contour at a distance less than that of the proposal. If this occurs, the particle reflects off the boundary and carries on down the reflected vector with the same velocity. Unlike MCMC methods that rely on differentiating the likelihood surface, in GMC, nothing about the underlying surface is used to propose positions, so it samples evenly from within the likelihood contour, fulfilling detailed balance [Ski12]. This means that particles are equally likely to transition between positions in the forward process as the reverse process; in GMC the proposal process to move up-vector is identical to the one to move down vector, eg. a particle reflecting normal to the boundary.

When a reflection is being proposed, samples are sequentially taken in three "directions", with the first of these to fall within the likelihood contour used as the reflected model-particle. A schematic summary of this scheme is presented in Figure 15.2. The orientation of the likelihood boundary is inferred by the difference in likelihood between the current particle position and the forward position outside the contour, which is used to estimate a normal vector. The "east" reflection is calculated first,

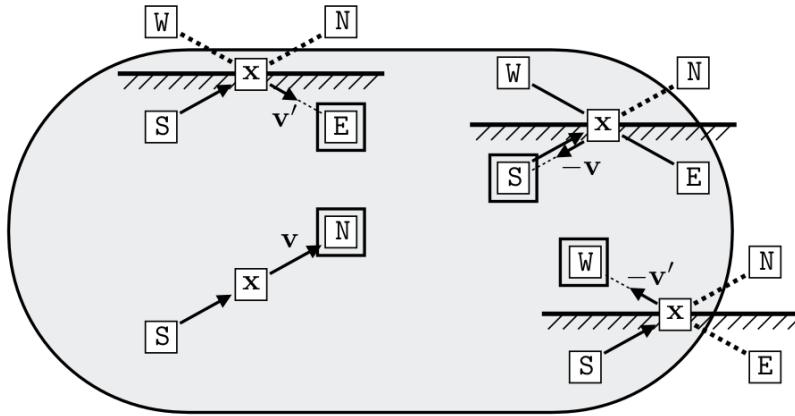


Figure 15.2: **The four directions of GMC particle movement.**

”North” is forward movement along the trajectory; East, South ,and West are reflected vectors. Excerpted from [Ski19] under CC-BY 4.0.

followed by the “west” reflection, which uses the reversed reflection vector, followed by the “south” reflection, which reverses the particle velocity rather than relying on the boundary normal estimate. In `GMC_NS.jl`’s implementation, each time a forward or reflected model is rejected because it is outside the likelihood contour, this is logged in the particle’s PID tuner memory, which reduces the timestep for another round of forward, east, west, and south searches. Eventually, if no new model can be found above the ensemble’s minimum timestep, the particle’s trajectory is terminated.

### 15.2.7 Nested sampling

Nested sampling is an algorithmic process for estimating the quality of a model, given observations. It arises from fundamental considerations of imposing a measure on statements about a model of a phenomenon [Ski12]. The most basic such statement is an atomic true/false proposition. A measure is a function that assigns a value to a set. A measure of a model’s evidence values a statement or a set of statements about the model, given some observations. A set of statements about the model can be represented as a boolean lattice of the statements and their OR combinations [KS12]. The sum and product rules arise in combining measures assigned to these statements into measures of their sets: ie. the probability of A OR B is  $p(A) + p(B)$ , while the probability of A AND B is  $p(A) \times p(B)$ . These operators are required in order to preserve consistent ordering between measures and their combinations. This lattice-oriented view thus gives rise to a unique calculus for measuring the divergence of model output from observations over sets of model-propositions[KS12]. This calculus of divergence leads naturally to the concept of information [Ski12], which can be understood in a nested sampling context as related to the compression ratio of the final posterior sample on the prior<sup>9</sup>. Moreover, it allows the measure of the total quality of the model, given data: Bayesian evidence or logZ. The uniqueness, basic symmetry, and non-arbitrary nature of the axioms underlying nested sampling strongly suggest that this approach is fundamentally superior to other, more ad hoc methods of estimating model quality (eg. AIC). While nested sampling can use any quality function, normally a measure of likelihood is used.

<sup>9</sup>Information, H, is the negative logarithm of the compression ratio. [Ski06]

The most basic nested sampling procedure involves initialising an ensemble of model-particles by sampling evenly from prior distributions on their parameters. Given this initial ensemble, “learning” is performed by compression, where the least likely particle is removed, and replaced by one from inside the likelihood contour established by the least likely particle. Over many such iterates, the ensemble of model-particles is compressed into more and more likely areas of the parameter space. These samples allow us to reconstruct the posterior from their likelihood-weighted parameter locations. More importantly, they allow the estimation of the Bayesian model evidence for the dataset.

Integrating under the quality surface of the parameter space is likely to be an intractable problem for high-dimensional models. However, the dimensionality of the problem can be reduced. If the samples are considered solely in terms of the amount of prior mass enclosed within their likelihood contour, and the likelihood itself, the problem becomes a two-dimensional integration. To do this, the samples taken from the ensemble must be put in order from least to most likely (hence the requirement for consistent ordering in the underlying quality measure, mentioned above). Nested sampling produces this ordering by removing the least likely particle from the ensemble with each iterate. A schematic of the evidence integration is displayed in Figure 15.3.

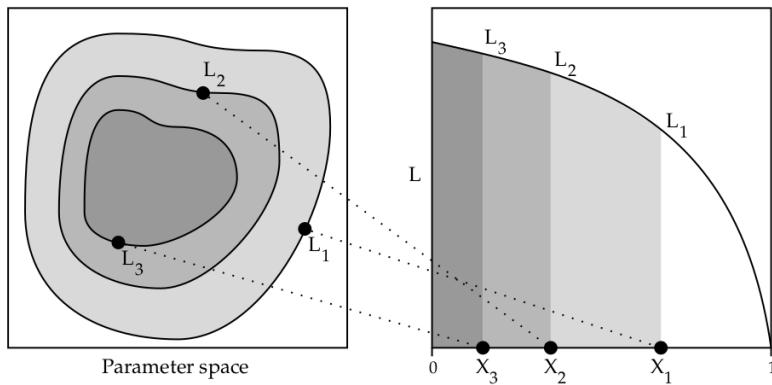


Figure 15.3: **Schematic of evidence integration by nested sampling.**  
 $L$ , likelihood.  $X$  gives the fraction of prior mass remaining. Excerpted from [Ski06], copyright John Skilling.

Both samplers implemented in this thesis use the common crude approximation of the sequence of enclosed prior mass values, distributed along the  $X$  axis in Figure 15.3. In this approximation, the fraction of prior mass enclosed by iterate  $i$  in an ensemble of  $N$  particles is given as  $X_i = \exp(-i/N)$  [Ski06]. Very accurate samplers use this approximation to good effect, including MultiNest [FHB09]. Having an estimate of the prior mass enclosed at each iterate allows us to determine how much of the prior lies between two iterates by subtracting the inner contour’s enclosed mass from the outer<sup>10</sup>. Doing so estimates the width of the prior between the two iterates. Multiplying the iterate’s sampled likelihood by its prior width gives an estimate of the mass of the evidence between these two contours. Summing the logarithm of the masses of these nested shells between iterates estimates the model evidence,  $\log Z$ . As Skilling notes, this value is not dimensionless, but rather has data units [Ski12]. That is, the  $\log Z$  value depends on the observations supplied to the likelihood function. However, by taking the evidence

<sup>10</sup>GMC\_NS.jl and BMI.jl both use a trapezoidal approximation of the shell width, rather than simple subtraction, although the difference this makes to the evidence estimates is slight.

ratio between models (by subtracting one evidence value from another), we produce a dimensionless quantity,  $\log ZR$ . This expresses the extent to which the data favour one model over the other (a  $\log ZR$  of 0. would indicate the models are equally credible in light of the data). The uncertainty of  $\log Z$  values, and hence  $\log ZR$ , are available by using the information,  $H$  of the ensemble. This uncertainty is estimated as  $\sqrt{H/N}$  for an ensemble of  $N$  particles. The uncertainty of a  $\log ZR$  value allows us to express the credibility range of the  $\log ZR$  estimate as a measure of significance. We calculate standard deviations  $\sigma$  of significance by taking the ratio of the uncertainty over the absolute value of  $\log ZR$ . This suggests the probability that the  $\log ZR$  ratio includes 0., indicating equal credibility of the models, assuming normally distributed errors. For instance, a  $3\sigma$  significance on a  $\log ZR$  value indicates a 99.7% probability that the favoured model in fact has greater model evidence than the disfavoured model.

### 15.2.8 Expectation-Maximization optimization of HMMs

To select background models for *D. rerio* genomic noise in Chapter 5, HMMs must be optimised against samples of genomic sequences. HMMs are initialised from prior distributions on their parameters. The prior on symbol emission is usually uninformative. However, because the genomic features we wish to capture with these models consist of runs of bases with particular nucleotide biases, we supply a prior favouring larger values for the transition matrix diagonal, making autotransition (state residency) more probable. Once models are initialised, new model-proposals are generated by iterating expectation-maximization algorithm steps on the HMM. EM algorithms monotonically produce more probable models with every step [Rab89], without any models being rejected. However, they are generally thought to be susceptible to models getting “stuck” at saddle points and in local minima, although EM algorithms will reliably find global optima for some HMMs, under certain conditions [YBW15]. It is also possible to use [Markov Chain Monte Carlo](#) to perform this task, although doing so requires a carefully constructed sampler [Ryd08].

Expectation-maximization algorithms work by iteratively cycling through (E)xpectation and (M)aximization steps. In the E step, the most likely hidden states of the HMM, given the observations, are calculated. In the M step, the model parameters are set to those that make the inferred hidden state sequences most likely. More than one way of calculating these quantities is available. `BioBackgroundModels.jl` implements the classic Baum-Welch (BW) algorithm [BP66], closely following Rabiner [Rab89], with the corrections for errata in multiple observations calculation provided by Rahimi [Rah]. It also implements the linear memory algorithm of Churbanov et al., [CW08].

As the name suggests, the Churbanov algorithm’s memory use increases linearly with the number of HMM states, while BW’s increases geometrically. This arises from the different way in which the calculations are performed. For  $O$  observations of  $T$  length, a  $K$ -state HMM requires an array of  $O \cdot T \cdot K_i \cdot K_j$  to calculate  $es$ , which are the probabilities of being in states  $i$  and  $j$  at subsequent times, given the model and observations. Memory use increases over the length of the observations vector, resulting in large spikes of memory use over the course of the E-M iteration. The relevant code is found in Section 17.5.7, line 26. For large (genome-scale)  $O$  and  $T$ , memory use on  $\geq 6 - K$  HMMs is prohibitive, especially if more than one model is being learnt on the same machine. By contrast, the Churbanov algorithm cumulates the relevant probabilities using two  $O \cdot K_i \cdot K_j \cdot K_m$  arrays, representing the marginal probability of all state paths emitting the observed sequence, that pass through an  $i-j$  transition at least once, and terminate in state  $m$ . These two arrays are the memory commit for observations of any length (i.e. there is no  $T$  dimension). The relevant code is found in Section 17.5.9, line 12. Usually,

this represents a trivial memory commitment, and longer observations do not increase it. Theoretically, this algorithm can be used to learn HMMs on sequences of any length, unlike standard BW. Although the linear memory algorithm is theoretically less performant than BW [CW08], our implementation benchmarks faster than BW in most realistic observation sets. BW may be more appropriate for small numbers of short observations, but Churbanov should usually be the first choice for learning background models on large samples of genomic sequence.

## Chapter 16

# Theoretical Appendix B: Metaphysical Arguments

### 16.1 Chance is not a valid explanation for biological phenomena; Randomness is a measure of sequence compressibility and not an explanation

Part of this document examines Harris' regular invocation of "stochastic" and related adjectives to describe the behaviour of RPCs. This is what lead me to characterise the theory primarily in those terms- the Stochastic Mitotic Mode Explanation. The significance of this may not be obvious; most scientists assume that they know what words like "stochastic" and "random" mean well enough to use them in rigorous technical publications. We may not be aware that there has been a sprawling debate on the meaning of these terms since the earliest statistical formulations began to appear in the 19th century. However, even the simplest examples (as current today as they were in Laplace's time) reveal how difficult this topic can be.

If we take the example of the coin flip, a process whose outcome we generally regard as being in some way "stochastic" or due to "chance", we immediately face the question of whether these descriptions refer to our inability to know the outcome of the process, or to properties of the process itself. In other words, if we could specify the mechanics of the coin toss with sufficient precision, could we predict the outcome? This reflects two senses in which we may describe a process as "stochastic": an epistemic sense (it is stochastic because we are unable to predict its outcome *a priori* precisely and so must make resort to stochastic modelling), or an ontological one (we describe the process as stochastic because this, in some way, describes how it *really is* independent of our knowledge of it).

Complicating matters is the sheer number of implications that we tend to associate with "stochasticity" and "randomness". We may be saying something about the causal structure of an event with deep metaphysical implications. It is common to distinguish between "deterministic" and "stochastic" processes, as though "stochastic" literally meant "indeterministic"- something like the Copenhagen interpretation of quantum physics. We may mean something about the apparent disorderliness of a series of outcomes of some process, with mathematical and information theoretical implications. What is an apparently simple observation- cellular fate distribution in RPC lineages is "stochastic", now seems to

require at least a little clarification or interpretation. We may say that stochasticity, for Harris, applies to at least the following entities:

1. The population-level outcomes of the RPC fate specification process (the phenomenon itself)
2. The overall behaviour of the macromolecular system whose operations produces these outcomes (the macromolecular mechanism itself)
3. The particular behaviour of some component of the macromolecular system, eg. stochastic expression of transcription factors (the macromolecules themselves)
4. The statistical generalisations used to characterise relevant aspects of the behaviour of the mechanism or the macromolecules (the model itself)

It is hardly fair to expect the SMME to be advancing a coherent theory about the ontological, objective basis of randomness or probability. Still, this leaves us in the awkward position of not knowing quite what the leading explanation for retinal formation is actually saying about its explanandum. The SMME is thus at risk of circularity- the explanandum (unpredictable variability in clonal outcomes) has as explanans a mechanistic explanation containing an abstract mathematical model tuned to produce this unpredictable variability. This may, in other words, turn out to be a case of Jaynes’ “mind-projection fallacy” [JBE03, p.506], where a property of a statistical model is taken as a property of nature. Before considering this, we need to define our terms more carefully to avoid the pervasive confusion mentioned above.

#### 16.1.0.1 Chance versus Randomness

A commonplace belief is that randomness refers to outcomes produced by chance events. In an extensive and useful discussion, Antony Eagle reviews the evidence for this Commonplace Thesis, or (**CT**) [Eag18], drawing on discussions in the physics literature. Importantly, he notes that chance and randomness are not identical, and that one can conceivably exist without the other. This, in effect, disproves the (CT)- it is very difficult to imagine how the two concepts can be directly related in this productive fashion. I will attempt a brief summary of Eagle’s argument:

Chance is used to refer to processes. Exemplars are coin flips and die rolls. We can think of these as “single-case” probabilities that we take to inhere in the process. For instance, we may say that an evenly weighted coin has a .5 probability of returning a value of heads on a flip, even if it is only flipped once. That is, probabilities can be taken to be objective properties of individual instances of processes, and not only descriptions of the frequencies of the process’ outcomes over many repetitions. This is closely related to the logical concept of “possibility”. If something is possible, it has a chance of occurring. However, possibility is a logical binary; something is either possible or impossible. A “single-case” probability is understood as something like an objective feature of a system as a whole, given its actual configuration and the relevant natural laws.

Randomness, by contrast, mainly refers to process *outcomes*. That is, randomness is a property of a series of outcomes of some process. In general, we may say that a random sequence of outcomes is one that cannot be generated by an description shorter than the sequence itself. That is, there is no set of rules that can generate a genuinely random sequence from a shorter sequence. In algorithmic information theory, the length of the ruleset required to produce some piece of information (like a

sequence of measured outcomes) is called the Kolmogorov complexity of that object; if the Kolmogorov complexity of the object is equal to the object's length, the object definitionally has the property of algorithmic or Kolmogorov randomness<sup>1</sup>.

Eagle produces numerous examples of the dissociability of these concepts, from which I have selected two concise illustrations:

### Chance Without Randomness

...

A fair coin, tossed 1000 times, has a positive chance of landing heads more than 700 times. But any outcome sequence of 1000 tosses which contains more than 700 heads will be compressible (long runs of heads are common enough to be exploited by an efficient coding algorithm, and 1000 outcomes is long enough to swamp the constants involved in defining the universal prefix-free Kolmogorov complexity). So any such outcome sequence will not be random, even though it quite easily could come about by chance.

...

### Randomness Without Chance

...

Open or dissipative systems, those which are not confined to a state space region of constant energy, are one much studied class [of the objects of deterministic classical physics- notably, biological systems are dissipative], because such systems are paradigms of chaotic systems ... the behaviour of a chaotic system will be intuitively random ... [t]he sensitive dependence on initial conditions means that, no matter how accurate our finite discrimination of the initial state of a given chaotic system is, there will exist states indiscriminable from the initial state (and so consistent with our knowledge of the initial state), but which would diverge arbitrarily far from the actual evolution of the system. No matter, then, how well we know the initial condition (as long as we do not have infinite powers of discrimination)<sup>2</sup>, there is another state the system could be in for all we know that will evolve to a discriminably different future condition. Since this divergence happens relatively quickly, the system is unable to be predicted ... Just as before, the classical physical theory underlying the dynamics of these chaotic systems is one in which probability does not feature. [Eag18]

Therefore, the (CT) is untenable. Processes are “chancy”; collections of process outcomes, “trials”, or instantiations are “random”. It is tempting to say that Harris is explaining random fate outcomes with descriptions of chancy processes occurring internally to RPCs. Let us examine whether this is plausible.

#### 16.1.0.2 Chance in molecular mechanisms

What might it mean to describe the behaviour of a biological macromolecular system as “chancy”? Let us again distinguish between the ontological and epistemic dimensions of this description. There is a sense in which the operation of a macromolecular mechanism could be said to be objectively chancy, and one in which the “chancy” outcome reflects our ignorance of some source of variability in the process.

Eagle proffers two common lines of argument in favour of chanciness as an objective property of processes. The first is the notion of the “single-case” probability mentioned above. The examples given

<sup>1</sup>The Kolmogorov complexity of a sequence can be estimated, contrary to common belief [LV08]. Minimum Message Length expresses a similar concept [WD99]. More prosaically, one may simply compress a serialized representation of the sequence on one's hard drive with a reasonably good compression algorithm; the degree of compression achieved is a good indicator of the degree of non-random order available to shorten the sequence's description.

<sup>2</sup>Note that this condition defines chaotic randomness as an epistemic, rather than ontological, feature of complex systems- a being with infinite powers of discrimination *could* predict the evolution of a complex classical system with perfect accuracy.

are single coin flips, and the decay of single radioactive atoms, which are commonly taken to have chancy outcomes irrespective of anyone's beliefs about them. As Eagle notes, this is closely related to frequentist ideas about stable processes, or trials:

It is the stable trial principle that has the closest connection with single-case chance, however. For in requiring that duplicate trials should receive the same chances, it is natural to take the chance to be grounded in the properties of that trial, plus the laws of nature. It is quite conceivable that the same laws could obtain even if that kind of trial has only one instance, and the very same chances should be assigned in that situation. But then there are well-defined chances even though that type of event occurs only once.

...

The upshot of this discussion is that chance is a *process* notion, rather than being entirely determined by features of the outcome to which the surface grammar of chance ascriptions assigns the chance. For if there can be a single-case chance of  $\frac{1}{2}$  for a coin to land heads on a toss even if there is only one actual toss, and it lands tails, then surely the chance cannot be fixed by properties of the outcome 'lands heads', as that outcome does not exist. The chance must rather be grounded in features of the process that can produce the outcome: the coin-tossing trial, including the mass distribution of the coin and the details of how it is tossed, in this case, plus the background conditions and laws that govern the trial. Whether or not an event happens by chance is a feature of the process that produced it, not the event itself. The fact that a coin lands heads does not fix that the coin landed heads by chance, because if it was simply placed heads up, as opposed to tossed in a normal fashion, we have the same outcome not by chance. Sometimes features of the outcome event cannot be readily separated from the features of its causes that characterise the process by means of which it was produced.

[Eag18]

To address this argument, we may pose the question: if we knew enough about the mechanics of the toss, could we predict its outcome? The answer is yes, we can- the Bell Labs statistician Persi Diaconis has built a coin tossing machine that reliably produces heads or tails [Kes04]. We know that tightly controlling the mechanics of a coin toss allows us to treat this system as entirely deterministic, without any significant element of chance in the outcome. A coin toss is only chancy when the human doing it does not have full control over the mechanical parameters of the process. Conceptually, there is no *a priori* reason why a coin-tosser should not be able to regularise the angular momentum of their thumb-flick by training with a strain gauge, place the coin on a stable surface in a wind-free box allowing flicking, and achieve the same effect as the coin-tossing machine. In this case, Eagle's suggestion that "[s]ometimes features of the outcome event cannot be readily separated from the features of its causes that characterise the process" seems obviously wrong- the "chancy" element of coin tossing is fully separable from the rest of the coin tossing process, and replaceable with a non-chancy component.

If the foregoing argument is correct, it seems that the coin toss is an example of the epistemic, rather than ontological, dimension of chance. The process appears to be chancy, or random, because the human tossing the coin is not able to precisely control the mechanical parameters of the process. Indeed, as Diaconis notes, these epistemically-limited tossers do not actually produce unbiased random outcomes- human coin tosses come up as they were started slightly more often than with the obverse face [DHM07].

Eagle's second example of an "objective single-case chance", is the decay of a radioactive atom. There is no known physical process whose parameters are thought to define the lifetime of individual radioactive atoms, in the way that there is a well-specified physical process that produces a particular coin toss outcome. This is an appeal to the Copenhagen theoretical principle that it is *a priori* impossible

to predict the lifetimes of individual atoms. As appeals to quantum theory to ground “objective chance” in biological processes are becoming more common, let us consider whether a quantum theoretical explanation might plausibly underpin the “stochasticity” of RPC fate specification.

#### 16.1.0.3 Can macromolecular chanciness be rooted in quantum indeterminacy?

Eagle suggests that, because the Copenhagen interpretation of quantum physics has wide currency among physicists, the theory’s implied indeterminacy of physical phenomena at the quantum level could ground “objective chance”. While common, this argument downplays the fact that quantum theory is not a homogenous scientific tradition. The conventional history of mid 20th-century quantum theory holds that John Stewart Bell, in the demonstration of his famous inequality, conclusively proved that deterministic (so-called “hidden variable”) theories of quantum mechanics were incorrect. As demonstrated in another section of the publication Eagle’s argument appears in, this is an incomplete view. Bohmian theorist Sheldon Goldstein shows that Bell was an advocate of deterministic Bohmian mechanical theory, and thought his famous inequality demonstrated that quantum phenomena could not be *local*, not that they could not be *deterministic* [Gol17].

Indeed, Bohmian quantum mechanics are fully deterministic, describe all of the same phenomena as Copenhagen, and in several cases resolve problems that orthodox quantum theory cannot [Gol17]. We are not, therefore, facing unanimous expert consensus that there is objective chance at the quantum level. We rather have a situation where physical phenomena are described by two different theory-sets, one of which takes its statistical generalisations to be descriptions of ontological indeterminacy (Copenhagen), and the other to reflect epistemic uncertainty about a determinate universe (Bohm). Moreover, there is no obvious reason to prefer the Copenhagen approach, given that the Bohmian theory explains Copenhagen’s paradoxical results “without further ado” [Gol17], deals with empirically verified phenomena of physical and biological interest that Copenhagen does not (eg. electron tunneling), and was the preferred approach of Bell himself.

Therefore, Eagle’s premise fails. There is no reason to suppose that quantum theoretical models that posit objective chance are good evidence for the *reality* of objective chance. Moreover, there are good reasons to suppose that the converse is true. In sum, then, we must reject Harris’ argument to the extent it argues for *objective, ontological* chance, since the arguments for the existence of both single-case objective chance or quantum chance are weak and biologically irrelevant. Clearly, however, the *epistemic* dimension of chance is in play here.

#### 16.1.0.4 Randomness in RPC fate specification

Having dealt with how the concept of chance might apply to Harris’ SMME above, let us consider how the term “random” might relate to the process of RPC fate specification and differentiation. As introduced earlier, the technical meaning of “randomness” pertains to sequences of process outcomes. The process outcomes Harris is concerned with are the temporally-arranged fate outcomes of some particular RPC lineage. Therefore, we can ask whether these sequences are likely to be algorithmically random.

The structure of the SMME models precludes algorithmic randomness. That is, the sequence of outcomes has a structure that can be meaningfully compressed by rules which produce typical RPC fate outcomes (Harris’ mathematical models are such rule sets). One might object that Harris’ meaning is that the particular rules which give rise to cellular fate “choice” in his models involve random number generation. In this case, the claim is trivially about the model and not about the sequence of outcomes

that is actually observed in zebrafish eyes. Indeed, all of Harris' later models assume a tripartite temporal structure to the differentiation process<sup>3</sup>. Therefore, the output of the SMME models themselves demonstrate that RPC fate specification is not an algorithmically random process. One could quantify the Kolmogorov complexity of short lineage outcome sequences in order to ask *how* random the sequence is [SZDG14], but this would not establish randomness as an explanation for variability, but rather would use randomness to describe it.

We should further note that the question of how predictably ordered, i.e. non-random processes like fate specification arise in biological systems is a fundamental question of the biological sciences. In many ways, then, it is the extent to which variable sequences of outcomes like RPC fate specification *depart* from algorithmic randomness which of interest when we are asking questions like “how does the ordered structure of a retina arise from RPC activity?”

#### 16.1.0.5 Summary: “Stochastic” or “variable”?

Above, I argue that the RPC fate specification process is not objectively chancy (since objective chance is an empirically unsupported concept), nor algorithmically random (since the sequence of RPC lineage outcomes is structured and therefore non-random). What, then, should we make of the argument that this process is “stochastic”? Let us consider the forceful argument of the great Bayesian statistician Edwin Thompson Jaynes:

“Belief in the existence of ‘stochastic processes’ in the real world; i.e. that the property of being ‘stochastic’ rather than ‘deterministic’ is a real physical property of a process, that exists independently of human information, is [an] example of the mind projection fallacy: attributing one’s own ignorance to Nature instead. The current literature of probability theory is full of claims to the effect that a ‘Gaussian random process’ is fully determined by its first and second moments<sup>4</sup>. If it were made clear that this is only the defining property for an abstract mathematical model, there could be no objection to this; but it is always presented in verbiage that implies that one is describing an objectively true property of a real physical process. To one who believes such a thing literally, there could be no motivation to investigate the causes more deeply than noting the first and second moments, and so the real processes at work might never be discovered. This is not only irrational because one is throwing away the very information that is essential to understand the physical process; if carried into practice it can have disastrous consequences. Indeed, there is no such thing as a ‘stochastic process’ in the sense that the individual events have no specific causes.” [JBE03]

It is, thus, important to emphasize that the utility of statistical modelling techniques should not be taken to suggest that the modelled phenomenon is actually, i.e. irreducibly, random and without causal structure. When speaking of biological “randomness” or “stochasticity”, biologists rarely precisely define what is meant by these terms. This vagueness sometimes arises from, or results in, a theoretical deficit where properties of statistical models are understood to directly reflect the system being modelled; the scientist has failed to heed Korbzysky’s dictum insisting that “a map *is not* the territory it represents, but, if correct, it has a *similar structure* to the territory, which accounts for its usefulness” [Kor05] (italics in original). The “structural similarity” here is between the model’s outcomes and the collection of actually-observed population outcomes, *not* the underlying biological process giving rise to measured outcomes. I conclude by suggesting that this particular example applies to all such explanations in

---

<sup>3</sup>That is, an early bias in RGC production is produced in these models by the a priori commitment to a “rule” which results in early RGC production.

<sup>4</sup>That is, the mean and the variance of the Normal Gaussian model are said to determine the process being modeled.

the biological sciences. There is presently no good reason to accept scientific explanations rooted in chance. As for explanations rooted in randomness, to be credible, these must actually estimate the degree of randomness present in the relevant outcomes, show how it produces the outcomes, and explain departures from pure incompressibility.

## 16.2 Macromolecular mechanistic explanations in the Systems era

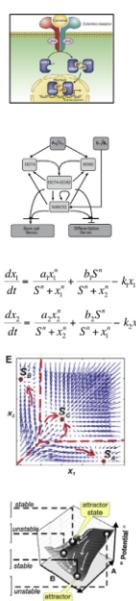
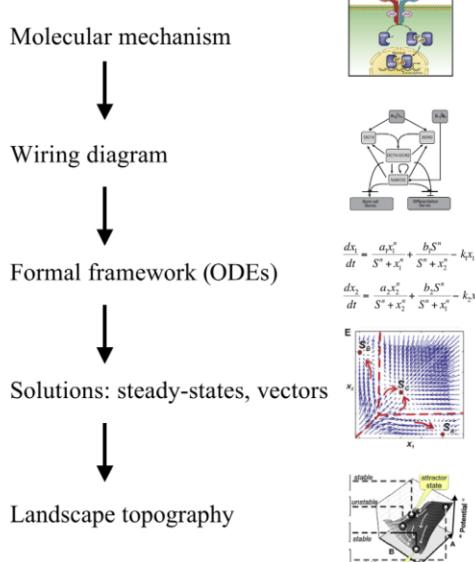


Figure 16.1: **Cellular systems model-construction.**

Excerpted from [Fag15, p.7], under CC-BY-NC-ND 4.0. The system of equations and subsequent steps are based on a 2-element wiring diagram.

of Medicine graduate pharmacology seminars in the early 2000s routinely included dark warnings about the collapse of antidepressant effect sizes over time, and the need for new statistical approaches. Reports that most biomedical research results are not replicable were met with a sort of palpable relief [Ioa05], if not much practical change; at least the growing welter of conflicting results could be explained this way, rather than risking interpretative collapse. Ultimately, the power of the macromolecular mechanism in pharmacology proved overstated; the productivity lull of the 2000-2010 era has been overcome not by rational mechanistic design of traditional small molecules, but by an influx of new classes of drugs, often with unknown or poorly characterised mechanisms of action [Mun19]. The arrival of new statistics has not helped matters much; serious Bayesian analyses tend to doubt whether medical pharmacological

The data chapters of this thesis are concerned with two issues in scientific practice: the comparison of models with different structures, and their interpretation. I began my career in stem cell biology with a background in molecular pharmacology, and the explanatory worldview that training inscribed. For me, biological explanation was about elucidating mechanisms, which consisted of descriptions of macromolecules and their accessory small molecule messengers interacting. "Models," in the sense of numerical simulations susceptible to formal statistical testing, were only the formal encoding of a body of knowledge that was first proved out at the bench, in interventional experiments. That these results could be encoded by a system of differential equations only confirmed the rigour of the original analysis, which composed the mechanistic explanation in the first place. That engineers should bring their numerical techniques into our laboratories, only to have these analyses confirm the plain-sense interpretations of the benchworkers producing the data, served to reinforce the sense that the pharmacological approach was fundamentally the correct one. It hardly ever entered our minds that model comparison was not occurring at all; it seemed that null hypotheses had already been slain by the flurry of t-tests and ANOVAs applied to the underlying datasets before the eggheads showed up.

That there were problems with this approach was already apparent by the beginning of my graduate education; Faculty

and students in the field were well aware of the limitations of the mechanistic paradigm. The collapse of antidepressant effect sizes over time, and the need for new statistical approaches. Reports that most biomedical research results are not replicable were met with a sort of palpable relief [Ioa05], if not much practical change; at least the growing welter of conflicting results could be explained this way, rather than risking interpretative collapse. Ultimately, the power of the macromolecular mechanism in pharmacology proved overstated; the productivity lull of the 2000-2010 era has been overcome not by rational mechanistic design of traditional small molecules, but by an influx of new classes of drugs, often with unknown or poorly characterised mechanisms of action [Mun19]. The arrival of new statistics has not helped matters much; serious Bayesian analyses tend to doubt whether medical pharmacological

interventions are effective at all in most cases [Ste18]. Still, by the time I had left for greener pastures with the stem cell biologists at the new Department of Cell and Systems Biology, it was clear that the pharmacological view of biological explanation had won significant discursive battles.

A fascinating artefact of this discursive victory is present in the best available study on the use of mechanistic explanations in stem cell biology, Melinda Fagan’s “Philosophy of stem cell biology: knowledge in flesh and blood”. In concluding chapters outlining the future of stem cell biology, Fagan provides a diagrammatic summary of what she takes to be the field’s consensus on its general direction into its future “Systems Biology” incarnation, which I reproduce here in Figure 16.1.

The perspicacious reader will observe that this is nothing other than the pharmacological view I outline above: the practice of molecular biology is the elucidation of mechanisms, the practice of systems biology is the formalization of these mechanisms as systems of differential equations. What was most peculiar about the presence of this view is that it makes no account of the Simple Stochastic Model, which receives significant coverage elsewhere in Fagan’s book. This was particularly important to me because I was trying to understand how to use Harris’ models to explain my results. What Harris was doing was clearly intended to be both a mechanistic explanation, in the sense that it eventually nominated a pair of particular transcription factors as the causative agents, and also a Systems explanation, in the sense that it relied, for persuasive effect, on complex numerical modelling techniques. It was formulated in SSM terms, however, not as a SODE, and critically was about the metaphysical interpretation of a mathematical construct, not about the joint action of the named macromolecules, as Fagan suggests we should understand mechanistic explanations [Fag13]<sup>5</sup>.

Moreover, much of the sophisticated literature on mathematical models of stem cell variability are based on interpretations of dynamical systems theory or chaos theory [FK12, HLZQ17] that deal with the macromolecular constituents of cells as bulk expression values; that is, a protein may be represented in one of these models as a coordinate on an dimension representing the amount of the protein expressed. It is extraordinarily rare for these models to be very concerned about crystal structures, phosphorylation states, or any of the other mechanistic details the pharmacological view is replete with. Other approaches rely on control systems theory [SK15, YSK15], others happily use technical information-theoretic terms like “noise” without any effort to define or measure it [CHB<sup>+</sup>08], still others treat the expression of macromolecules as the endpoints of tissue-mechanical forces [PLM<sup>+</sup>17], and so on. The actual variety of “systems” explanatory strategies found “in the field” is bewildering; to become proficient in even one is a years-long project. As I began to grasp the sheer number of technical and theoretical subdomains implicated in these explanations, the full force of Nicholas Reschers’ argument on this point became apparent:

The ramifications and implications of philosophical contentions do not respect a discipline’s taxonomic boundaries. And we all too easily risk losing sight of this interconnectedness when we pursue the technicalities of a narrow subdomain. In actuality, the stance we take on questions in one domain will generally have substantial implications and ramifications for very different issues in other, seeming unrelated domains. And this is exactly why systematization is so important in philosophy - because the way we *do* answer some questions will have limiting repercussions for the way we *can* answer others. We cannot emplace our philosophical convictions into conveniently delineated compartments in the comfortable expectation that what we maintain in one area of the field will have no unwelcome implications for what we

---

<sup>5</sup>Arguably, the problem here is with the attempt to reify SODE models of macromolecular mechanisms as the standard form for these explanations; however, many relevant explanations, including Harris’, and those developed in this thesis, do not descend to the “basement level” of macromolecular interaction at all.

are inclined to maintain in others. [Res05, p.97]

Indeed, the sudden injection of the philosophical contents of the “complexity sciences” into biological discourse felt like an invasion; suddenly, it was not very clear what a mechanism or a biological explanation might consist of after all. Michel Morange has argued persuasively that molecular biology is essentially mature [Mor03] and that “systems” explanations consist mainly in the complementation of ordinary molecular practice with sophisticated mathematical models [Mor08], while acknowledging the increasing space available for the entry of physical explanations in molecular biology [Mor11]. This account is appealing, but it belies the chaotic nature of the recent scientific scene, and cannot make sense of why there are so many contending, often quite incompatible, views on how to explain cellular and molecular biological phenomena, and why so few of those views include any idea on *how those explanations might be tested against one another*.

### 16.2.1 The Feyerabendian modeller

Chapter 2 cites Paul Feyerabend in establishing that scientific theories exert their cognitive effects on us by making metaphysical claims about reality. This point informs the model-analysis in that chapter, showing how the models of the Stochastic Mitotic Mode Explanation (SMME) become progressively more abstract and restricted to the mitotic mode concept. Mitotic mode is not a physical existent, but rather a conceptual compound of the fate of multiple cells- it is in this sense plainly meta-physical. Feyerabend has a number of essential insights on scientific metaphysics, which, when judiciously understood, allow us to make sense of what is happening within stem cell biology and more broadly, in the “systems biology” era. The central, seminal insight of Feyerabend’s *Against Method* is that the observed historical succession of scientific theories occurs by counterpositional advancement of incompatible opposing theories. Only counterinductive comparisons *between* theories are capable of showing up their implicit assumptions and allowing them to be challenged. Feyerabend explains:

... it emerges that the evidence that might refute a theory can often be unearthed only with the help of an incompatible alternative: the advice (which goes back to Newton and which is still very popular today) to use alternatives only when refutations have already discredited the orthodox theory puts the cart before the horse. Also, some of the most important formal properties of a theory are found by contrast, and not by analysis. A scientist who wishes to maximize the empirical content of the views he holds and who wants to understand them as clearly as he possibly can must therefore introduce other views; that is, he must adopt a *pluralistic methodology*. He must compare ideas with other ideas rather than with ‘experience’ and he must try to improve rather than discard the views that have failed in the competition. [Fey93, p.20]

Feyerabend is interested in debunking the notion that science differs from other discursive social practices by showing that no universal method vouchsafes the progressive replacement of earlier, inferior theories with later, improved ones. He famously establishes that Galileo used a form of metaphysical propaganda, particularly in the ad hoc invocation of the now-discredited concept of circular inertia, to establish the credibility of the Copernican model against its orthodox Aristotlean competitor, championed by the Catholic Church [Fey93]. Although we commonly think of the Copernican Revolution as the replacement of an obviously defective theological explanation by a properly formed theory from the empirical sciences, Feyerabend shows that this conceals the actual means by which Galileo makes his persuasive case for the (itself badly defective) Copernican model. Galileo’s theory substantially contradicted the available evidence, erroneously asserted the reliability of some telescopic observations and

discredited others<sup>6</sup>, made extensive use of ad hoc hypotheses, and was advanced by propagandistic and even dishonest means. Much of this was unavoidable. Ad hoc hypotheses are necessary for new theories, because the auxiliary sciences associated with them have not been developed. Scientific development is intrinsically uneven, obligating the use of these makeshift theoretical devices. Without a certain level of rhetorical sleight of hand on Galileo's part, the Copernican program would have succumbed to the greater development and argumentative weight of the Scholastic tradition. As Feyerabend notes, if any of the typically suggested criteria for a universal scientific method were applied, the Church would have won the debate, and we might still have an Aristotlean cosmology.

Because of his concern to attack what he sees as the overweening social influence of scientific and technical experts, Feyerabend puts emphasis on the discursive process of establishing orthodoxy within a field, and the commonalities between scientific and nonscientific domains [Fey75, Fey81, Fey93]. That is, the practice of the natural sciences are susceptible to the same irreducibly subjective and historically contingent forces as other domains of human culture. In this sense there is nothing special about scientific practice that shields its conclusions from error introduced by these means. These extra-scientific forces have significant effects on the forms scientific models take and the interpretations they are given, and it helps to remember this when reading modelling papers mainly published to keep the lights on. Still, I suggest that it is important not to interpret these arguments too pessimistically. Feyerabend genuinely wanted to promote what he called "Open Exchange" between different scientific and metaphysical traditions, centered around a noncoercive exchange of, in effect, models and standards for judging them [Fey93]. He also may not have taken heed of the unique aspects of the natural sciences; statistical descriptions of collections of outcomes and of uncertainty, and a self-reinforcing dynamic of scaffolding methodological complexity, are both unique to the modern natural sciences.

Ultimately, the value in Feyerabend's perspective is seeing that scientists routinely operate as epistemological anarchists, finding what works in their local institutional surroundings, and that we must expect that this will be the case. In other words, Fagan's (extremely carefully argued) conceptualization of the orthodox molecular mechanism, as found in stem cell biology, is useless for interpreting Harris' theories. This is so because Harris does not care about adhering to this standard of orthodoxy. The field has, in typical anarchic fashion, begun adapting models and numerical techniques from a panoply of engineering, physics, statistical mechanics, AI, etc. subdisciplines, more or less willy nilly, and often without any sort of analysis of the adequacy of the model relative to alternatives. Moreover, these models pertain to a variety of levels of biological organisation, from the tissue down to subcellular nuclear compartments, and frequently do not refer in any concrete way to biological macromolecules.

I argue that, from this point of view, the macromolecular mechanistic explanation Fagan reifies is, in fact, already dead. The era of painstaking, effector-by-effector construction of macromolecular pathways, to be virtually enshrined in their ultimate SODE incarnation, perhaps ultimately to be assembled into some Monodian model cell-as-cybernetic-factory, is over, if it ever existed. This is not to say that Morange is wrong- as he asserts, molecular biology is not dead, it has not been replaced by "systems biology". In practice, however, the traditional form of molecular biological explanation is rapidly being replaced by explanatory forms from other fields, and it is not always clear that this has been salutary. For Feyerabend, science, like other human social practices consists of a variety of interacting traditions with differing assumptions, methods, sensory interpretations, and so on. The development of science is

---

<sup>6</sup>Galileo asserted that comets are optical illusions, for instance, since their non-circular orbits disconfirm the Copernican system, which insists on the circularity of orbital motion.

thus “not the interaction of a practice with something different and external, *but the development of one tradition under the impact of others.*” [Fey93, p.232]. The concept of “systems biology” is fundamentally an artefact of the impact of advanced statistics and statistical mechanics on biology. The resolution to the question of how this impact is mediated will determine the institutional and intellectual landscape that results from it.

To have any agency in this process, molecular biologists must be able to assess the theoretical contents that are being imported into our field for ourselves. If we do not, we cannot assess the impact of the metaphysical commitments of these theories on the rest of our theorising, as Rescher persuasively insists that we must. Indeed, it was Feyerabend’s perspective that led me to realise that there were at least three scientific orthodoxies hindering me from understanding the metaphysical contents of Harris’ theories. These were the mechanism-to-SODE pipeline instilled by my pharmacological training and recapitulated by Fagan, the unsupported insistence on the actual reality of chanciness by Copenhagen theorists, and the inadequate statistical approaches recommended by frequentist statisticians. By realising that any conceptual element could appear in a mechanistic explanation, by rejecting the objective reality of chance, and embracing the original formulation of statistical orthodoxy suggested by LaPlace (that is, Bayesian statistics), I was able to counter-inductively discover the fundamental problems with the SMME. In continuing to test models with fundamentally different structures, I attempt to improve older theories about linear stages of RPC activity in order to counterinductively weigh these against alternatives.

Moreover, by the conscious adoption of Bayesian methods from cosmology to address local problems in stem cell biology, I attempt to model Feyerabend’s Open Exchange between two scientific traditions. My selection of Skilling’s nested sampling system of inference is driven by my appreciation for its logical consistency, simplicity, deep roots in fundamental logic and information theory, and broad applicability to a wide range of problems. While Feyerabend is no doubt correct that, ultimately, the selection of criteria to distinguish between scientific theories must itself be subjective, the deep comportment of Skilling’s system with the fundamental human drive toward cognitive harmony [Res05] compels me to suggest that it may serve as a near-universal yardstick of the adequacy of scientific models with respect to their underlying datasets in the not-too-distant future.

# Chapter 17

## Code Appendix

### 17.1 SMME

Github repository: <https://github.com/mmattocks/SMME>

#### 17.1.1 setup\_project.py

---

```
1 """Copyright (c) 2005-2017, University of Oxford.
2 All rights reserved.
3
4 University of Oxford means the Chancellor, Masters and Scholars of the
5 University of Oxford, having an administrative office at Wellington
6 Square, Oxford OX1 2JD, UK.
7
8 This file is part of Chaste.
9
10 Redistribution and use in source and binary forms, with or without
11 modification, are permitted provided that the following conditions are
12 met:
13 * Redistributions of source code must retain the above copyright notice,
14   this list of conditions and the following disclaimer.
15 * Redistributions in binary form must reproduce the above copyright
16   notice,
17   this list of conditions and the following disclaimer in the
18   documentation
19   and/or other materials provided with the distribution.
20 * Neither the name of the University of Oxford nor the names of its
21   contributors may be used to endorse or promote products derived from
22   this
23   software without specific prior written permission.
```

```
21 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
22 → IS"
23 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
24 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
25 → PURPOSE
26 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
27 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
28 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
29 → SUBSTITUTE
30 GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
31 → INTERRUPTION)
32 HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
33 → STRICT
34 LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
35 → OUT
36 OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
37 → DAMAGE.

38 """
39
40 import os
41
42
43 def find_and_replace(filename, old_string, new_string):
44     # Read the file
45     f = open(filename, 'r')
46     file_contents = f.read()
47     f.close()
48
49     # Write to the file
50     f = open(filename, 'w')
51     f.write(file_contents.replace(old_string, new_string))
52     f.close()
53
54     return
55
56
57 def ask_for_response(question):
58     # Display the question
59     print(question)
60
61     # Define permitted yes/no answers
62     yes = {'yes', 'y', 'ye', ''}
63     no = {'no', 'n'}
```

```
57
58     # Take the lower case raw input
59     choice = raw_input().lower()
60
61     # Decide on the choice
62     if choice in yes:
63         return True
64     elif choice in no:
65         return False
66     else:
67         ask_for_response("Please respond with yes or no:")
68
69
70 def main():
71     # The absolute path to the project directory
72     path_to_project = os.path.dirname(os.path.realpath(__file__))
73
74     # Identify the name of the project
75     project_name = os.path.basename(path_to_project)
76
77     # Paths to the CMakeLists.txt files
78     base_cmakelists = os.path.join(path_to_project, 'CMakeLists.txt')
79     apps_cmakelists = os.path.join(path_to_project, 'apps',
80                                    'CMakeLists.txt')
80     test_cmakelists = os.path.join(path_to_project, 'test',
81                                    'CMakeLists.txt')
81
82     # Perform the find-and-replace tasks to update the template project
83     find_and_replace(base_cmakelists,
84                       'chaste_do_project(template_project', 'chaste_do_project(' +
85                       project_name)
84     find_and_replace(apps_cmakelists,
85                       'chaste_do_apps_project(template_project',
86                       'chaste_do_apps_project(' + project_name)
85     find_and_replace(test_cmakelists,
86                       'chaste_do_test_project(template_project',
87                       'chaste_do_test_project(' + project_name)
86
87     # Amend the components
88     components_list = []
89
90     if ask_for_response("Does this project depend on the cell_based
91                         component? [Y/n] "):
```

```

91     components_list.append('cell_based')

92
93     if ask_for_response("Does this project depend on the crypt component?
94         ↪ [Y/n] "):
95         components_list.append('crypt')

96     if ask_for_response("Does this project depend on the heart component?
97         ↪ [Y/n] "):
98         components_list.append('heart')

99     if ask_for_response("Does this project depend on the lung component?
100        ↪ [Y/n] "):
101         components_list.append('lung')

102     # If the list is non-empty, replace the default components
103     if components_list:
104         components_string = ' '.join(components_list)
105         default_components = 'continuum_mechanics global io linalg mesh
106             ↪ ode pde'

107         find_and_replace(base_cmakelists, default_components,
108                         ↪ components_string)

109
110 if __name__ == "__main__":
111     main()

```

---

### 17.1.2 /apps/src/BoijeSimulator.cpp

---

```

1 #include <iostream>
2 #include <string>
3
4 #include <cxxtest/TestSuite.h>
5 #include "ExecutableSupport.hpp"
6 #include "Exception.hpp"
7 #include "PetscTools.hpp"
8 #include "PetscException.hpp"
9
10 #include "BoijeCellCycleModel.hpp"
11 #include "OffLatticeSimulationPropertyStop.hpp"
12
13 #include "AbstractCellBasedTestSuite.hpp"

```

```
14
15 #include "WildTypeCellMutationState.hpp"
16 #include "TransitCellProliferativeType.hpp"
17 #include "DifferentiatedCellProliferativeType.hpp"
18 #include "BoijeRetinalNeuralFates.hpp"
19
20 #include "CellsGenerator.hpp"
21 #include "HoneycombMeshGenerator.hpp"
22 #include "NodesOnlyMesh.hpp"
23 #include "NodeBasedCellPopulation.hpp"
24 #include "VertexBasedCellPopulation.hpp"
25
26 #include "ColumnDataWriter.hpp"
27
28 int main(int argc, char *argv[])
29 {
30     ExecutableSupport::StartupWithoutShowingCopyright(&argc, &argv);
31     //main() returns code indicating sim run success or failure mode
32     int exit_code = ExecutableSupport::EXIT_OK;
33
34     if (argc != 13)
35     {
36         ExecutableSupport::PrintError(
37             "Wrong arguments for simulator.\nUsage (replace ◇ with
38             → values, pass bools as 0 or 1):\n BoijeSimulator
39             → <directoryString> <filenameString>
40             → <outputModeUnsigned(0=counts,1=events,2=sequence)>
41             → <debugOutputBool> <startSeedUnsigned>
42             → <endSeedUnsigned> <endGenerationUnsigned>
43             → <phase2GenerationUnsigned> <phase3GenerationUnsigned>
44             → <pAtoh7Double(0-1)> <pPtf1aDouble(0-1)>
45             → <pngDouble(0-1)>",
46             true);
47         exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
48         return exit_code;
49     }
50
51
52     /*****
53     * SIMULATOR PARAMETERS
54     *****/
55
56     std::string directoryString, filenameString;
57     int outputMode; //0 = counts; 1 = mitotic mode events; 2 = mitotic
58     → mode sequence sampling
```

```
48     bool debugOutput;
49     unsigned startSeed, endSeed, endGeneration, phase2Generation,
50         → phase3Generation;
51     double pAtoh7, pPtf1a, png; //stochastic model parameters
52
53     //PARSE ARGUMENTS
54     directoryString = argv[1];
55     filenameString = argv[2];
56     outputMode = std::stoi(argv[3]);
57     debugOutput = std::stoul(argv[4]);
58     startSeed = std::stoul(argv[5]);
59     endSeed = std::stoul(argv[6]);
60     endGeneration = std::stoul(argv[7]);
61     phase2Generation = std::stoul(argv[8]);
62     phase3Generation = std::stoul(argv[9]);
63     pAtoh7 = std::stod(argv[10]);
64     pPtf1a = std::stod(argv[11]);
65     png = std::stod(argv[12]);
66
66     ****
67     * PARAMETER/ARGUMENT SANITY CHECK
68     ****/
69     bool sane = 1;
70
71     if (outputMode != 0 && outputMode != 1 && outputMode != 2)
72     {
73         ExecutableSupport::PrintError(
74             "Bad outputMode (argument 3). Must be 0 (counts) 1
75             → (mitotic events) or 2 (sequence sampling)");
76         sane = 0;
77     }
78
79     if (endSeed < startSeed)
80     {
81         ExecutableSupport::PrintError("Bad start & end seeds (arguments,
82             → 5, 6). endSeed must not be < startSeed");
83         sane = 0;
84     }
85
86     if (endGeneration ≤ 0)
87     {
88         ExecutableSupport::PrintError("Bad endGeneration (argument 7).
89             → endGeneration must be > 0");
```

```
87         sane = 0;
88     }
89
90     if (phase3Generation < phase2Generation)
91     {
92         ExecutableSupport::PrintError(
93             "Bad phase2Generation or phase3Generation (arguments 8,
94             ↪ 9). phase3Generation must be > phase2Generation. Both
95             ↪ must be >0");
96         sane = 0;
97     }
98
99     if (pAtoh7 < 0 || pAtoh7 > 1)
100    {
101        ExecutableSupport::PrintError("Bad pAtoh7 (argument 10). Must be
102            ↪ 0-1");
103        sane = 0;
104    }
105
106    if (pPtf1a < 0 || pPtf1a > 1)
107    {
108        ExecutableSupport::PrintError("Bad pPtf1a (argument 11). Must be
109            ↪ 0-1");
110        sane = 0;
111    }
112
113    if (png < 0 || png > 1)
114    {
115        ExecutableSupport::PrintError("Bad png (argument 12). Must be
116            ↪ 0-1");
117        sane = 0;
118    }
119
120    if (sane == 0)
121    {
122        ExecutableSupport::PrintError("Exiting with bad arguments. See
123            ↪ errors for details");
124        exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
125        return exit_code;
126    }
127
128    /*****
129     * SIMULATOR OUTPUT SETUP
130     */
```

```

124     *****/
125
126 //Set up singleton LogFile
127    LogFile* p_log = LogFile::Instance();
128     p_log->Set(0, directoryString, filenameString);
129
130     ExecutableSupport::Print("Simulator writing file " + filenameString +
131     " to directory " + directoryString);
132
133 //Log entry counter
134     unsigned entry_number = 1;
135
136 //Write appropriate headers to log
137     if (outputMode == 0) *p_log << "Entry\tSeed\tCount\n";
138     if (outputMode == 1) *p_log << "Time (hpf)\tSeed\tCellID\tMitotic
139     Mode (0=PP;1=PD;2=DD)\n";
140     if (outputMode == 2) *p_log << "Entry\tSeed\tSequence\n";
141
142 //Instance RNG
143     RandomNumberGenerator* p_RNG = RandomNumberGenerator::Instance();
144
145 //Initialise pointers to relevant singleton ProliferativeTypes and
146     Properties
147     MAKE_PTR(WildTypeCellMutationState, p_state);
148     MAKE_PTR(TransitCellProliferativeType, p_Mitotic);
149     MAKE_PTR(DifferentiatedCellProliferativeType, p_PostMitotic);
150     MAKE_PTR(RetinalGanglion, p_RGC_fate);
151     MAKE_PTR(AmacrineHorizontal, p_AC_HC_fate);
152     MAKE_PTR(ReceptorBipolar, p_PR_BC_fate);
153     MAKE_PTR(CellLabel, p_label);
154
155     *****/
156     /* SIMULATOR SETUP & RUN
157     *****/
158
159 //iterate through supplied seed range, executing one simulation per seed
160     for (unsigned seed = startSeed; seed <= endSeed; seed++)
161     {
162         if (outputMode == 2) *p_log << entry_number << "\t" << seed <<
163             "\t"; //write seed to log - sequence written by
164             cellcyclemodel objects
165
166         //initialise pointer to debugWriter

```

```

162 ColumnDataWriter* debugWriter;
163
164 //initialise SimulationTime (permits cellcyclemodel setup)
165 SimulationTime::Instance()→SetStartTime(0.0);
166
167 //Reseed the RNG with the required seed
168 p_RNG→Reseed(seed);
169
170 //Initialise a HeCellCycleModel and set it up with appropriate
171 // TiL values
172 BoijeCellCycleModel* p_cycle_model = new BoijeCellCycleModel;
173
174 if (debugOutput)
175 {
176     //Pass ColumnDataWriter to cell cycle model for debug output
177     boost::shared_ptr<ColumnDataWriter> p_debugWriter(
178         new ColumnDataWriter(directoryString, filenameString
179             + "DEBUG_" + std::to_string(seed), false, 10));
180     p_cycle_model→EnableModelDebugOutput(p_debugWriter);
181     debugWriter = &p_debugWriter;
182 }
183
184 //Setup lineages' cycle model with appropriate parameters
185 p_cycle_model→SetDimension(2);
186 p_cycle_model→SetPostMitoticType(p_PostMitotic);
187
188 //Setup vector containing lineage founder with the properly set
189 // up cell cycle model
190 std::vector<CellPtr> cells;
191 CellPtr p_cell(new Cell(p_state, p_cycle_model));
192 p_cell→SetCellProliferativeType(p_Mitotic);
193 p_cycle_model→SetModelParameters(phase2Generation,
194     phase3Generation, pAtoh7, pPtf1a, png);
195 p_cycle_model→SetSpecifiedTypes(p_RGC_fate, p_AC_HC_fate,
196     p_PR_BC_fate);
197 if (outputMode == 2)
198     p_cycle_model→EnableSequenceSampler(p_label);
199 if (outputMode == 2) p_cell→AddCellProperty(p_label);
200 p_cell→InitialiseCellCycleModel();
201 cells.push_back(p_cell);
202
203 //Generate 1x1 mesh for single-cell colony
204 HoneycombMeshGenerator generator(1, 1);

```

```

199     MutableMesh<2, 2>* p_generating_mesh = generator.GetMesh();
200     NodesOnlyMesh<2> mesh;
201     mesh.ConstructNodesWithoutMesh(*p_generating_mesh, 1.5);
202
203     //Setup cell population
204     NodeBasedCellPopulation<2>* cell_population(new
205         → NodeBasedCellPopulation<2>(mesh, cells));
206
207     //Setup simulator & run simulation
208     boost::shared_ptr<OffLatticeSimulationPropertyStop<2>>
209         → p_simulator(
210             new
211                 → OffLatticeSimulationPropertyStop<2>(*cell_population));
212     p_simulator→SetStopProperty(p_Mitotic); //simulation to stop if
213         → no mitotic cells are left
214     p_simulator→SetDt(0.25);
215     p_simulator→SetEndTime(endGeneration);
216     p_simulator→SetOutputDirectory("UnusedSimOutput" +
217         → filenameString); //unused output
218     p_simulator→Solve();
219
220     //Count lineage size
221     unsigned count = cell_population→GetNumRealCells();
222
223     if (outputMode == 0) *p_log << entry_number << "\t" << seed <<
224         → "\t" << count << "\n";
225     if (outputMode == 2) *p_log << "\n";
226
227     //Reset for next simulation
228     SimulationTime::Destroy();
229     delete cell_population;
230     entry_number++;
231
232     if (debugOutput)
233     {
234         debugWriter→Close();
235     }
236
237     p_RNG→Destroy();
238     LogFile::Close();
239
240
241 }
```

```

236     return exit_code;
237 }
238 ;

```

---

### 17.1.3 /apps/src/GomesSimulator.cpp

```

1 #include <iostream>
2 #include <string>
3
4 #include <cxxtest/TestSuite.h>
5 #include "ExecutableSupport.hpp"
6 #include "Exception.hpp"
7 #include "PetscTools.hpp"
8 #include "PetscException.hpp"
9
10 #include "GomesCellCycleModel.hpp"
11 #include "OffLatticeSimulationPropertyStop.hpp"
12
13 #include "AbstractCellBasedTestSuite.hpp"
14
15 #include "WildTypeCellMutationState.hpp"
16 #include "TransitCellProliferativeType.hpp"
17 #include "DifferentiatedCellProliferativeType.hpp"
18 #include "GomesRetinalNeuralFates.hpp"
19
20 #include "CellsGenerator.hpp"
21 #include "HoneycombMeshGenerator.hpp"
22 #include "NodesOnlyMesh.hpp"
23 #include "NodeBasedCellPopulation.hpp"
24 #include "VertexBasedCellPopulation.hpp"
25
26 #include "ColumnDataWriter.hpp"
27
28 int main(int argc, char *argv[])
29 {
30     ExecutableSupport::StartupWithoutShowingCopyright(&argc, &argv);
31     //main() returns code indicating sim run success or failure mode
32     int exit_code = ExecutableSupport::EXIT_OK;
33
34     if (argc != 15)
35     {
36         ExecutableSupport::PrintError(

```

```

37      "Wrong arguments for simulator.\nUsage (replace < with
38      ↳ values, pass bools as 0 or 1):\n GomesSimulator
39      ↳ <directoryString> <filenameString>
40      ↳ <outputModeUnsigned(0=counts,1=events,2=sequence)>
41      ↳ <debugOutputBool> <startSeedUnsigned>
42      ↳ <endSeedUnsigned> <endTimeDoubleHours>
43      ↳ <cellCycleNormalMeanDouble>
44      ↳ <cellCycleNormalStdDouble> <pPPDouble(0-1)>
45      ↳ <pPDDouble(0-1)> <pBCDouble(0-1)> <pACDouble(0-1)>
46      ↳ <pMGDouble(0-1)>",
47      true);
48
49      exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
50
51      return exit_code;
52  }

53
54  //*****
55  * SIMULATOR PARAMETERS
56  *****/
57
58  std::string directoryString, filenameString;
59  int outputMode; //0 = counts; 1 = mitotic mode events; 2 = mitotic
60  ↳ mode sequence sampling
61  bool debugOutput;
62  unsigned startSeed, endSeed;
63  double endTime;
64  double normalMu, normalSigma, pPP, pPD, pBC, pAC, pMG; //stochastic
65  ↳ model parameters

66
67  //PARSE ARGUMENTS
68  directoryString = argv[1];
69  filenameString = argv[2];
70  outputMode = std::stoi(argv[3]);
71  debugOutput = std::stoul(argv[4]);
72  startSeed = std::stoul(argv[5]);
73  endSeed = std::stoul(argv[6]);
74  endTime = std::stod(argv[7]);
75  normalMu = std::stod(argv[8]);
76  normalSigma = std::stod(argv[9]);
77  pPP = std::stod(argv[10]);
78  pPD = std::stod(argv[11]);
79  pBC = std::stod(argv[12]);
80  pAC = std::stod(argv[13]);
81  pMG = std::stod(argv[14]);

```

```

69  /*****
70   * PARAMETER/ARGUMENT SANITY CHECK
71   *****/
72   bool sane = 1;
73
74   if (outputMode != 0 && outputMode != 1 && outputMode != 2)
75   {
76       ExecutableSupport::PrintError(
77           "Bad outputMode (argument 3). Must be 0 (counts) 1
78           ↳ (mitotic events) or 2 (sequence sampling)");
79       sane = 0;
80   }
81
82   if (endSeed < startSeed)
83   {
84       ExecutableSupport::PrintError("Bad start & end seeds (arguments,
85           ↳ 5, 6). endSeed must not be < startSeed");
86       sane = 0;
87   }
88
89   if (endTime <= 0)
90   {
91       ExecutableSupport::PrintError("Bad endTime (argument 7). endTime
92           ↳ must be > 0");
93       sane = 0;
94   }
95
96   if (normalMu <= 0 || normalSigma <= 0)
97   {
98       ExecutableSupport::PrintError("Bad cell cycle normal mean or std
99           ↳ (arguments 8, 9). Must be >0");
100      sane = 0;
101  }
102
103  if (pPP + pPD > 1 || pPP > 1 || pPP < 0 || pPD > 1 || pPD < 0)
104  {
105      ExecutableSupport::PrintError(
106          "Bad mitotic mode probabilities (arguments 10, 11). pPP +
107          ↳ pPD should be ≥0, ≤1, sum should not exceed 1");
108      sane = 0;
109  }

```

```

106     if (pBC + pAC + pMG > 1 || pBC > 1 || pBC < 0 || pAC > 1 || pAC < 0
107     → || pMG > 1 || pMG < 0)
108 {
109     ExecutableSupport::PrintError(
110         "Bad specification probabilities (arguments 12, 13, 14).
111         → pBC, pAC, pMG should be  $\geq 0$ ,  $\leq 1$ , sum should not
112         → exceed 1");
113     sane = 0;
114 }
115
116 if (sane == 0)
117 {
118     ExecutableSupport::PrintError("Exiting with bad arguments. See
119         → errors for details");
120     exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
121     return exit_code;
122 }
123
124 //*****
125 * SIMULATOR OUTPUT SETUP
126 *****/
127
128 //Set up singleton LogFile
129    LogFile* p_log = LogFile::Instance();
130     p_log→Set(0, directoryString, filenameString);
131
132 ExecutableSupport::Print("Simulator writing file " + filenameString +
133     " to directory " + directoryString);
134
135 //Log entry counter
136     unsigned entry_number = 1;
137
138 //Write appropriate headers to log
139     if (outputMode == 0) *p_log << "Entry\tSeed\tCount\n";
140     if (outputMode == 1) *p_log << "Time (hpf)\tSeed\tCellID\tMitotic
141         Mode (0=PP;1=PD;2=DD)\n";
142     if (outputMode == 2) *p_log << "Entry\tSeed\tSequence\n";
143
144 //Instance RNG
145     RandomNumberGenerator* p_RNG = RandomNumberGenerator::Instance();
146
147 //Initialise pointers to relevant singleton ProliferativeTypes and
148     → Properties

```

```

142 MAKE_PTR(WildTypeCellMutationState, p_state);
143 MAKE_PTR(TransitCellProliferativeType, p_Mitotic);
144 MAKE_PTR(DifferentiatedCellProliferativeType, p_PostMitotic);
145 MAKE_PTR(RodPhotoreceptor, p_RPh_fate);
146 MAKE_PTR(AmacrineCell, p_AC_fate);
147 MAKE_PTR(BipolarCell, p_BC_fate);
148 MAKE_PTR(MullerGlia, p_MG_fate);
149 MAKE_PTR(CellLabel, p_label);

150
151 /*****
152 * SIMULATOR SETUP & RUN
153 *****/
154
155 //iterate through supplied seed range, executing one simulation per seed
156 for (unsigned seed = startSeed; seed <= endSeed; seed++)
157 {
158     if (outputMode == 2) *p_log << entry_number << "\t" << seed <<
159         "\t"; //write seed to log - sequence written by
160         //cellcyclemodel objects
161
162     //initialise pointer to debugWriter
163     ColumnDataWriter* debugWriter;
164
165     //initialise SimulationTime (permits cellcyclemodel setup)
166     SimulationTime::Instance()→SetStartTime(0.0);
167
168     //Reseed the RNG with the required seed
169     p_rng→Reseed(seed);
170
171     //Initialise a HeCellCycleModel and set it up with appropriate
172     // TiL values
173     GomesCellCycleModel* p_cycle_model = new GomesCellCycleModel;
174
175     if (debugOutput)
176     {
177         //Pass ColumnDataWriter to cell cycle model for debug output
178         boost::shared_ptr<ColumnDataWriter> p_debugWriter(
179             new ColumnDataWriter(directoryString, filenameString
180                 + "DEBUG_" + std::to_string(seed), false, 10));
181         p_cycle_model→EnableModelDebugOutput(p_debugWriter);
182         debugWriter = &p_debugWriter;
183     }
184
185 }
```

```

181     //Setup lineages' cycle model with appropriate parameters
182     p_cycle_model→SetDimension(2);
183     p_cycle_model→SetPostMitoticType(p_PostMitotic);

184
185     //Setup vector containing lineage founder with the properly set
186     // up cell cycle model
187     std::vector<CellPtr> cells;
188     CellPtr p_cell(new Cell(p_state, p_cycle_model));
189     p_cell→SetCellProliferativeType(p_Mitotic);
190     p_cycle_model→SetModelParameters(normalMu, normalSigma, pPP,
191     ↵ pPD, pBC, pAC, pMG);
192     p_cycle_model→SetModelProperties(p_RPh_fate, p_AC_fate,
193     ↵ p_BC_fate, p_MG_fate);
194     if (outputMode == 2)
195         ↵ p_cycle_model→EnableSequenceSampler(p_label);
196     if (outputMode == 2) p_cell→AddCellProperty(p_label);
197     p_cell→InitialiseCellCycleModel();
198     cells.push_back(p_cell);

199
200     //Generate 1x1 mesh for single-cell colony
201     HoneycombMeshGenerator generator(1, 1);
202     MutableMesh<2, 2>* p_generating_mesh = generator.GetMesh();
203     NodesOnlyMesh<2> mesh;
204     mesh.ConstructNodesWithoutMesh(*p_generating_mesh, 1.5);

205
206     //Setup cell population
207     NodeBasedCellPopulation<2>* cell_population(new
208         ↵ NodeBasedCellPopulation<2>(mesh, cells));

209
210     //Setup simulator & run simulation
211     boost::shared_ptr<OffLatticeSimulationPropertyStop<2>>
212         ↵ p_simulator(
213             new
214                 ↵ OffLatticeSimulationPropertyStop<2>(*cell_population));
215     p_simulator→SetStopProperty(p_Mitotic); //simulation to stop if
216     ↵ no mitotic cells are left
217     p_simulator→SetDt(0.25);
218     p_simulator→SetEndTime(endTime);
219     p_simulator→SetOutputDirectory("UnusedSimOutput" +
220         ↵ filenameString); //unused output
221     p_simulator→Solve();

222
223     //Count lineage size

```

```

215     unsigned count = cell_population->GetNumRealCells();
216
217     if (outputMode == 0) *p_log << entry_number << "\t" << seed <<
218         "\t" << count << "\n";
219     if (outputMode == 2) *p_log << "\n";
220
221     //Reset for next simulation
222     SimulationTime::Destroy();
223     delete cell_population;
224     entry_number++;
225
226     if (debugOutput)
227     {
228         debugWriter->Close();
229     }
230
231     p_RNG->Destroy();
232     LogFile::Close();
233
234     return exit_code;
235 }
236 ;

```

---

#### 17.1.4 /apps/src/HeSimulator.cpp

```

1 #include <iostream>
2 #include <string>
3
4 #include <cxxtest/TestSuite.h>
5 #include "ExecutableSupport.hpp"
6 #include "Exception.hpp"
7 #include "PetscTools.hpp"
8 #include "PetscException.hpp"
9
10 #include "HeCellCycleModel.hpp"
11 #include "OffLatticeSimulationPropertyStop.hpp"
12
13 #include "AbstractCellBasedTestSuite.hpp"
14
15 #include "WildTypeCellMutationState.hpp"

```

```
16 #include "TransitCellProliferativeType.hpp"
17 #include "DifferentiatedCellProliferativeType.hpp"
18
19 #include "CellsGenerator.hpp"
20 #include "HoneycombMeshGenerator.hpp"
21 #include "NodesOnlyMesh.hpp"
22 #include "NodeBasedCellPopulation.hpp"
23 #include "VertexBasedCellPopulation.hpp"
24
25 #include "ColumnDataWriter.hpp"
26
27 int main(int argc, char *argv[])
28 {
29     ExecutableSupport::StartupWithoutShowingCopyright(&argc, &argv);
30     //main() returns code indicating sim run success or failure mode
31     int exit_code = ExecutableSupport::EXIT_OK;
32
33     if (argc != 22 && argc != 20)
34     {
35         ExecutableSupport::PrintError(
```

```

36         "Wrong arguments for simulator.\nUsage (replace <> with
37         ← values, pass bools as 0 or 1):\nStochastic
38         ← Mode:\nHeSimulator <directoryString> <filenameString>
39         ← <outputModeUnsigned(0=counts,1=events,2=sequence)>
40         ← <deterministicBool=0>
41         ← <fixtureUnsigned(0=He;1=Wan;2=test)>
42         ← <founderAth5Mutant?Bool> <debugOutputBool>
43         ← <startSeedUnsigned> <endSeedUnsigned>
44         ← <inductionTimeDoubleHours>
45         ← <earliestLineageStartTime>
46         ← <latestLineageStartTime> <endTimeDoubleHours>
47         ← <mMitoticModePhase2Double> <mMitoticModePhase3Double>
48         ← <pPP1Double(0-1)> <pPD1Double(0-1)> <pPP1Double(0-1)>
49         ← <pPD1Double(0-1)> <pPP1Double(0-1)>
      ← <pPD1Double(0-1)>\nDeterministic Mode:\nHeSimulator
      ← <directoryString> <filenameString>
      ← <outputModeUnsigned(0=counts,1=events,2=sequence)>
      ← <deterministicBool=1>
      ← <fixtureUnsigned(0=He;1=Wan;2=test)>
      ← <founderAth5Mutant?Bool> <debugOutputBool>
      ← <startSeedUnsigned> <endSeedUnsigned>
      ← <inductionTimeDoubleHours>
      ← <earliestLineageStartTime>
      ← <latestLineageStartTime> <endTimeDoubleHours>
      ← <phase1ShapeDouble(>0)> <phase1ScaleDouble(>0)>
      ← <phase2ShapeDouble(>0)> <phase2ScaleDouble(>0)>
      ← <phaseBoundarySisterShiftWidthDouble>\n",
      ← true);
    exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
    return exit_code;
}

/*****
 * SIMULATOR PARAMETERS
 *****/
std::string directoryString, filenameString;
int outputMode; //0 = counts; 1 = mitotic mode events; 2 = mitotic
   ← mode sequence sampling
bool deterministicMode, ath5founder, debugOutput;
unsigned fixture, startSeed, endSeed; //fixture 0 = He2012; 1 =
   ← Wan2016
double inductionTime, earliestLineageStartTime,
   ← latestLineageStartTime, endTime;

```

```
50     double mitoticModePhase2, mitoticModePhase3, pPP1, pPD1, pPP2, pPD2,
51     ↵ pPP3, pPD3; //stochastic model parameters
52     double phase1Shape, phase1Scale, phase2Shape, phase2Scale,
53     ↵ phaseSisterShiftWidth, phaseOffset;
54
55 //PARSE ARGUMENTS
56 directoryString = argv[1];
57 filenameString = argv[2];
58 outputMode = std::stoi(argv[3]);
59 deterministicMode = std::stoul(argv[4]);
60 fixture = std::stoul(argv[5]);
61 ath5founder = std::stoul(argv[6]);
62 debugOutput = std::stoul(argv[7]);
63 startSeed = std::stoul(argv[8]);
64 endSeed = std::stoul(argv[9]);
65 inductionTime = std::stod(argv[10]);
66 earliestLineageStartTime = std::stod(argv[11]);
67 latestLineageStartTime = std::stod(argv[12]);
68 endTime = std::stod(argv[13]);
69
70 if (deterministicMode == 0)
71 {
72     mitoticModePhase2 = std::stod(argv[14]);
73     mitoticModePhase3 = std::stod(argv[15]);
74     pPP1 = std::stod(argv[16]);
75     pPD1 = std::stod(argv[17]);
76     pPP2 = std::stod(argv[18]);
77     pPD2 = std::stod(argv[19]);
78     pPP3 = std::stod(argv[20]);
79     pPD3 = std::stod(argv[21]);
80 }
81 else if (deterministicMode == 1)
82 {
83     phase1Shape = std::stod(argv[14]);
84     phase1Scale = std::stod(argv[15]);
85     phase2Shape = std::stod(argv[16]);
86     phase2Scale = std::stod(argv[17]);
87     phaseSisterShiftWidth = std::stod(argv[18]);
88     phaseOffset = std::stod(argv[19]);
89 }
```

```

90     ExecutableSupport::PrintError("Bad deterministicMode (argument
91         ↳ 4). Must be 0 or 1");
92     exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
93     return exit_code;
94 }
95 /**
96 * PARAMETER/ARGUMENT SANITY CHECK
97 */
98 bool sane = 1;
99
100 if (outputMode != 0 && outputMode != 1 && outputMode != 2)
101 {
102     ExecutableSupport::PrintError(
103         "Bad outputMode (argument 3). Must be 0 (counts) 1
104             ↳ (mitotic events) or 2 (sequence sampling)");
105     sane = 0;
106 }
107 if (fixture != 0 && fixture != 1 && fixture != 2)
108 {
109     ExecutableSupport::PrintError("Bad fixture (argument 5). Must be
110         ↳ 0 (He), 1 (Wan), or 2 (validation/test)");
111     sane = 0;
112 }
113 if (ath5founder != 0 && ath5founder != 1)
114 {
115     ExecutableSupport::PrintError("Bad ath5founder (argument 6). Must
116         ↳ be 0 (wild type) or 1 (ath5 mutant)");
117     sane = 0;
118 }
119 if (endSeed < startSeed)
120 {
121     ExecutableSupport::PrintError("Bad start & end seeds (arguments,
122         ↳ 8, 9). endSeed must not be < startSeed");
123     sane = 0;
124 }
125 if (inductionTime ≥ endTime)
126 {

```

```
127     ExecutableSupport::PrintError("Bad latestLineageStartTime  
128         ↪ (argument 10). Must be <endTime(arg13)");  
129     sane = 0;  
130 }  
131 if (earliestLineageStartTime ≥ endTime || earliestLineageStartTime  
132     ↪ ≥ latestLineageStartTime)  
133 {  
134     ExecutableSupport::PrintError(  
135         "Bad earliestLineageStartTime (argument 11). Must be  
136             ↪ <endTime(arg13), <latestLineageStarTime (arg12)");  
137     sane = 0;  
138 }  
139 if (latestLineageStartTime > endTime)  
140 {  
141     ExecutableSupport::PrintError("Bad latestLineageStartTime  
142         ↪ (argument 12). Must be ≤ endTime(arg13)");  
143     sane = 0;  
144 }  
145  
146 if (deterministicMode == 0)  
147 {  
148     if (mitoticModePhase2 < 0)  
149     {  
150         ExecutableSupport::PrintError("Bad mitoticModePhase2  
151             ↪ (argument 14). Must be >0");  
152         sane = 0;  
153     }  
154     if (mitoticModePhase3 < 0)  
155     {  
156         ExecutableSupport::PrintError("Bad mitoticModePhase3  
157             ↪ (argument 15). Must be >0");  
158         sane = 0;  
159     }  
160     ExecutableSupport::PrintError(  
161         "Bad phase 1 probabilities (arguments 16, 17). pPP1 +  
162             ↪ pPD1 should be ≥0, ≤1, sum should not exceed  
163             ↪ 1");  
164     sane = 0;  
165 }
```

```

160     if (pPP2 + pPD2 > 1 || pPP2 > 1 || pPP2 < 0 || pPD2 > 1 || pPD2 <
161         0)
162     {
163         ExecutableSupport::PrintError(
164             "Bad phase 2 probabilities (arguments 18, 19). pPP2 +
165             pPD2 should be  $\geq 0$ ,  $\leq 1$ , sum should not exceed
166             1");
167         sane = 0;
168     }
169     if (pPP3 + pPD3 > 1 || pPP3 > 1 || pPP3 < 0 || pPD3 > 1 || pPD3 <
170         0)
171     {
172         ExecutableSupport::PrintError(
173             "Bad phase 3 probabilities (arguments 20, 21). pPP3 +
174             pPD3 should be  $\geq 0$ ,  $\leq 1$ , sum should not exceed
175             1");
176         sane = 0;
177     }
178     if (deterministicMode == 1)
179     {
180         if (phase1Shape  $\leq 0$ )
181         {
182             ExecutableSupport::PrintError("Bad phase1Shape (argument 14).
183                 Must be  $> 0$ ");
184             sane = 0;
185         }
186         if (phase1Scale  $\leq 0$ )
187         {
188             ExecutableSupport::PrintError("Bad phase1Scale (argument 15).
189                 Must be  $> 0$ ");
190             sane = 0;
191         }
192         if (phase2Shape  $\leq 0$ )
193         {
194             ExecutableSupport::PrintError("Bad phase1Shape (argument 16).
195                 Must be  $> 0$ ");
196             sane = 0;
197         }
198         if (phase2Scale  $\leq 0$ )
199         {
200             ExecutableSupport::PrintError("Bad phase1Scale (argument 17).
201                 Must be  $> 0$ ");
202             sane = 0;
203         }
204     }
205 
```

```

193         ExecutableSupport::PrintError("Bad phase1Scale (argument 17).
194             ↪ Must be >0");
195         sane = 0;
196     }
197     if (phaseSisterShiftWidth ≤ 0)
198     {
199         ExecutableSupport::PrintError("Bad phaseSisterShiftWidth
200             ↪ (argument 18). Must be >0");
201         sane = 0;
202     }
203     if (sane == 0)
204     {
205         ExecutableSupport::PrintError("Exiting with bad arguments. See
206             ↪ errors for details");
207         exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
208         return exit_code;
209     }
210
211 /*****
212 * SIMULATOR OUTPUT SETUP
213 *****/
214
215 //Set up singleton LogFile
216 LogFile* p_log = LogFile::Instance();
217 p_log→Set(0, directoryString, filenameString);
218
219 ExecutableSupport::Print("Simulator writing file " + filenameString +
220     ↪ " to directory " + directoryString);
221
222 //Log entry counter
223     unsigned entry_number = 1;
224
225 //Write appropriate headers to log
226     if (outputMode == 0) *p_log << "Entry\tInduction Time
227         ↪ (h)\tSeed\tCount\n";
228     if (outputMode == 1) *p_log << "Time (hpf)\tSeed\tCellID\tMitotic
229         ↪ Mode (0=PP;1=PD;2=DD)\n";
230     if (outputMode == 2) *p_log << "Entry\tSeed\tSequence\n";
231
232 //Instance RNG

```

```
230     RandomNumberGenerator* p_RNG = RandomNumberGenerator::Instance();  
231  
232 //Initialise pointers to relevant singleton ProliferativeTypes and  
233     ↵ Properties  
234     MAKE_PTR(WildTypeCellMutationState, p_state);  
235     MAKE_PTR(TransitCellProliferativeType, p_Mitotic);  
236     MAKE_PTR(DifferentiatedCellProliferativeType, p_PostMitotic);  
237     MAKE_PTR(Ath5Mo, p_Morpholino);  
238     MAKE_PTR(CellLabel, p_label);  
239  
240 //*****  
241 * SIMULATOR SETUP & RUN  
242 *****/  
243  
244 //iterate through supplied seed range, executing one simulation per seed  
245 for (unsigned seed = startSeed; seed ≤ endSeed; seed++)  
246 {  
247     if (outputMode == 2) *p_log << entry_number << "\t" << seed <<  
248         ↵ "\t"; //write seed to log - sequence written by  
249         ↵ cellcyclemodel objects  
250  
251     //initialise pointer to debugWriter  
252     ColumnDataWriter* debugWriter;  
253  
254     //initialise SimulationTime (permits cellcyclemodel setup)  
255     SimulationTime::Instance()→SetStartTime(0.0);  
256  
257     //Reseed the RNG with the required seed  
258     p_RNG→Reseed(seed);  
259  
260     //Initialise a HeCellCycleModel and set it up with appropriate  
261     ↵ TiL values  
262     HeCellCycleModel* p_cycle_model = new HeCellCycleModel;  
263  
264     if (debugOutput)  
265     {  
266         //Pass ColumnDataWriter to cell cycle model for debug output  
267         boost::shared_ptr<ColumnDataWriter> p_debugWriter(  
268             new ColumnDataWriter(directoryString, filenameString  
269                 ↵ + "DEBUG_" + std::to_string(seed), false, 10));  
270         p_cycle_model→EnableModelDebugOutput(p_debugWriter);  
271         debugWriter = &p_debugWriter;  
272     }
```



```

298     }
299     else if (fixture == 1) //Wan 2016-type fixture - each lineage
300         ↵ founder selected randomly across residency time, simulator
301         ↵ allowed to run until end of residency time
302         //passing residency time (as latestLineageStartTime) and endTime
303         ↵ separately allows for creation of "shadow CMZ" population
304         //this allows investigation of different assumptions about how
305         ↵ Wan et al.'s model output was generated
306     {
307         //generate random lineage start time from even random distro
308         ↵ across CMZ residency time
309         currTiL = p_RNG->ranf() * latestLineageStartTime;
310         currSimEndTime = std::max(.05, endTime - currTiL); //minimum
311         ↵ 1 timestep, prevents 0 timestep SimulationTime error
312         if (outputMode == 1) p_cycle_model->EnableModeEventOutput(0,
313             ↵ seed);
314     }
315     else if (fixture == 2) //validation fixture- all founders have
316         ↵ TiL given by induction time
317     {
318         currTiL = inductionTime;
319         currSimEndTime = endTime;
320     }
321
322     //Setup lineages' cycle model with appropriate parameters
323     p_cycle_model->SetDimension(2);
324     //p_cycle_model->SetPostMitoticType(p_PostMitotic);
325
326     if (!deterministicMode)
327     {
328         p_cycle_model->SetModelParameters(currTiL, mitoticModePhase2,
329             ↵ mitoticModePhase2 + mitoticModePhase3, pPP1,
330                 pPD1, pPP2, pPD2, pPP3,
331                     ↵ pPD3);
332     }
333     else
334     {
335         //Gamma-distribute phase3 boundary
336         double currPhase2Boundary = phaseOffset +
337             ↵ p_RNG->GammaRandomDeviate(phase1Shape, phase1Scale);
338         double currPhase3Boundary = currPhase2Boundary +
339             ↵ p_RNG->GammaRandomDeviate(phase2Shape, phase2Scale);
340     }
341 
```

```

329     p_cycle_model->SetDeterministicMode(currTil,
330         ↳ currPhase2Boundary, currPhase3Boundary,
331         ↳ phaseSisterShiftWidth);
332 }
333
334     //Setup vector containing lineage founder with the properly set
335     ↳ up cell cycle model
336     std::vector<CellPtr> cells;
337     CellPtr p_cell(new Cell(p_state, p_cycle_model));
338     p_cell->SetCellProliferativeType(p_Mitotic);
339     if (ath5founder == 1) p_cell->AddCellProperty(p_Morpholino);
340     if (outputMode == 2) p_cell->AddCellProperty(p_label);
341     p_cell->InitialiseCellCycleModel();
342     cells.push_back(p_cell);

343     //Generate 1x1 mesh for single-cell colony
344     HoneycombMeshGenerator generator(1, 1);
345     MutableMesh<2, 2>* p_generating_mesh = generator.GetMesh();
346     NodesOnlyMesh<2> mesh;
347     mesh.ConstructNodesWithoutMesh(*p_generating_mesh, 1.5);

348     //Setup cell population
349     NodeBasedCellPopulation<2>* cell_population(new
350         ↳ NodeBasedCellPopulation<2>(mesh, cells));

351     //Setup simulator & run simulation
352     boost::shared_ptr<OffLatticeSimulationPropertyStop<2>>
353         ↳ p_simulator(
354             new
355                 ↳ OffLatticeSimulationPropertyStop<2>(*cell_population));
356     p_simulator->SetStopProperty(p_Mitotic); //simulation to stop if
357     ↳ no mitotic cells are left
358     p_simulator->SetDt(0.05);
359     p_simulator->SetEndTime(currSimEndTime);
360     p_simulator->SetOutputDirectory("UnusedSimOutput" +
361         ↳ filenameString); //unused output
362     p_simulator->Solve();

363     //Count lineage size
364     unsigned count = cell_population->GetNumRealCells();
365

```

```

364     if (outputMode == 0) *p_log << entry_number << "\t" <<
365         inductionTime << "\t" << seed << "\t" << count << "\n";
366     if (outputMode == 2) *p_log << "\n";
367
368     //Reset for next simulation
369     SimulationTime::Destroy();
370     delete cell_population;
371     entry_number++;
372
373     if (debugOutput)
374     {
375         debugWriter->Close();
376     }
377
378
379     p_RNG->Destroy();
380     LogFile::Close();
381
382     return exit_code;
383 }
384 ;

```

---

### 17.1.5 /apps/src/WanSimDebug.cpp

```

1 #include <iostream>
2 #include <string>
3
4 #include <cxxtest/TestSuite.h>
5 #include "ExecutableSupport.hpp"
6 #include "Exception.hpp"
7 #include "PetscTools.hpp"
8 #include "PetscException.hpp"
9
10 #include "WanStemCellCycleModel.hpp"
11 #include "HeCellCycleModel.hpp"
12 #include "OffLatticeSimulation.hpp"
13
14 #include "AbstractCellBasedTestSuite.hpp"
15
16 #include "WildTypeCellMutationState.hpp"
17 #include "StemCellProliferativeType.hpp"

```

```
18 #include "TransitCellProliferativeType.hpp"
19 #include "DifferentiatedCellProliferativeType.hpp"
20
21 #include "CellsGenerator.hpp"
22 #include "HoneycombMeshGenerator.hpp"
23 #include "NodesOnlyMesh.hpp"
24 #include "NodeBasedCellPopulation.hpp"
25 #include "VertexBasedCellPopulation.hpp"
26
27 #include "CellProliferativeTypesCountWriter.hpp"
28
29
30 int main(int argc, char *argv[])
31 {
32     ExecutableSupport::StartupWithoutShowingCopyright(&argc, &argv);
33     //main() returns code indicating sim run success or failure mode
34     int exit_code = ExecutableSupport::EXIT_OK;
35
36     /*****
37     * SIMULATOR PARAMETERS
38     *****/
39     std::string directoryString, filenameString;
40
41     //PARSE ARGUMENTS
42     directoryString = argv[1];
43     filenameString = argv[2];
44
45     /*****
46     * SIMULATOR OUTPUT SETUP
47     *****/
48
49 //Set up singleton LogFile
50    LogFile* p_log = LogFile::Instance();
51     p_log->Set(0, directoryString, filenameString);
52
53     ExecutableSupport::Print("Simulator writing file " + filenameString +
54     → " to directory " + directoryString);
55
56 //Log entry counter
57     //unsigned entry_number = 1;
58
59 //Write appropriate header to log
```

```

59     *p_log <<
60     "Entry\tTotalCells\tStemCount\tProgenitorCount\tPostMitoticCount\n";
61
62 //Instance RNG
63     RandomNumberGenerator* p_rng = RandomNumberGenerator::Instance();
64
65 //Initialise pointers to relevant singleton ProliferativeTypes and
66 //Properties
67     boost::shared_ptr<AbstractCellProperty>
68         p_state(CellPropertyRegistry::Instance()→Get<WildTypeCellMutationState>());
69     boost::shared_ptr<AbstractCellProperty>
70         p_Stem(CellPropertyRegistry::Instance()→Get<StemCellProliferativeType>());
71     boost::shared_ptr<AbstractCellProperty>
72         p_Transit(CellPropertyRegistry::Instance()→Get<TransitCellProliferativeType>());
73     boost::shared_ptr<AbstractCellProperty>
74         p_PostMitotic(CellPropertyRegistry::Instance()→Get<DifferentiatedCellProlife
75
76 //*****
77 // SIMULATOR SETUP & RUN
78 //*****
79
80 //initialise SimulationTime (permits cellcyclemodel setup)
81 SimulationTime::Instance()→SetStartTime(0.0);
82
83 std::vector<CellPtr> cells;
84
85 WanStemCellCycleModel* p_stem_model = new WanStemCellCycleModel;
86
87 boost::shared_ptr<ColumnDataWriter> p_debugWriter(
88     new ColumnDataWriter(directoryString, filenameString +
89     "DEBUG_WAN", false, 10));
90
91 p_stem_model→SetDimension(2);
92 p_stem_model→EnableModelDebugOutput(p_debugWriter);
93 CellPtr p_cell(new Cell(p_state, p_stem_model));
94 p_cell→InitialiseCellCycleModel();
95 cells.push_back(p_cell);
96
97 //Generate 1x#cells mesh for abstract colony
98 HoneycombMeshGenerator generator(1, 1);
99 MutableMesh<2, 2>* p_generating_mesh = generator.GetMesh();
100 NodesOnlyMesh<2> mesh;

```

```
95     mesh.ConstructNodesWithoutMesh(*p_generating_mesh, 1.5);
96
97     //Setup cell population
98     boost::shared_ptr<NodeBasedCellPopulation<2>> cell_population(new
99         → NodeBasedCellPopulation<2>(mesh, cells));
100
101    → cell_population→AddCellPopulationCountWriter<CellProliferativeTypesCountWriter>
102
103    //Setup simulator & run simulation
104    boost::shared_ptr<OffLatticeSimulation<2>> p_simulator(
105        → new OffLatticeSimulation<2>(*cell_population));
106    p_simulator→SetDt(1);
107    p_simulator→SetOutputDirectory(directoryString + "/WanDebug");
108    p_simulator→SetEndTime(100);
109    p_simulator→Solve();
110
111    std::vector<unsigned> prolTypes =
112        → cell_population→GetCellProliferativeTypeCount();
113    unsigned realCells = cell_population→GetNumRealCells();
114    unsigned allCells = cell_population→GetNumAllCells();
115
116    Timer::Print("stem: " + std::to_string(prolTypes[0]) + " transit: " +
117        → std::to_string(prolTypes[1]) + " postmitotic: " +
118        → std::to_string(prolTypes[2]));
119    Timer::Print("realcells: " + std::to_string(realCells) + " allcells:
120        → " + std::to_string(allCells));
121
122    //Reset for next simulation
123    SimulationTime::Destroy();
124    cell_population.reset();
125
126    p_RNG→Destroy();
127    LogFile::Close();
128
129    return exit_code;
130 }
131 ;
```

### 17.1.6 /apps/src/WanSimulator.cpp

---

```
1 #include <iostream>
2 #include <string>
3
4 #include <cxxtest/TestSuite.h>
5 #include "ExecutableSupport.hpp"
6 #include "Exception.hpp"
7 #include "PetscTools.hpp"
8 #include "PetscException.hpp"
9
10 #include "WanStemCellCycleModel.hpp"
11 #include "HeCellCycleModel.hpp"
12 #include "OffLatticeSimulationPropertyStop.hpp"
13
14 #include "AbstractCellBasedTestSuite.hpp"
15
16 #include "WildTypeCellMutationState.hpp"
17 #include "StemCellProliferativeType.hpp"
18 #include "TransitCellProliferativeType.hpp"
19 #include "DifferentiatedCellProliferativeType.hpp"
20
21 #include "CellsGenerator.hpp"
22 #include "HoneycombMeshGenerator.hpp"
23 #include "NodesOnlyMesh.hpp"
24 #include "NodeBasedCellPopulation.hpp"
25 #include "VertexBasedCellPopulation.hpp"
26
27 #include "CellProliferativeTypesCountWriter.hpp"
28
29 int main(int argc, char *argv[])
30 {
31     ExecutableSupport::StartupWithoutShowingCopyright(&argc, &argv);
32     //main() returns code indicating sim run success or failure mode
33     int exit_code = ExecutableSupport::EXIT_OK;
34
35     if (argc != 23)
36     {
37         ExecutableSupport::PrintError(
```

```

38         "Wrong arguments for simulator.\nUsage (replace <> with
39         ← values, pass bools as 0 or 1):\n WanSimulator
40         ← <directoryString> <startSeedUnsigned>
41         ← <endSeedUnsigned> <cmzResidencyTimeDoubleHours>
42         ← <stemDivisorDouble>
43         ← <meanProgenitorPopualtion@3dpfDouble>
44         ← <stdProgenitorPopulation@3dpfDouble>
45         ← <stemGammaShiftDouble> <stemGammaShapeDouble>
46         ← <stemGammaScaleDouble> <progenitorGammaShiftDouble>
47         ← <progenitorGammaShapeDouble>
48         ← <progenitorGammaScaleDouble>
49         ← <progenitorSisterShiftDouble>
50         ← <mMitoticModePhase2Double> <mMitoticModePhase3Double>
51         ← <pPP1Double(0-1)> <pPD1Double(0-1)> <pPP1Double(0-1)>
52         ← <pPD1Double(0-1)> <pPP1Double(0-1)>
53         ← <pPD1Double(0-1)>",
54         true);
55     exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
56     return exit_code;
57 }
58
59 //*****
60 * SIMULATOR PARAMETERS
61 *****/
62 std::string directoryString;
63 unsigned startSeed, endSeed;
64 double cmzResidencyTime;
65 double stemDivisor, progenitorMean, progenitorStd;
66 double stemGammaShift, stemGammaShape, stemGammaScale,
67     → progenitorGammaShift, progenitorGammaShape,
68     → progenitorGammaScale, progenitorGammaSister;
69 double mitoticModePhase2, mitoticModePhase3, pPP1, pPD1, pPP2, pPD2,
70     → pPP3, pPD3; //stochastic He model parameters
71
72 //PARSE ARGUMENTS
73 directoryString = argv[1];
74 startSeed = std::stoul(argv[2]);
75 endSeed = std::stoul(argv[3]);
76 cmzResidencyTime = std::stod(argv[4]);
77 //starting cell number distributions
78 stemDivisor = std::stod(argv[5]);
79 progenitorMean = std::stod(argv[6]);
80 progenitorStd = std::stod(argv[7]);

```

```

64    //cycle duration params
65    stemGammaShift = std::stod(argv[8]);
66    stemGammaShape = std::stod(argv[9]);
67    stemGammaScale = std::stod(argv[10]);
68    progenitorGammaShift = std::stod(argv[11]);
69    progenitorGammaShape = std::stod(argv[12]);
70    progenitorGammaScale = std::stod(argv[13]);
71    progenitorGammaSister = std::stod(argv[14]);
72    //He model params
73    mitoticModePhase2 = std::stod(argv[15]);
74    mitoticModePhase3 = std::stod(argv[16]);
75    pPP1 = std::stod(argv[17]);
76    pPD1 = std::stod(argv[18]);
77    pPP2 = std::stod(argv[19]);
78    pPD2 = std::stod(argv[20]);
79    pPP3 = std::stod(argv[21]);
80    pPD3 = std::stod(argv[22]);
81
82    std::vector<double> stemOffspringParams = { mitoticModePhase2,
83        ↪ mitoticModePhase2 + mitoticModePhase3, pPP1, pPD1,
84                                ↪ pPP2, pPD2, pPP3, pPD3,
85                                ↪ progenitorGammaShift,
86                                ↪ progenitorGammaShape,
87                                ↪ progenitorGammaScale,
88                                ↪ progenitorGammaSister
89                                ↪ };
90
91    /*****
92     * PARAMETER/ARGUMENT SANITY CHECK
93     *****/
94
95    bool sane = 1;
96
97    if (endSeed < startSeed)
98    {
99        ExecutableSupport::PrintError("Bad start & end seeds (arguments,
100            ↪ 3, 4). endSeed must not be < startSeed");
101        sane = 0;
102    }
103
104    if (cmzResidencyTime < 0)
105    {
106        ExecutableSupport::PrintError("Bad CMZ residency time (argument
107            ↪ 5). cmzResidencyTime must be positive-valued");
108    }

```

```
100     sane = 0;
101 }
102
103 if (stemDivisor <= 0)
104 {
105     ExecutableSupport::PrintError("Bad stemDivisor (argument 6).
106     ↳ stemDivisor must be positive-valued");
107     sane = 0;
108 }
109
110 if (progenitorMean <= 0 || progenitorStd <= 0)
111 {
112     ExecutableSupport::PrintError("Bad progenitorMean or
113     ↳ progenitorStd (arguments 7,8). Must be positive-valued");
114     sane = 0;
115 }
116
117 if (stemGammaShift < 0 || stemGammaShape < 0 || stemGammaScale < 0)
118 {
119     ExecutableSupport::PrintError(
120         "Bad stemGammaShift, stemGammaShape, or stemGammaScale
121         ↳ (arguments 9, 10, 11). Shifts must be ≥0, cycle
122         ↳ shape and scale params must be positive-valued");
123     sane = 0;
124 }
125
126 if (progenitorGammaShift < 0 || progenitorGammaShape < 0 ||
127     ↳ progenitorGammaScale < 0 || progenitorGammaSister < 0)
128 {
129     ExecutableSupport::PrintError(
130         "Bad progenitorGammaShift, progenitorGammaShape,
131         ↳ progenitorGammaScale, or progenitorGammaSisterShift
132         ↳ (arguments 12,13,14,15). Shifts must be ≥0, cycle
133         ↳ shape and scale params must be positive-valued");
134     sane = 0;
135 }
136
137 if (mitoticModePhase2 < 0)
138 {
139     ExecutableSupport::PrintError("Bad mitoticModePhase2 (argument
140     ↳ 14). Must be >0");
141     sane = 0;
142 }
143 }
```

```

134     if (mitoticModePhase3 < 0)
135     {
136         ExecutableSupport::PrintError("Bad mitoticModePhase3 (argument
137             ↳ 15). Must be >0");
138         sane = 0;
139     }
140     if (pPP1 + pPD1 > 1 || pPP1 > 1 || pPP1 < 0 || pPD1 > 1 || pPD1 < 0)
141     {
142         ExecutableSupport::PrintError(
143             "Bad phase 1 probabilities (arguments 16, 17). pPP1 +
144                 ↳ pPD1 should be ≥0, ≤1, sum should not exceed 1");
145         sane = 0;
146     }
147     if (pPP2 + pPD2 > 1 || pPP2 > 1 || pPP2 < 0 || pPD2 > 1 || pPD2 < 0)
148     {
149         ExecutableSupport::PrintError(
150             "Bad phase 2 probabilities (arguments 18, 19). pPP2 +
151                 ↳ pPD2 should be ≥0, ≤1, sum should not exceed 1");
152         sane = 0;
153     }
154     if (pPP3 + pPD3 > 1 || pPP3 > 1 || pPP3 < 0 || pPD3 > 1 || pPD3 < 0)
155     {
156         ExecutableSupport::PrintError(
157             "Bad phase 3 probabilities (arguments 20, 21). pPP3 +
158                 ↳ pPD3 should be ≥0, ≤1, sum should not exceed 1");
159         sane = 0;
160     }
161     if (sane == 0)
162     {
163         ExecutableSupport::PrintError("Exiting with bad arguments. See
164             ↳ errors for details");
165         exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
166         return exit_code;
167     }
168
169     ****
170     * SIMULATOR SETUP & RUN
171     ****
172
173     ExecutableSupport::Print("Simulator writing files to directory " +
174             ↳ directoryString);
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2488
2489
2490
2491
2492
24
```

```

171 //Instance RNG
172     RandomNumberGenerator* p_RNG = RandomNumberGenerator::Instance();
173
174 //Initialise pointers to relevant singleton ProliferativeTypes and
175     ↳ Properties
176     boost::shared_ptr<AbstractCellProperty>
177         ↳ p_state(CellPropertyRegistry::Instance()→Get<WildTypeCellMutationState>());
178     boost::shared_ptr<AbstractCellProperty>
179         ↳ p_Stem(CellPropertyRegistry::Instance()→Get<StemCellProliferativeType>());
180     boost::shared_ptr<AbstractCellProperty> p_Transit(
181
182             ↳ CellPropertyRegistry::Instance()→Get<TransitCellProliferativeType>()
183     boost::shared_ptr<AbstractCellProperty> p_PostMitotic(
184
185             ↳ CellPropertyRegistry::Instance()→Get<DifferentiatedCellProliferativeType>()
186
187 //iterate through supplied seed range, executing one simulation per seed
188     for (unsigned seed = startSeed; seed ≤ endSeed; seed++)
189     {
190         //initialise SimulationTime (permits cellcyclemodel setup)
191         SimulationTime::Instance()→SetStartTime(0.0);
192
193         //Reseed the RNG with the required seed
194         p_RNG→Reseed(seed);
195
196         //unsigned numberStem =
197             ↳ int(std::round(p_RNG→NormalRandomDeviate(stemMean,
198                 ↳ stemStd)));
199         unsigned numberProgenitors =
200             ↳ int(std::round(p_RNG→NormalRandomDeviate(progenitorMean,
201                 ↳ progenitorStd)));
202         unsigned numberStem = int(std::round(numberProgenitors /
203             ↳ stemDivisor));
204
205         std::vector<CellPtr> stems;
206         std::vector<CellPtr> cells;
207
208         for (unsigned i = 0; i < numberStem; i++)
209         {
210             WanStemCellCycleModel* p_stem_model = new
211                 ↳ WanStemCellCycleModel;
212             p_stem_model→SetDimension(2);

```

```

202     p_stem_model→SetModelParameters(stemGammaShift,
203     ↳ stemGammaShape, stemGammaScale, stemOffspringParams);
204
205     CellPtr p_cell(new Cell(p_state, p_stem_model));
206     p_cell→InitialiseCellCycleModel();
207     stems.push_back(p_cell);
208     cells.push_back(p_cell);
209 }
210
211 for (unsigned i = 0; i < numberProgenitors; i++)
212 {
213     double currTiL = p_RNG→ranf() * cmzResidencyTime;
214
215     HeCellCycleModel* p_prog_model = new HeCellCycleModel;
216     p_prog_model→SetDimension(2);
217     p_prog_model→SetModelParameters(currTiL, mitoticModePhase2,
218     ↳ mitoticModePhase2 + mitoticModePhase3, pPP1,
219     ↳ pPD1, pPP2, pPD2, pPP3,
220     ↳ pPD3);
221     p_prog_model→EnableKillSpecified();
222
223 }
224
225 //Generate 1x#cells mesh for abstract colony
226 HoneycombMeshGenerator generator(1, (numberProgenitors +
227   ↳ numberStem));
228 MutableMesh<2, 2>* p_generating_mesh = generator.GetMesh();
229 NodesOnlyMesh<2> mesh;
230 mesh.ConstructNodesWithoutMesh(*p_generating_mesh, 1.5);
231
232 //Setup cell population
233 boost::shared_ptr<NodeBasedCellPopulation<2>> cell_population(new
234   ↳ NodeBasedCellPopulation<2>(mesh, cells));
235
236   ↳ cell_population→AddCellPopulationCountWriter<CellProliferativeTypesCount>();
237
238 //Give Wan stem cells the population & base stem pop size
239 for (auto p_cell : stems)
240 {

```

```

238     WanStemCellCycleModel* p_cycle_model =
239         → dynamic_cast<WanStemCellCycleModel*>(p_cell→GetCellCycleModel());
240     p_cycle_model→EnableExpandingStemPopulation(numberStem,
241         → cell_population);
242 }
243
244 //Setup simulator & run simulation
245 boost::shared_ptr<OffLatticeSimulationPropertyStop<2>>
246     → p_simulator(
247         new
248             → OffLatticeSimulationPropertyStop<2>(*cell_population));
249 p_simulator→SetStopProperty(p_Transit); //simulation to stop if
250     → no RPCs are left
251 p_simulator→SetDt(1);
252 p_simulator→SetOutputDirectory(directoryString + "/Seed" +
253     → std::to_string(seed) + "Results");
254 p_simulator→SetEndTime(8568); // 360dpf - 3dpf simulation start
255     → time
256 p_simulator→Solve();
257
258 //Reset for next simulation
259 SimulationTime::Destroy();
260 cell_population.reset();
261 }
262
263 p_RNG→Destroy();
264
265
266     return exit_code;
267 }
268 ;

```

---

### 17.1.7 /python\_fixtures/He\_output\_fixture.py

---

```

1 import multiprocessing
2 import os
3 import subprocess
4 import datetime
5
6 import numpy as np
7 from imageio.plugins._bsdf import BsdfSerializer
8
9 executable = '/home/main/chaste_build/projects/ISP/apps/HeSimulator'

```

```

10
11 if not(os.path.isfile(executable)):
12     raise Exception('Could not find executable: ' + executable)
13
14 ######
15 # SIMULATION PARAMETERS
16 #####
17
18 #Define start and end RNG seeds; determines:
19 #No. lineages per loss function run
20 #unique sequence of RNG results for each lineage
21 start_seed = 0
22 end_seed_counts = 9999
23 end_seed_events = 999
24 run_modes = [0,1,2] #1=deterministic mode, 0=refit stochastic mode,
    ↪ 2=original fit
25 debug_output = 0 #0=off;1=on
26
27 count_directory = "HeCounts"
28 count_filenames = ["induction", "wan", "ath5", "validate"]
29 event_directory = "HeModeEvent"
30 event_filenames = ["inductionMode", "validateMode"]
31 induction_times = [24, 32, 48]
32
33 #####
34 #GLOBAL MODEL PARAMETERS
35 #####
36
37 #Values defining different marker induction timepoints & relative start
    ↪ time of TiL counter
38 earliest_lineage_start_time = 23.0 #RPCs begin to enter "He model regime"
    ↪ nasally at 23hpf
39 latest_lineage_start_time = 39.0 #last temporal retinal RPC has entered
    ↪ "He model regime" at 39hpf
40 counts_end_time = 72.0
41 events_end_time = 80.0
42 wan_residency_time = 17.0
43
44 #####
45 # SPECIFIC MODEL PARAMETERS - THETAHAT
46 #####
47
48 #These parameters are the results of the SPSA optimisation fixture

```

```

49
50 #STOCHASTIC MITOTIC MODE
51 #HE ORIGINAL PARAMETERS
52 #mitotic mode per-phase probabilities
53 #3-phase mitotic mode time periodisation
54 o_mitotic_mode_phase_2 = 8 #These are phase lengths, so
55 o_mitotic_mode_phase_3 = 7 #Phase3 boundary = mmp2 + mmp3
56 o_phase_1_pPP = 1.0
57 o_phase_1_pPD = 0.0
58 o_phase_2_pPP = 0.2
59 o_phase_2_pPD = 0.4
60 o_phase_3_pPP = 0.2
61 o_phase_3_pPD = 0.0
62
63 #SPSA REFIT PARAMETERS
64 #mitotic mode per-phase probabilities
65 #3-phase mitotic mode time periodisation
66 mitotic_mode_phase_2 = 4.1483 #These are phase lengths, so
67 mitotic_mode_phase_3 = 11.6416 #Phase3 boundary = mmp2 + mmp3
68 phase_1_pPP = 1.0
69 phase_1_pPD = 0.0
70 phase_2_pPP = 0.1959
71 phase_2_pPD = 0.5168
72 phase_3_pPP = 0.2934
73 phase_3_pPD = 0.0
74 he_model_params = 15
75
76 #DETERMINISTIC MITOTIC MODE
77 #Phase boundary shift parameters
78 phase_1_shape = 3.7371
79 phase_1_scale = 1.8114
80 phase_2_shape = 2.5769
81 phase_2_scale = 1.6814
82 phase_sister_shift_widths = 1.6326
83 phase_offset = 1.2333
84 det_model_params = 12
85
86 original_theta_string = str(o_mitotic_mode_phase_2) + " " +
    ↵ str(o_mitotic_mode_phase_3) + " " + str(o_phase_1_pPP) + " " +
    ↵ str(o_phase_1_pPD) + " " + str(o_phase_2_pPP) + " " +
    ↵ str(o_phase_2_pPD) + " " + str(o_phase_3_pPP) + " " +
    ↵ str(o_phase_3_pPD)

```

```

87 stochastic_theta_string = str(mitotic_mode_phase_2) + " " +
→ str(mitotic_mode_phase_3) + " " +str(phase_1_pPP) + " " +
→ str(phase_1_pPD) + " " + str(phase_2_pPP) + " " + str(phase_2_pPD) +
→ " " + str(phase_3_pPP) + " " + str(phase_3_pPD)
88 deterministic_theta_string = str(phase_1_shape) + " " +
→ str(phase_1_scale) + " " + str(phase_2_shape) + " " +
→ str(phase_2_scale) + " " + str(phase_sister_shift_widths) + " " +
→ str(phase_offset)

89
90 def main():
91
92     command_list = []
93
94     for m in range(0,len(run_modes)):
95         curr_list = []
96
97         run_mode_string = str(run_modes[m])
98         deterministic_mode = run_modes[m]
99         if run_modes[m] == 2:
100             deterministic_mode = 0
101         command_count = 0 #for iterating seed numbers
102         base_command = executable
103
104         #induction count commands
105         for i in range(0,len(induction_times)):
106
107             output_mode = 0 #0=lineage counts;1=mitotic event
→ logging;2=sequence sampling
108             fixture = 0 #0=He 2012;1=Wan 2016
109             curr_start_seed = start_seed + command_count *
→ (end_seed_counts+1)
110             curr_end_seed = curr_start_seed + end_seed_counts
111             ath5founder = 0
112
113             command = base_command+" "\n
114                 +count_directory+" "\n
115
→ +count_filenames[0]+str(induction_times[i])+SDMode+run_
→ "\n
116             +str(output_mode)+" "\n
117             +str(deterministic_mode)+" "\n
118             +str(fixture)+" "\n
119             +str(ath5founder)+" "

```

```

120                     +str(debug_output)+" "\n
121                     +str(curr_start_seed)+" "\n
122                     +str(curr_end_seed)+" "\n
123                     +str(induction_times[i])+" "\n
124                     +str(earliest_lineage_start_time)+" "\n
125                     +str(latest_lineage_start_time)+" "\n
126                     +str(counts_end_time)+" "
127             curr_list.append(command)
128             command_count += 1
129
130     #wan command
131     output_mode = 0
132     fixture = 1
133     curr_start_seed = start_seed + command_count *
134         ↪ (end_seed_counts+1)
135     curr_end_seed = curr_start_seed + end_seed_counts
136     ath5founder = 0
137     wan_command = base_command+" \
138                     +count_directory+" "\n
139                     +count_filenames[1]+"SDMode"+run_mode_string+" "\n
140                     +str(output_mode)+" "\n
141                     +str(deterministic_mode)+" "\n
142                     +str(fixture)+" "\n
143                     +str(ath5founder)+" "\n
144                     +str(debug_output)+" "\n
145                     +str(curr_start_seed)+" "\n
146                     +str(curr_end_seed)+" "\n
147                     +str(0)+" "\n
148                     +str(0)+" "\n
149                     +str(wan_residency_time)+" "\n
150                     +str(counts_end_time)+" "
151             curr_list.append(wan_command)
152             command_count += 1
153
154     #wan- 17 hr constraint- no "shadow RPCs"
155     curr_start_seed = start_seed + command_count *
156         ↪ (end_seed_counts+1)
157     curr_end_seed = curr_start_seed + end_seed_counts
158     wan_command = base_command+" \
159                     +count_directory+" "\n
160                     ↪ +count_filenames[1]+"NoShadSDMode"+run_mode_string+" "
161                     ↪ "\n"

```

```
159             +str(output_mode)+" "\n"
160             +str(deterministic_mode)+" "\n"
161             +str(fixture)+" "\n"
162             +str(ath5founder)+" "\n"
163             +str(debug_output)+" "\n"
164             +str(curr_start_seed)+" "\n"
165             +str(curr_end_seed)+" "\n"
166             +str(0)+" "\n"
167             +str(0)+" "\n"
168             +str(wan_residency_time)+" "\n"
169             +str(wan_residency_time)+" "
170     curr_list.append(wan_command)
171     command_count += 1
172
173 #ath5 command
174 output_mode = 0
175 fixture = 2
176 curr_start_seed = start_seed + command_count *
177     ↪ (end_seed_counts+1)
177 curr_end_seed = curr_start_seed + end_seed_counts
178 ath5founder = 1
179
180 ath5_command = base_command+" "\n"
181             +count_directory+" "\n"
182             +count_filenames[2]+"SDMode"+run_mode_string+" "\n"
183             +str(output_mode)+" "\n"
184             +str(deterministic_mode)+" "\n"
185             +str(fixture)+" "\n"
186             +str(ath5founder)+" "\n"
187             +str(debug_output)+" "\n"
188             +str(curr_start_seed)+" "\n"
189             +str(curr_end_seed)+" "\n"
190             +str(0)+" "\n"
191             +str(earliest_lineage_start_time)+" "\n"
192             +str(latest_lineage_start_time)+" "\n"
193             +str(counts_end_time)+" "
194     curr_list.append(ath5_command)
195     command_count += 1
196
197 #validate counts command
198 curr_start_seed = start_seed + command_count *
199     ↪ (end_seed_counts+1)
200 curr_end_seed = curr_start_seed + end_seed_counts
```

```
200     ath5founder = 0
201
202     validate_command = base_command + " " \
203                         + count_directory + " " \
204                         + count_filenames[3] + "SDMode" + run_mode_string + " " \
205                         + str(output_mode) + " " \
206                         + str(deterministic_mode) + " " \
207                         + str(fixture) + " " \
208                         + str(ath5founder) + " " \
209                         + str(debug_output) + " " \
210                         + str(curr_start_seed) + " " \
211                         + str(curr_end_seed) + " " \
212                         + str(0) + " " \
213                         + str(earliest_lineage_start_time) + " " \
214                         + str(latest_lineage_start_time) + " " \
215                         + str(counts_end_time) + " "
216     curr_list.append(validate_command)
217     command_count += 1
218
219     #mitotic mode rate command
220     output_mode = 1
221     fixture = 0
222     curr_start_seed = start_seed + command_count *
223                         → (end_seed_counts+1)
224     curr_end_seed = curr_start_seed + end_seed_events
225     mode_rate_command = base_command + " " \
226                         + event_directory + " " \
227                         + event_filenames[0] + "SDMode" + run_mode_string + " " \
228                         + str(output_mode) + " " \
229                         + str(deterministic_mode) + " " \
230                         + str(fixture) + " " \
231                         + str(ath5founder) + " " \
232                         + str(debug_output) + " " \
233                         + str(curr_start_seed) + " " \
234                         + str(curr_end_seed) + " " \
235                         + str(earliest_lineage_start_time) + " " \
236                         + str(earliest_lineage_start_time) + " " \
237                         + str(latest_lineage_start_time) + " " \
238                         + str(events_end_time) + " "
239     curr_list.append(mode_rate_command)
240     command_count += 1
241
```

```

242     #validate rate command
243     fixture = 2
244     curr_start_seed = start_seed + command_count *
245         ↵ (end_seed_counts+1)
246     curr_end_seed = curr_start_seed + end_seed_events
247     validate_rate_command = base_command+" \"\
248         +event_directory+" \"\
249         +event_filenames[1]+"SDMode"+run_mode_string+" \"\
250         +str(output_mode)+" \"\
251         +str(deterministic_mode)+" \"\
252         +str(fixture)+" \"\
253         +str(ath5founder)+" \"\
254         +str(debug_output)+" \"\
255         +str(curr_start_seed)+" \"\
256         +str(curr_end_seed)+" \"\
257         +str(0)+" \"\
258         +str(earliest_lineage_start_time)+" \"\
259         +str(latest_lineage_start_time)+" \"\
260         +str(events_end_time)+" "
261
262     curr_list.append(validate_rate_command)
263     command_count += 1
264
265     for i in range(0,len(curr_list)):
266         if run_modes[m] == 2:
267             curr_list[i] = curr_list[i] + original_theta_string
268         if run_modes[m] == 0:
269             curr_list[i] = curr_list[i] + stochastic_theta_string
270         if run_modes[m] == 1:
271             curr_list[i] = curr_list[i] + deterministic_theta_string
272
273     command_list = command_list + curr_list
274
275     # Use processes equal to the number of cpus available
276     cpu_count = multiprocessing.cpu_count()
277
278     print(command_list)
279
280     print("Starting simulations with " + str(cpu_count) + " processes")
281
282     # Generate a pool of workers
283     pool = multiprocessing.Pool(processes=cpu_count)

```

```

284     # Pass the list of bash commands to the pool, block until pool is
285     # complete
286     pool.map(execute_command, command_list, 1)
287
288 # This is a helper function for run_simulation that runs bash commands in
289 # separate processes
290 def execute_command(cmd):
291     print("Executing command: " + cmd)
292     return subprocess.call(cmd, shell=True)
293
294 if __name__ == "__main__":
295     main()

```

---

### 17.1.8 /python\_fixtures/Kolmogorov\_fixture.py

```

1 import multiprocessing
2 import os
3 import subprocess
4 import datetime
5 import numpy as np
6 from imageio.plugins._bsdf import BsdfSerializer
7
8 gomes_executable =
9     '/home/main/chaste_build/projects/ISP/apps/GomesSimulator'
10 he_executable = '/home/main/chaste_build/projects/ISP/apps/HeSimulator'
11 boije_executable =
12     '/home/main/chaste_build/projects/ISP/apps/BoijeSimulator'
13
14 empirical_data =
15     '/home/main/git/chaste/projects/ISP/empirical_data/empirical_lineages.csv'
16
17 if not(os.path.isfile(gomes_executable)):
18     raise Exception('Could not find executable: ' + gomes_executable)
19 if not(os.path.isfile(he_executable)):
20     raise Exception('Could not find executable: ' + he_executable)
21 if not(os.path.isfile(boije_executable)):
22     raise Exception('Could not find executable: ' + boije_executable)
23
24 #####
25 # SIMULATION PARAMETERS

```

```

23 #####
24
25 #Define start and end RNG seeds; determines:
26 #No. lineages per loss function run
27 #unique sequence of RNG results for each lineage
28
29 directory = "KolmogorovSequences"
30 empirical_sequences_name = "E0sequences"
31 output_mode = 2 #sequence sampler
32 debug_output = 0 #0=off;1=on
33 start_seed = 0
34 end_seed = 9999
35
36 #####
37 #GLOBAL MODEL PARAMETERS
38 #####
39
40 number_traversal_lineages = 60
41
42 #####
43 # SPECIFIC MODEL PARAMETERS
44 #####
45
46 #GOMES MODEL
47 g_end_time = 480
48 g_normal_mean = 3.9716
49 g_normal_sigma = .32839
50 g_pPP = .055
51 g_pPD = .221
52 g_pBC = .128
53 g_pAC = .106
54 g_pMG = .028
55
56 g_string = gomes_executable + " " + directory + " Gomes " +
   ↵ str(output_mode) + " " + str(debug_output) + " " + str(start_seed) +
   ↵ " " + str(end_seed) + " " + str(g_end_time) + " " +
   ↵ str(g_normal_mean) + " " + str(g_normal_sigma) + " " + str(g_pPP) + " "
   ↵ " " + str(g_pPD) + " " + str(g_pBC) + " " + str(g_pAC) + " " +
   ↵ str(g_pMG)
57
58 #HE MODEL - GENERAL
59 h_fixture = 2
60 h_ath5founder = 0

```

```

61 h_start_time = 0
62 h_end_time = 80
63
64 #HE MODEL - STOCHASTIC
65 h_deterministic_mode = 0
66 h_mitotic_mode_phase_2 = 8 #These are phase lengths, so
67 h_mitotic_mode_phase_3 = 7 #Phase3 boundary = mmp2 + mmp3
68 h_phase_1_pPP = 1.0
69 h_phase_1_pPD = 0.0
70 h_phase_2_pPP = 0.2
71 h_phase_2_pPD = 0.4
72 h_phase_3_pPP = 0.2
73 h_phase_3_pPD = 0.0
74
75 h_string = he_executable + " " + directory + " He " + str(output_mode) +
    " " + str(h_deterministic_mode) + " " + str(h_fixture) + " " +
    str(h_ath5founder) + " " + str(debug_output) + " " + str(start_seed)
    + " " + str(end_seed) + " " + str(h_start_time) + " " +
    str(h_start_time) + " " + str(h_start_time+1) + " " + str(h_end_time)
    + " " + str(h_mitotic_mode_phase_2) + " " +
    str(h_mitotic_mode_phase_3) + " " +str(h_phase_1_pPP) + " " +
    str(h_phase_1_pPD) + " " + str(h_phase_2_pPP) + " " +
    str(h_phase_2_pPD) + " " + str(h_phase_3_pPP) + " " +
    str(h_phase_3_pPD)
76
77 #HE MODEL -STOCHASTIC (REFIT)
78 hr_mitotic_mode_phase_2 = 4.1483 #These are phase lengths, so
79 hr_mitotic_mode_phase_3 = 11.6416 #Phase3 boundary = mmp2 + mmp3
80 hr_phase_1_pPP = 1.0
81 hr_phase_1_pPD = 0.0
82 hr_phase_2_pPP = 0.1959
83 hr_phase_2_pPD = 0.5168
84 hr_phase_3_pPP = 0.2934
85 hr_phase_3_pPD = 0.0
86

```

```

87 hr_string = he_executable + " " + directory + " HeRefit " +
→ str(output_mode) + " " + str(h_deterministic_mode) + " " +
→ str(h_fixture) + " " + str(h_ath5founder) + " " + str(debug_output) +
→ " " + str(start_seed) + " " + str(end_seed) + " " + str(h_start_time)
→ + " " + str(h_start_time) + " " + str(h_start_time+1) + " " +
→ str(h_end_time) + " " + str(hr_mitotic_mode_phase_2) + " " +
→ str(hr_mitotic_mode_phase_3) + " " + str(hr_phase_1_pPP) + " " +
→ str(hr_phase_1_pPD) + " " + str(hr_phase_2_pPP) + " " +
→ str(hr_phase_2_pPD) + " " + str(hr_phase_3_pPP) + " " +
→ str(hr_phase_3_pPD)

88
89 #HE MODEL - DETERMINISTIC ALTERNATIVE
90 d_deterministic_mode = 1
91 #Phase boundary shift parameters
92 d_phase_1_shape = 3.7371
93 d_phase_1_scale = 1.8114
94 d_phase_2_shape = 2.5769
95 d_phase_2_scale = 1.6814
96 d_phase_sister_shift_widths = 1.6326
97 d_phase_offset = 1.2333

98
99 d_string = he_executable + " " + directory + " Deterministic " +
→ str(output_mode) + " " + str(d_deterministic_mode) + " " +
→ str(h_fixture) + " " + str(h_ath5founder) + " " + str(debug_output) +
→ " " + str(start_seed) + " " + str(end_seed) + " " + str(h_start_time)
→ + " " + str(h_start_time) + " " + str(h_start_time+1) + " " +
→ str(h_end_time) + " " + str(d_phase_1_shape) + " " +
→ str(d_phase_1_scale) + " " + str(d_phase_2_shape) + " " +
→ str(d_phase_2_scale) + " " + str(d_phase_sister_shift_widths) + " " +
→ str(d_phase_offset)

100
101 #BOIJE MODEL
102 b_end_generation = 250
103 b_phase_2_generation = 3
104 b_phase_3_generation = 5
105 b_pAtoh7 = .32
106 b_pPtf1a = .3
107 b_png = .8
108

```

```

109 b_string = boije_executable + " " + directory + " Boije " +
→   str(output_mode) + " " + str(debug_output) + " " + str(start_seed) +
→   " " + str(end_seed) + " " + str(b_end_generation) + " " +
→   str(b_phase_2_generation) + " " + str(b_phase_3_generation) + " " +
→   str(b_pAtoh7) + " " + str(b_pPtf1a) + " " + str(b_png)

110
111 #setup the log file, appending to any existing results
112 e_filename =
→   "/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/"
→   +directory +"/" + empirical_sequences_name
113 os.makedirs(os.path.dirname(e_filename), exist_ok=True)

114
115 e_file = open(e_filename, "w")
116 e_file.write("Entry\tSequence\n")

117
118 def main():

119
120     command_list = []
121     command_list.append(g_string)
122     command_list.append(h_string)
123     command_list.append(hr_string)
124     command_list.append(d_string)
125     command_list.append(b_string)

126
127     # Use processes equal to the number of cpus available
128     cpu_count = multiprocessing.cpu_count()

129
130     print(command_list)

131
132     print("Starting simulations with " + str(cpu_count) + " processes")

133
134     # Generate a pool of workers
135     pool = multiprocessing.Pool(processes=cpu_count)

136
137     # Pass the list of bash commands to the pool, block until pool is
→       complete
138     pool.map(execute_command, command_list, 1)

139
140     traverse_lineages(empirical_data)

141
142 def traverse_lineages(data_filename):

143
144     mode_sequences = [None]*(end_seed+1)

```

```

145
146     #load lineage tracing data
147     e_data = np.loadtxt(data_filename, skiprows = 1, usecols = (0,1,2,3))
148
149     #reproducible RNG
150     p_RNG = np.random.RandomState(seed = 0)
151
152     k=0
153
154     for curr_seed in range (start_seed,end_seed+1):
155         random_lineage_number =
156             ↪ p_RNG.randint(1,number_traversal_lineages)
157         lineage_events =
158             ↪ np.array(e_data[np.where(e_data[:,0]==random_lineage_number)])
159
160         #find mitotic mode of first event and write to log
161         current_event = 1
162         mitotic_mode =
163             ↪ int(lineage_events[np.where(lineage_events[:,1]==current_event)][:,3])
164         mode_sequences[k] = f'{mitotic_mode:.0f}'
165
166         while mitotic_mode != 2:
167             if mitotic_mode == 1 and p_RNG.random_sample() < .5: break
168
169             child_events =
170                 ↪ lineage_events[np.where(lineage_events[:,2]==current_event)]
171             random_child_row =
172                 ↪ p_RNG.randint(0,np.ma.size(child_events,0))
173             current_event= child_events[random_child_row,1]
174             mitotic_mode = child_events[random_child_row,3]
175             mode_sequences[k] = mode_sequences[k] + f'{mitotic_mode:.0f}'

176             e_file.write(str(k+1)+"\t"+mode_sequences[k]+\n)
177             k += 1
178
179     return mode_sequences
180
181
182     # This is a helper function for run_simulation that runs bash commands in
183     ↪ separate processes
184     def execute_command(cmd):
185         print("Executing command: " + cmd)
186         return subprocess.call(cmd, shell=True)

```

```
182 if __name__ == "__main__":
183     main()
```

---

### 17.1.9 /python\_fixtures/SPSA\_fixture.py

```
1 import multiprocessing
2 import os
3 import subprocess
4 import datetime
5
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from scipy.stats import bernoulli
9 from fileinput import filename
10 from statsmodels.sandbox.distributions.gof_new import a_st70_upp
11
12 executable = '/home/main/chaste_build/projects/ISP/apps/HeSimulator'
13
14 if not(os.path.isfile(executable)):
15     raise Exception('Could not find executable: ' + executable)
16
17 #####
18 # SPSA COEFFICIENTS
19 #####
20
21 a_s = .025
22 c_s = .1
23
24 a_d = .0019
25 c_d = .1
26
27 A = 20 #10% of expected # iterations
28 alpha = .602
29 gamma = .101
30
31 #####
32 # SPSA & AIC UTILITY VARS
33 #####
34
35 max_iterations = 199
```

```

36 number_comparisons = 3030 #total number of comparison points between
   ↵ model output and empirical data (1000 per induction time, 10 per rate
   ↵ mode)
37 number_comparisons_per_induction = 1000
38 error_samples = 5000 #number of samples to draw when estimating
   ↵ plausibility interval for simulations
39
40 #####
41 # SIMULATION PARAMETERS
42 #####
43
44 #Define start and end RNG seeds; determines:
45 #No. lineages per loss function run
46 #unique sequence of RNG results for each lineage
47 start_seed = 0
48 end_seed = 249
49 rate_end_seed = 99
50 directory_name = "SPSA"
51 file_name_he = "HeSPSA"
52 file_name_det = "DetSPSA"
53 log_name = "HeSPSAOutput"
54 deterministic_modes = [0, 1] #1=det. mode enabled
55 count_output_mode = 0 #0=lineage counts;1=mitotic event
   ↵ logging;2=sequence sampling
56 event_output_mode = 1
57 fixture = 0 #0=He 2012;1=Wan 2016
58 debug_output = 0 #0=off;1=on
59 ath5founder = 0 #0=no morpholino 1=ath5 morpholino
60
61 #####
62 #GLOBAL MODEL PARAMETERS
63 #####
64
65 #Values defining different marker induction timepoints & relative start
   ↵ time of TiL counter
66 earliest_lineage_start_time = 23.0 #RPCs begin to enter "He model regime"
   ↵ nasally at 23hpf
67 latest_lineage_start_time = 39.0 #last temporal retinal RPC has entered
   ↵ "He model regime" at 39hpf
68 induction_times = [ 24, 32, 48 ]
69 end_time = 72.0
70 rate_end_time = 80
71

```

```

72 #####
73 # SPECIFIC MODEL PARAMETERS - THETAHAT
74 #####
75
76 #STOCHASTIC MITOTIC MODE
77 #mitotic mode per-phase probabilities
78 #3-phase mitotic mode time periodisation
79 mitotic_mode_phase_2 = 8 #These are phase lengths, so
80 mitotic_mode_phase_3 = 7 #Phase3 boundary = mmp2 + mmp3
81 phase_1_pPP = 1.0
82 phase_1_pPD = 0.0
83 phase_2_pPP = 0.2
84 phase_2_pPD = 0.4
85 phase_3_pPP = 0.2
86 phase_3_pPD = 0.0
87 he_model_params = 15
88
89 #DETERMINISTIC MITOTIC MODE
90 #Phase boundary shift parameters
91 phase_1_shape = 3
92 phase_1_scale = 2
93 phase_2_shape = 2
94 phase_2_scale = 2
95 phase_sister_shift_widths = .25
96 phase_offset = 0
97 det_model_params = 13
98
99 #array "theta_spsa" is manipulated during SPSA calculations, these are
    # starting point references
100 stochastic_theta_zero = np.array([mitotic_mode_phase_2,
    # mitotic_mode_phase_3, phase_2_pPP, phase_2_pPD, phase_3_pPP ])
101 determinisitic_theta_zero = np.array([phase_1_shape, phase_1_scale,
    # phase_2_shape, phase_2_scale, phase_sister_shift_widths,
    # phase_offset])
102
103 #scales ak gain sequence for probability variables
104 prob_scale_vector = np.array([ 1, 1, .1, .1, .1])
105 shift_scale_vector = np.array([1, 1, 1, 1, 1, 3])
106
107 #####
108 # HE ET AL EMPIRICAL RESULTS
109 #####
110

```

```

111 count_bin_sequence = np.arange(1,32,1)
112 count_x_sequence = np.arange(1,31,1)
113 count_trim_value = 30
114
115 rate_bin_sequence = np.arange(30,85,5)
116 rate_x_sequence = np.arange(30,80,5)
117 rate_trim_value = 10
118
119 #Count probability arrays
120 raw_counts =
    ↵ np.loadtxt('/home/main/git/chaste/projects/ISP/empirical_data/empirical_counts.csv',
    ↵ skiprows=1, usecols=(3,4,5,6,7,8,9,10)) #collect the per-cell-type
    ↵ counts
121 raw_counts_24 = raw_counts[0:64,:]
122 raw_counts_32 = raw_counts[64:233,:]
123 raw_counts_48 = raw_counts[233:396,:]
124
125 prob_24,bin_edges =
    ↵ np.histogram(np.sum(raw_counts_24,axis=1),count_bin_sequence,density=True)
    ↵ #obtain probability density histogram for counts, retabulating by
    ↵ summing across types
126 prob_32,bin_edges =
    ↵ np.histogram(np.sum(raw_counts_32,axis=1),count_bin_sequence,density=True)
127 prob_48,bin_edges =
    ↵ np.histogram(np.sum(raw_counts_48,axis=1),count_bin_sequence,density=True)
128
129 #extend histogram to 1000 - allows large lineages of some iterates to be
    ↵ included in AIC comparison
130 prob_empirical_24 = np.concatenate([prob_24,
    ↵ np.zeros(int(number_comparisons_per_induction) - prob_24.size)])
131 prob_empirical_32 = np.concatenate([prob_32,
    ↵ np.zeros(int(number_comparisons_per_induction) - prob_32.size)])
132 prob_empirical_48 = np.concatenate([prob_48,
    ↵ np.zeros(int(number_comparisons_per_induction) - prob_48.size)])
133
134 count_prob_list = [prob_empirical_24, prob_empirical_32,
    ↵ prob_empirical_48]
135
136 #no. of lineages observed per induction timepoint/event group
137 lineages_sampled_24 = 64
138 lineages_sampled_32 = 169
139 lineages_sampled_48 = 163
140 lineages_sampled_events = 60

```

```

141 lineages_sampled_list =
142     ↳ [lineages_sampled_24,lineages_sampled_32,lineages_sampled_48]
143
144 #Mitotic mode rate probability arrays
145 raw_events =
146     ↳ np.loadtxt('/home/main/git/chaste/projects/ISP/empirical_data/empirical_lineages.
147     ↳ skiprows=1, usecols=(3,5,8))
148 observed_events = raw_events[np.where(raw_events[:,2]==1)] #exclude any
149     ↳ mitosis whose time was too early for recording
150
151 observed_PP = observed_events[np.where(observed_events[:,0]==0)]
152 observed_PD = observed_events[np.where(observed_events[:,0]==1)]
153 observed_DD = observed_events[np.where(observed_events[:,0]==2)]
154
155 histo_PP,bin_edges =
156     ↳ np.histogram(observed_PP,rate_bin_sequence,density=False)
157 histo_PD,bin_edges =
158     ↳ np.histogram(observed_PD,rate_bin_sequence,density=False)
159 histo_DD,bin_edges =
160     ↳ np.histogram(observed_DD,rate_bin_sequence,density=False)
161
162 #hourly per-lineage probabilities- NOT probability density function
163 prob_empirical_PP = np.array((histo_PP/lineages_sampled_events)/5)
164 prob_empirical_PD = np.array((histo_PD/lineages_sampled_events)/5)
165 prob_empirical_DD = np.array((histo_DD/lineages_sampled_events)/5)
166
167 event_prob_list = [prob_empirical_PP,prob_empirical_PD,prob_empirical_DD]
168
169 #setup the log file, appending to any existing results
170 log_filename =
171     ↳ "/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/"
172     ↳ +directory_name +"/" + log_name
173 os.makedirs(os.path.dirname(log_filename), exist_ok=True)
174
175 log = open(log_filename,"w")
176
177 #plotting utility stuff
178 #interactive plot mode
179 plt.ion()
180
181 #setup new iterate plot
182 fig, ((plt0, plt1, plt2),(plt3, plt4, plt5)) =
183     ↳ plt.subplots(2,3,figsize=(12,6))
184 plot_list = [plt0,plt1,plt2,plt3,plt4,plt5]

```

```

174
175
176 def main():
177     global theta_spsa,
178         ↳ a,c,file_name,scale_vector,end_seed,rate_end_seed,alpha,gamma
179
180     for m in range(0,len(deterministic_modes)):
181
182         #traceable RNG
183         p_RNG = np.random.RandomState(seed=786)
184
185         deterministic_mode = deterministic_modes[m]
186         if deterministic_mode == 0:
187             theta_spsa = stochastic_theta_zero
188             a = a_s
189             c = c_s
190             file_name = file_name_he
191             scale_vector = prob_scale_vector
192             now = datetime.datetime.now()
193             log.write("Began SPSA optimisation of He model @\n" +
194             ↳ str(datetime.datetime.now()) + "\n")
195             log.write("k\tphase2\tphase3\tPP2\tPD2\tPP3\n")
196             number_params = he_model_params
197             if deterministic_mode == 1:
198                 theta_spsa = determinisitic_theta_zero
199                 a = a_d
200                 c = c_d
201                 file_name = file_name_det
202                 scale_vector = shift_scale_vector
203                 now = datetime.datetime.now()
204                 log.write("Began SPSA optimisation of deterministic model
205                 ↳ @\n" + str(datetime.datetime.now()) + "\n")
206                 log.write("k\tP1Sh\tP1Sc\tP2Sh\tP2Sc\tSisterShift\tOffset\n")
207                 number_params = det_model_params
208
209                 #SPSA algorithm iterator k starts at 0
210                 k=0
211
212                 while k <= max_iterations:
213                     if k == 0:
214                         end_seed = 249
215                         rate_end_seed = 99

```

```

214     #@defined iterate, increase # of seed to decrease RNG noise
215     #> to low level
216     if k == 169:
217         end_seed = 999
218         rate_end_seed = 249
219
220     #@defined iterate, increase # of seeds to decrease RNG noise
221     #> to close to nil, switch to asymptotically optimal alpha
222     #> and gamma
223     if k == 189:
224         end_seed = 4999
225         rate_end_seed = 1249
226         alpha = 1
227         gamma = (1/6)
228
229     #write the parameter set to be evaluated to file
230     if deterministic_mode == 0: log.write(str(k) + "\t" +
231     #> str(theta_spsa[0]) + "\t" + str(theta_spsa[1]) + "\t" +
232     #> str(theta_spsa[2]) + "\t" + str(theta_spsa[3]) + "\t" +
233     #> str(theta_spsa[4]) + "\n")
234     if deterministic_mode == 1: log.write(str(k) + "\t" +
235     #> str(theta_spsa[0]) + "\t" + str(theta_spsa[1]) + "\t" +
236     #> str(theta_spsa[2]) + "\t" + str(theta_spsa[3]) + "\t" +
237     #> str(theta_spsa[4]) + "\t" + str(theta_spsa[5]) + "\n")
238
239     #populate the deltak perturbation vector with samples from a
240     #> .5p bernoulli +1 -1 distribution
241     delta_k = bernoulli.rvs(.5, size=theta_spsa.size,
242     #> random_state=p_RNG)
243     delta_k[delta_k == 0] = -1
244
245     ak = a / ((A + k + 1)**alpha) #calculate ak from gain
246     #> sequence
247     scaled_ak = ak * scale_vector #scale ak appropriately for
248     #> parameters expressed in hrs & percent
249
250     ck = c / ((k + 1)**gamma) #calculate ck from gain sequence
251     scaled_ck = ck * scale_vector
252
253     #Project theta_spsa into space bounded by ck to allow
254     #> gradient sampling at bounds of probability space

```

```

242     projected_theta = project_theta(theta_spsa, scaled_ck,
243                                     ↵ deterministic_mode)
244
245     #Calculate theta+ and theta- vectors for gradient estimate
246     theta_plus = projected_theta + scaled_ck * delta_k
247     theta_minus = projected_theta - scaled_ck * delta_k
248
249     AIC_gradient = evaluate_AIC_gradient(k, theta_plus,
250                                         ↵ theta_minus, deterministic_mode, number_params,
251                                         ↵ file_name)
252
253     ghat = ((AIC_gradient) / (2 * ck)) * delta_k
254
255     log.write("g0: " + str(ghat[0]) + "\n")
256
257     #update new theta_spsa
258     theta_spsa = theta_spsa - scaled_ak * ghat
259
260     #constrain updated theta_spsa
261     theta_spsa = project_theta(theta_spsa,
262                                 ↵ np.zeros(theta_spsa.size), deterministic_mode)
263
264     k+=1
265
266     #write final result and close log
267     if deterministic_mode == 0: log.write(str(k) + "\t" +
268                                         ↵ str(theta_spsa[0]) + "\t" + str(theta_spsa[1]) + "\t" +
269                                         ↵ str(theta_spsa[2]) + "\t" + str(theta_spsa[3]) + "\t" +
270                                         ↵ str(theta_spsa[4]) + "\n")
271     if deterministic_mode == 1: log.write(str(k) + "\t" +
272                                         ↵ str(theta_spsa[0]) + "\t" + str(theta_spsa[1]) + "\t" +
273                                         ↵ str(theta_spsa[2]) + "\t" + str(theta_spsa[3]) + "\t" +
274                                         ↵ str(theta_spsa[4]) + "\t" + str(theta_spsa[5]) + "\n")
275
276     log.close
277
278 def project_theta(theta, boundary, deterministic_mode):
279     # Required projection is onto unit triangle modified by boundary (ck
280     ↵ value)
281
282     # Values outside the lower bounds are reset first
283     # Then, for phase 2 probabilities, we project (PPa,PDa)→(PPb,PDb) such
284     ↵ that if( (PPa+ck) + (PDa+ck) >1), PPb+ck + PDb +ck = 1.
285
286     # This takes care of the upper bound

```

```

273
274     #project all negative parameters back into bounded space
275     if deterministic_mode == False:
276         for i in range(0, theta.size):
277             if i == 0:
278                 if theta[i] < boundary[i] + 4.0: theta[i] = boundary[i] +
279                     → 4.0 #project mitotic_mode_phase_2 to a minimum of 4-
280                     → below this param has no effect (due to refractory
281                     → period after first division)
282             else:
283                 if theta[i] < boundary[i]: theta[i] = boundary[i]
284
285             #if PP3 exceeds 1-boundary, project back to 1-boundary
286             if theta[4] > 1 - boundary[4]: theta[4] = 1 - boundary[4]
287
288             if theta[2] + theta[3] > 1 - boundary[2]: #if pPP2 + pPD2 gives a
289                 → total beyond the current boundary-reduced total of 1.0
290
291             if theta[2] > (1 - 2 * boundary[2]) and theta[2] ≤ theta[3]
292                 → - (1 - 2 * boundary[2]): # these (PP,PD) points will
293                 → project into negative PD space
294                 theta[2] = 1 - 2 * boundary[2]
295                 theta[3] = boundary[2]
296
297             elif theta[3] > (1 - 2 * boundary[2]) and theta[2] ≤
298                 → theta[3] - (1 - 2 * boundary[2]): # these (PP,PD) points
299                 → will project into negative PP space
300                 theta[3] = 1 - 2 * boundary[2]
301                 theta[2] = boundary[2]
302
303             else: # the (PP,PD) points can be projected onto the line PD
304                 → = -PP + 1 and modified by the boundary;
305                 v1 = [ 1, -1 ] # vector from (0,1) to (1,0)
306                 v2 = [ theta[2], theta[3] - 1 ] #vector from (0,1) to
307                     → (PP,PD)
308                 dot_product = np.dot(v1,v2)
309                 lengthv1 = 2
310                 theta[2] = (dot_product / lengthv1) - boundary[2]
311                 theta[3] = (1 - dot_product / lengthv1) - boundary[2]
312
313             if deterministic_mode == True:
314                 for i in range(0, theta.size-1): #exclude offset param

```

```

305         if theta[i] < boundary[i]: theta[i] = boundary[i] + 0.1
306             ↵ #prevents non→0 arguments for deterministic mode
307
308     #projects sister shift value such that 95% of sister shift values
309     ↵ will be less than smallest mean phase time
310     minPhase = min(theta[0]*theta[1],theta[2]*theta[3])
311     if theta[4] > minPhase/2 - boundary[4]: theta[4] = minPhase/2 -
312         ↵ boundary[4]
313
314     return theta
315
316
317 def evaluate_AIC_gradient(k, theta_plus, theta_minus, deterministic_mode,
318     ↵ number_params, file_name):
319     #Form the simulator commands for current thetas and deterministic
320     ↵ modes
321     command_list = []
322     base_command = executable
323
324     rate_settings = str(event_output_mode)+" "\
325         +str(deterministic_mode)+" "\
326         +str(fixture)+" "\
327         +str(ath5founder)+" "\
328         +str(debug_output)+" "\
329         +str(start_seed)+" "\
330         +str(rate_end_seed)+" "\
331         +str(earliest_lineage_start_time)+" "\
332         +str(earliest_lineage_start_time)+" "\
333         +str(latest_lineage_start_time)+" "\
334         +str(rate_end_time)+" "
335
336     count_settings_1 = str(count_output_mode)+" "\
337         +str(deterministic_mode)+" "\
338         +str(fixture)+" "\
339         +str(ath5founder)+" "\
340         +str(debug_output)+" "\
341         +str(start_seed)+" "\
342         +str(end_seed)+" "
343
344     count_settings_2 = str(earliest_lineage_start_time)+" "\
345         +str(latest_lineage_start_time)+" "\
346         +str(end_time)+" "
347
348     if deterministic_mode == 0:

```

```

343
344     stochastic_params_plus = str(theta_plus[0])+" "+\
345         +str(theta_plus[1])+" "+\
346         +str(phase_1_pPP)+" "+\
347         +str(phase_1_pPD)+" "+\
348         +str(theta_plus[2])+" "+\
349         +str(theta_plus[3])+" "+\
350         +str(theta_plus[4])+" "+\
351         +str(phase_3_pPD)

352
353     stochastic_params_minus = str(theta_minus[0])+" "+\
354         +str(theta_minus[1])+" "+\
355         +str(phase_1_pPP)+" "+\
356         +str(phase_1_pPD)+" "+\
357         +str(theta_minus[2])+" "+\
358         +str(theta_minus[3])+" "+\
359         +str(theta_minus[4])+" "+\
360         +str(phase_3_pPD)

361
362     command_rate_plus = base_command\
363         +" "+directory_name+" "+file_name+"RatePlus "+\
364         +rate_settings\
365         +stochastic_params_plus

366
367     command_rate_minus = base_command\
368         +" "+directory_name+" "+file_name+"RateMinus "+\
369         +rate_settings\
370         +stochastic_params_minus

371
372     command_list.append(command_rate_plus)
373     command_list.append(command_rate_minus)

374
375     for i in range(0,len(induction_times)):
376         command_plus = base_command\
377             +" "+directory_name+
378             " "+file_name+str(induction_times[i])+"Plus "+\
379             +count_settings_1\
380             +str(induction_times[i])+" "+\
381             +count_settings_2\
382             +stochastic_params_plus

383     command_minus = base_command\

```

```

384             + " "+directory_name+
385             ↵ "+file_name+str(induction_times[i])+"Minus "\n
386             +count_settings_1\
387             +str(induction_times[i])+ " "\n
388             +count_settings_2\
389             +stochastic_params_minus

390         command_list.append(command_plus)
391         command_list.append(command_minus)

392     if deterministic_mode == 1:

393         deterministic_params_plus = str(theta_plus[0])+" "\n
394             +str(theta_plus[1])+" "\n
395             +str(theta_plus[2])+" "\n
396             +str(theta_plus[3])+" "\n
397             +str(theta_plus[4])+" "\n
398             +str(theta_plus[5])

399         deterministic_params_minus = str(theta_minus[0])+" "\n
400             +str(theta_minus[1])+" "\n
401             +str(theta_minus[2])+" "\n
402             +str(theta_minus[3])+" "\n
403             +str(theta_minus[4])+" "\n
404             +str(theta_minus[5])

405         command_rate_plus = base_command\
406             +" "+directory_name+" "+ file_name+"RatePlus "\n
407             +rate_settings\
408             +deterministic_params_plus

409         command_rate_minus = base_command\
410             +" "+directory_name+" "+ file_name+"RateMinus "\n
411             +rate_settings\
412             +deterministic_params_minus

413         command_list.append(command_rate_plus)
414         command_list.append(command_rate_minus)

415     for i in range(0,len(induction_times)):
416         command_plus = base_command\
417             +" "+directory_name+
418             ↵ "+file_name+str(induction_times[i])+"Plus "\n

```

```

425                     +count_settings_1\
426                     +str(induction_times[i])+ " "\
427                     +count_settings_2\
428                     +deterministic_params_plus
429
430             command_minus = base_command\
431                 + " "+directory_name+
432                 ↳ " "+file_name+str(induction_times[i])+"Minus "\
433                     +count_settings_1\
434                     +str(induction_times[i])+ " "\
435                     +count_settings_2\
436                     +deterministic_params_minus
437
438             command_list.append(command_plus)
439             command_list.append(command_minus)
440
441
442     # Use processes equal to the number of cpus available
443     cpu_count = multiprocessing.cpu_count()
444
445     print("Starting simulations for iterate " + str(k) + " with " +
446           ↳ str(cpu_count) + " processes, " + str(end_seed+1) + " lineages
447           ↳ simulated for counts, " + str(rate_end_seed+1) + " lineages for
448           ↳ events")
449
450     log.flush() #required to prevent pool from jamming up log for some
451           ↳ reason
452
453     # Generate a pool of workers
454     pool = multiprocessing.Pool(processes=cpu_count)
455
456     # Pass the list of bash commands to the pool, block until pool is
457       ↳ complete
458     pool.map(execute_command, command_list, 1)
459
460     #these numpy arrays hold the individual RSS vals for timepoints +
461       ↳ rates
462     rss_plus = np.zeros(len(induction_times)+3)
463     rss_minus = np.zeros(len(induction_times)+3)
464
465     #clear the plots on the interactive figure
466     for i in range(0, len(plot_list)):
467         plt.sca(plot_list[i])

```



```

486     histo_rate_plus, bin_edges = np.histogram(mode_rate_plus[:,0],
487         ↵ rate_bin_sequence, density=False)
487     histo_rate_minus, bin_edges = np.histogram(mode_rate_minus[:,0],
488         ↵ rate_bin_sequence, density=False)

488
489     #hourly per-lineage probabilities
490     prob_histo_rate_plus = np.array(histo_rate_plus / ((rate_end_seed
491         ↵ + 1)*5))
491     prob_histo_rate_minus = np.array(histo_rate_plus /
492         ↵ ((rate_end_seed + 1)*5))

492
493     residual_plus = prob_histo_rate_plus - event_prob_list[i]
494     residual_minus = prob_histo_rate_minus - event_prob_list[i]

495
496     #PD RESIUDAL WEIGHTING
497     if i == 1:
498         rss_plus[i+3] = np.sum(1.5*np.square(residual_plus))
499         rss_minus[i+3] = np.sum(1.5*np.square(residual_minus))

500
501     else:
502         rss_plus[i+3] = np.sum(np.square(residual_plus))
503         rss_minus[i+3] = np.sum(np.square(residual_minus))

504
505     plotter(plot_list[i+3], mode_rate_plus[:,0],
506         ↵ mode_rate_minus[:,0],
507         ↵ event_prob_list[i],lineages_sampled_events, 1)

508     AIC_plus = 2 * number_params + number_comparisons *
509         ↵ np.log(np.sum(rss_plus))
510     AIC_minus = 2 * number_params + number_comparisons *
511         ↵ np.log(np.sum(rss_minus))

512
513     plt.show()
514
515     ↵ plt.savefig("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/"
516         ↵ +directory_name + "/" + "SDmode" + str(deterministic_mode) +
517         ↵ "iterate" + str(k) + ".png")

518
519     log.write("theta_plus:\n")
520     if deterministic_mode == 0: log.write(str(k) + "\t" +
521         ↵ str(theta_plus[0]) + "\t" + str(theta_plus[1]) + "\t" +
522         ↵ str(theta_plus[2]) + "\t" + str(theta_plus[3]) + "\t" +
523         ↵ str(theta_plus[4]) + "\n")

```

```

515     if deterministic_mode == 1: log.write(str(k) + "\t" +
516         str(theta_plus[0]) + "\t" + str(theta_plus[1]) + "\t" +
517         str(theta_plus[2]) + "\t" + str(theta_plus[3]) + "\t" +
518         str(theta_plus[4]) + "\t" + str(theta_plus[5]) +"\n")
519     log.write("theta_minus:\n")
520     if deterministic_mode == 0: log.write(str(k) + "\t" +
521         str(theta_minus[0]) + "\t" + str(theta_minus[1]) + "\t" +
522         str(theta_minus[2]) + "\t" + str(theta_minus[3]) + "\t" +
523         str(theta_minus[4]) + "\n")
524     if deterministic_mode == 1: log.write(str(k) + "\t" +
525         str(theta_minus[0]) + "\t" + str(theta_minus[1]) + "\t" +
526         str(theta_minus[2]) + "\t" + str(theta_minus[3]) + "\t" +
527         str(theta_minus[4]) + "\t" + str(theta_minus[5]) + "\n")
528
529     log.write("PositiveAIC: " + str(AIC_plus) + " NegativeAIC: " +
530         str(AIC_minus) + "\n")
531
532     AIC_gradient_sample = AIC_plus - AIC_minus;
533
534     return AIC_gradient_sample
535
536
537 #data plotter function for monitoring SPSA results
538 def plotter(subplot,plus,minus,empirical_prob,samples,mode):
539     global plt0,plt1,plt2,plt3,plt4,plt5, prob_histo_plus,
540         prob_histo_minus
541     bin_sequence = []
542     x_sequence = []
543     trim_value = 0
544
545     if mode == 0:
546         bin_sequence = count_bin_sequence
547         x_sequence = count_x_sequence
548         trim_value = count_trim_value
549         prob_histo_plus, bin_edges = np.histogram(plus, bin_sequence,
550             density=True)
551         prob_histo_minus, bin_edges = np.histogram(minus, bin_sequence,
552             density=True)
553
554     if mode == 1:
555         bin_sequence = rate_bin_sequence
556         x_sequence = rate_x_sequence
557         trim_value = rate_trim_value

```

```
544     histo_plus, bin_edges = np.histogram(plus, bin_sequence,
545         ↵ density=False)
546     histo_minus, bin_edges = np.histogram(minus, bin_sequence,
547         ↵ density=False)
548     prob_histo_plus = np.array(histo_plus / ((rate_end_seed + 1)*5))
549     prob_histo_minus = np.array(histo_plus / ((rate_end_seed + 1)*5))
550
551     trimmed_prob = empirical_prob[0:trim_value]
552
553     subplot.plot(x_sequence,trimmed_prob, 'k+')
554     plt.pause(0.0001)
555
556     interval_plus = sampler(plus,samples,bin_sequence)
557
558     subplot.plot(x_sequence,prob_histo_plus, 'g-')
559     plt.pause(0.0001)
560     subplot.fill_between(x_sequence, (prob_histo_plus - interval_plus),
561         ↵ (prob_histo_plus + interval_plus), alpha=0.2,
562         ↵ edgecolor='#008000', facecolor='#00FF00')
563     plt.pause(0.0001)
564
565     interval_minus = sampler(minus,samples,bin_sequence)
566
567     subplot.plot(x_sequence,prob_histo_minus, 'm-')
568     plt.pause(0.0001)
569     subplot.fill_between(x_sequence, (prob_histo_minus - interval_minus),
570         ↵ (prob_histo_minus + interval_minus), alpha=0.2,
571         ↵ edgecolor='#800080', facecolor='#FF00FF')
572     plt.pause(0.0001)
573
574     if subplot=plt0: plt0.set_ylim((0,.20))
575     if subplot=plt1: plt1.set_ylim(0,.20)
576     if subplot=plt2: plt2.set_ylim(0,.7)
577     if subplot=plt3: plt3.set_ylim(0, .30)
578     if subplot=plt4: plt4.set_ylim(0, .15)
579     if subplot=plt5: plt5.set_ylim(0, .35)
580
581 def sampler(data,samples,bin_sequence):
582
583     if len(data) == 0: #catches edge case w/ no entries for a mitotic
584         ↵ mode
585         data=[0]
```

```

580     base_sample=np.zeros((error_samples,len(bin_sequence)-1))
581
582     for i in range(0,error_samples):
583         new_data_sample = np.random.choice(data,samples)
584         new_histo_prob, bin_edges = np.histogram(new_data_sample,
585             ↪ bin_sequence, density=True)
586         base_sample[i,:] = new_histo_prob
587
588     sample_95CI = np.array(2 * (np.std(base_sample,0)))
589
590     return sample_95CI
591
592 # This is a helper function for run_simulation that runs bash commands in
593 # separate processes
594
595 def execute_command(cmd):
596     return subprocess.call(cmd, shell=True)
597
598 if __name__ == "__main__":
599     main()

```

---

### 17.1.10 /python\_fixtures/Wan\_output\_fixture.py

```

1 import multiprocessing
2 import os
3 import subprocess
4 import datetime
5
6 import numpy as np
7 from openpyxl.styles.builtins import output
8 from sklearn.datasets.tests.test_svmlight_format import currdir
9
10 executable = '/home/main/chaste_build/projects/ISP/apps/WanSimulator'
11
12 if not(os.path.isfile(executable)):
13     raise Exception('Could not find executable: ' + executable)
14
15 ##########
16 # SIMULATION PARAMETERS
17 #####
18
19 #Define start and end RNG seeds; determines:
20 #No. simulated CMZs per run

```

```

21 #unique sequence of RNG results for each lineage
22 start_seed = 0
23 end_seed = 99 #total seeds should be divisible by # cores
24
25 output_directory = "WanOutput"
26
27 #####
28 #MODEL PARAMETERS
29 #####
30
31 #CMZ residency time
32 wan_residency_time = 17.0
33 #factor to divide 3dpf progenitor population by to obtain estimate of
   ↳ stem cell population
34 stem_divisor = 10
35 #3dpf CMZ progenitor population mean and standard deviation
36 progenitor_mean = 792
37 progenitor_std = 160
38
39 #stem cell cycle parameters- cell cycle RV is a shifted gamma
   ↳ distribution
40 stem_gamma_shift = 4
41 stem_gamma_shape = 6.5 #mean 30 hr stem cell time - in reality is
   ↳ probably more like 60+
42 stem_gamma_scale = 4
43
44 progenitor_gamma_shift = 4 #default He et al. values, mean 6 hr cycle
   ↳ time
45 progenitor_gamma_shape = 2
46 progenitor_gamma_scale = 1
47 progenitor_gamma_sister = 1
48
49 cmz_theta_string = str(wan_residency_time) + " " + str(stem_divisor) + "
   ↳ " + str(progenitor_mean) + " " + str(progenitor_std) + " "
   ↳ str(stem_gamma_shift) + " " + str(stem_gamma_shape) + " "
   ↳ str(stem_gamma_scale) + " " + str(progenitor_gamma_shift) + " "
   ↳ str(progenitor_gamma_shape) + " " + str(progenitor_gamma_scale) + " "
   ↳ + str(progenitor_gamma_sister)
50
51 #STOCHASTIC MITOTIC MODE
52 #HE ORIGINAL PARAMETERS
53 #mitotic mode per-phase probabilities
54 #3-phase mitotic mode time periodisation

```

```

55 mitotic_mode_phase_2 = 8 #These are phase boundaries rather than lengths
  ↵ as in eg. He_output_fixture.py
56 mitotic_mode_phase_3 = 15
57 phase_1_pPP = 1.0
58 phase_1_pPD = 0.0
59 phase_2_pPP = 0.2
60 phase_2_pPD = 0.4
61 phase_3_pPP = 0.2
62 phase_3_pPD = 0.0
63
64 stochastic_theta_string = str(mitotic_mode_phase_2) + " " +
  ↵ str(mitotic_mode_phase_3) + " " +str(phase_1_pPP) + " " +
  ↵ str(phase_1_pPD) + " " + str(phase_2_pPP) + " " + str(phase_2_pPD) +
  ↵ " " + str(phase_3_pPP) + " " + str(phase_3_pPD)
65
66 def main():
67
68     command_list = []
69
70     # Use processes equal to the number of cpus available
71     cpu_count = multiprocessing.cpu_count()
72     seeds_per_command = (end_seed + 1) / cpu_count
73     base_command = executable + " " + output_directory
74
75     for i in range(0,cpu_count):
76         curr_start_seed = i * seeds_per_command
77         curr_end_seed = curr_start_seed + (seeds_per_command - 1)
78
79         command = base_command + " "\\
80             +str(curr_start_seed) + " "\\
81             +str(curr_end_seed) + " "\\
82             +cmz_theta_string + " "\\
83             +stochastic_theta_string
84
85     #command_list.append(command)
86
87     ↵ command_list.append("/home/main/chaste_build/projects/ISP/apps/WanSimulator"
     ↵ WanOutput 18.0 24.0 17.0 10 792 160 4 6.5 4 4 2 1 1 8 15 1.0 0.0
     ↵ 0.2 0.4 0.2 0.0")

```

```

88
    ↵ command_list.append( "/home/main/chaste_build/projects/ISP/apps/WanSimulator
    ↵ WanOutput 42.0 49.0 17.0 10 792 160 4 6.5 4 4 2 1 1 8 15 1.0 0.0
    ↵ 0.2 0.4 0.2 0.0" )

89
    ↵ command_list.append( "/home/main/chaste_build/projects/ISP/apps/WanSimulator
    ↵ WanOutput 51.0 57.0 17.0 10 792 160 4 6.5 4 4 2 1 1 8 15 1.0 0.0
    ↵ 0.2 0.4 0.2 0.0" )

90
    ↵ command_list.append( "/home/main/chaste_build/projects/ISP/apps/WanSimulator
    ↵ WanOutput 94.0 99.0 17.0 10 792 160 4 6.5 4 4 2 1 1 8 15 1.0 0.0
    ↵ 0.2 0.4 0.2 0.0" )

91
    ↵ command_list.append( "/home/main/chaste_build/projects/ISP/apps/WanSimulator
    ↵ WanOutput 58.0 64.0 17.0 10 792 160 4 6.5 4 4 2 1 1 8 15 1.0 0.0
    ↵ 0.2 0.4 0.2 0.0" )

92
    ↵ command_list.append( "/home/main/chaste_build/projects/ISP/apps/WanSimulator
    ↵ WanOutput 65.0 70.0 17.0 10 792 160 4 6.5 4 4 2 1 1 8 15 1.0 0.0
    ↵ 0.2 0.4 0.2 0.0" )

93
    ↵ command_list.append( "/home/main/chaste_build/projects/ISP/apps/WanSimulator
    ↵ WanOutput 71.0 74.0 17.0 10 792 160 4 6.5 4 4 2 1 1 8 15 1.0 0.0
    ↵ 0.2 0.4 0.2 0.0" )

94
print("Starting simulations with " + str(cpu_count) + " processes")

95
# Generate a pool of workers
pool = multiprocessing.Pool(processes=cpu_count)

96
97
98
99
100
# Pass the list of bash commands to the pool, block until pool is
    ↵ complete
101
pool.map(execute_command, command_list, 1)

102
103
104 # This is a helper function for run_simulation that runs bash commands in
    ↵ separate processes
105 def execute_command(cmd):
106     print("Executing command: " + cmd)
107     return subprocess.call(cmd, shell=True)

108
109
110 if __name__ == "__main__":

```

---

```
111     main()
```

---

### 17.1.11 /python\_fixtures/diagram\_utility\_scripts/Gomes\_He\_cycle\_plots.py

---

```
1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from fileinput import filename
5 from scipy.stats import lognorm
6 from scipy.stats import gamma
7 from statsmodels.sandbox.distributions.gof_new import a_st70_upp
8
9 #Model parameters
10 gomes_normal_sigma = .32839
11 gomes_normal_mu = 3.9716
12
13 he_gamma_shape = 2
14 he_gamma_scale = 1
15 he_gamma_offset = 4
16
17 def main():
18     gomes_x_range = np.arange(1,151,1)
19     gomes_y = lognorm.pdf(gomes_x_range, gomes_normal_sigma, 0,
20                           np.exp(gomes_normal_mu))
21
22     he_x_range = np.arange(1,16,.1)
23     he_y = gamma.pdf(he_x_range, he_gamma_shape, he_gamma_offset,
24                       he_gamma_scale)
25
26     fig, (ax_g, ax_h) = plt.subplots(1,2,figsize=(6,2))
27
28     ax_g.plot(gomes_x_range,gomes_y, 'k-')
29     ax_h.plot(he_x_range, he_y, 'k-')
30
31     plt.sca(ax_g)
32     plt.xlabel("Cell cycle duration (h)")
33     plt.ylabel("Probability")
34
35     plt.sca(ax_h)
36     plt.xlabel("Cell cycle duration (h)")
37     plt.ylabel("Probability")
```

```

36
37     plt.tight_layout()
38     plt.savefig('/home/main/Desktop/utility.png', transparent=True)
39     plt.show()
40
41 if __name__ == "__main__":
42     main()

```

---

### 17.1.12 /python\_fixtures/diagram\_utility\_scripts/He\_Boije\_signal\_plots.py

```

1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from fileinput import filename
5 from scipy.stats import lognorm
6 from scipy.stats import gamma
7 from statsmodels.sandbox.distributions.gof_new import a_st70_upp
8
9 #Model parameters
10 he_x = [0,8,15,25]
11 he_pp_y = [1,.2,.2,.2]
12 he_pd_y = [0,.4,0,0]
13 he_dd_y = [0,.4,.8,.8]
14
15 boije_x = [0,3,5,9]
16 boije_pAtoh7 = [0,.32,0,0]
17 boije_pPtf1a = [0,.3,0,0]
18 boije_pNg = [0,0,.8,.8]
19
20 def main():
21     fig, ((pp,pd,dd))= plt.subplots(1,3,figsize=(6,2),sharey=True)
22
23     fig2, ((pAtoh7),(pPtf1a),(pNg)) =
24         plt.subplots(3,1,figsize=(2,4),sharex=True)
25     pp.set_xlim((0,24))
26     pd.set_xlim((0,24))
27     dd.set_xlim((0,24))
28
29     pp.set_xticks([8,15])
30     pd.set_xticks([8,15])

```

```
31     dd.set_xticks([8,15])
32
33     pp.set_ylim((- .1,1.1))
34     pp.set_yticks(np.arange(0,1.2,.2))
35     pp.set_ylabel("Probability")
36     pp.set_xlabel("TiL (h)")
37     pp.axvline(8,linestyle='--',color='.1', alpha=.2)
38     pp.axvline(15,linestyle='--',color='.1', alpha=.2)
39     pp.step(he_x,he_pp_y, 'k-', where='post', linewidth=2)
40
41     pd.set_ylim((- .1,1.1))
42     pd.set_yticks(np.arange(0,1.2,.2))
43     pd.set_xlabel("TiL (h)")
44     pd.axvline(8,linestyle='--',color='.1', alpha=.2)
45     pd.axvline(15,linestyle='--',color='.1', alpha=.2)
46     pd.step(he_x,he_pd_y, 'k-', where='post', linewidth=2)
47
48     dd.set_ylim((- .1,1.1))
49     dd.set_yticks(np.arange(0,1.2,.2))
50     dd.set_xlabel("TiL (h)")
51     dd.axvline(8,linestyle='--',color='.1', alpha=.2)
52     dd.axvline(15,linestyle='--',color='.1', alpha=.2)
53     dd.step(he_x,he_dd_y, 'k-', where='post', linewidth=2)
54
55     pAtoh7.set_xlim((0,8))
56     pNg.set_xlim((0,8))
57     pPtf1a.set_xlim((0,8))
58
59     pAtoh7.set_xticks([3,5])
60     pNg.set_xticks([3,5])
61     pPtf1a.set_xticks([3,5])
62
63
64     pNg.set_xlabel("Generation")
65
66
67     pAtoh7.set_ylim((- .1,1.1))
68     pAtoh7.set_yticks(np.arange(0,1.25,.25))
69     pAtoh7.set_ylabel("Probability")
70     pAtoh7.axvline(3,linestyle='--',color='.1',alpha=.2)
71     pAtoh7.axvline(5,linestyle='--',color='.1',alpha=.2)
72     pAtoh7.step(boije_x,boije_pAtoh7, 'k-', where='post', linewidth=2)
73
```

```

74     pPtf1a.set_ylim((- .1, 1.1))
75     pPtf1a.set_yticks(np.arange(0, 1.25, .25))
76     pPtf1a.set_ylabel("Probability")
77     pPtf1a.axvline(3, linestyle='--', color='.1', alpha=.2)
78     pPtf1a.axvline(5, linestyle='--', color='.1', alpha=.2)
79     pPtf1a.step(boije_x, boije_pPtf1a, 'k-', where='post', linewidth=2)

80
81     pNg.set_ylim((- .1, 1.1))
82     pNg.set_yticks(np.arange(0, 1.25, .25))
83     pNg.set_ylabel("Probability")
84     pNg.axvline(3, linestyle='--', color='.1', alpha=.2)
85     pNg.axvline(5, linestyle='--', color='.1', alpha=.2)
86     pNg.step(boije_x, boije_pNg, 'k-', where='post', linewidth=2)

87
88
89     plt.tight_layout()
90     fig.savefig('Hemode.png', transparent=True)
91     fig2.savefig('Boijesignals.png', transparent=True)
92     plt.show()

93 if __name__ == "__main__":
94     main()

```

---

### 17.1.13 /python\_fixtures/diagram\_utility\_scripts/He\_Det\_signal\_plots.py

```

1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from fileinput import filename
5 from scipy.stats import lognorm
6 from scipy.stats import gamma
7 from statsmodels.sandbox.distributions.gof_new import a_st70_upp

8
9 #Model parameters
10 he_x = [0, 8, 15, 25]
11 he_pp_y = [1, .2, .2, .2]
12 he_pd_y = [0, .4, 0, 0]
13 he_dd_y = [0, .4, .8, .8]

14
15 d_pp_y = [1, 0, 0, 0]
16 d_pd_y = [0, 1, 0, 0]

```

```

17 d_dd_y = [0,0,1,1]
18
19 fig, ((spp,spd,sdd))= plt.subplots(1,3,figsize=(6,2),sharey=True)
20 fig2, ((dpp,dpd,ddd))= plt.subplots(1,3,figsize=(6,2),sharey=True)
21
22 for [pp,pd,dd] in [[spp,spd,sdd],[dpp,dpd,ddd]]:
23     for p in [pp,pd,dd]:
24         p.set_xlim((0,24))
25         p.set_xticks([8,15])
26         p.set_ylim((-1,1.1))
27         p.set_yticks(np.arange(0,1.2,.2))
28         p.set_xlabel("TiL (h)")
29         if p==spp or p==spd or p==sdd:
30             p.axvline(8,linestyle='--',color='.1', alpha=.2)
31             p.axvline(15,linestyle='--',color='.1', alpha=.2)
32         else:
33             p.set_xticklabels(["↔", "↔"])
34             p.tick_params(axis="x",colors="red")
35             p.axvline(8,linestyle='-',color='blue', alpha=.15,
36                         linewidth=15)
37             p.axvline(15,linestyle='-',color='blue', alpha=.15,
38                         linewidth=15)
39             pp.set_ylabel("Probability")
40             if pp==spp:
41                 pp.step(he_x,he_pp_y, 'k-', where='post', linewidth=2)
42             else:
43                 pp.step(he_x,d_pp_y, 'r-', where='post', linewidth=2)
44             if pd==spd:
45                 pd.step(he_x,he_pd_y, 'k-', where='post', linewidth=2)
46             else:
47                 pd.step(he_x,d_pd_y, 'r-', where='post', linewidth=2)
48             if dd==sdd:
49                 dd.step(he_x,he_dd_y, 'k-', where='post', linewidth=2)
50             else:
51                 dd.step(he_x,d_dd_y, 'r-', where='post', linewidth=2)
52
53 fig.tight_layout()
54 fig2.tight_layout()
55 fig.savefig('Hemode.png')
56 fig2.savefig('Detmode.png')
57 plt.show()

```

---

### 17.1.14 /python\_fixtures/figure\_plots/Cumulative\_EdU.py

---

```
1 import os
2 import numpy as np
3 import statsmodels.api as sm
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7
8 from PIL import Image
9 from io import BytesIO
10
11
12 #PLoS formatting stuff
13 plt.rcParams['font.size'] = 12
14 plt.rcParams['font.family'] = 'sans-serif'
15 plt.rcParams['font.sans-serif'] = ['Arial']
16
17 def main():
18     #Read & parse raw data file
19     raw_data =
20         → pd.read_excel('/home/main/git/chaste/projects/ISP/empirical_data/cumulative_e
21     #group data by dorsal/ventral
22     dv_grouped = raw_data.groupby('D/V')
23     #make new dataframe for totals and labelled fraction
24     totals_frame = dv_grouped.get_group('D')
25     #reset indices to ventral frame for addition operations
26     totals_frame =
27         → totals_frame.set_index(dv_grouped.get_group('V').index)
28     #sum D + V PCNA & EdU counts
29     totals_frame.PCNA = totals_frame.PCNA +
30         → dv_grouped.get_group('V').PCNA
31     totals_frame.EdU = totals_frame.EdU + dv_grouped.get_group('V').EdU
32     #create new column for labelled fraction
33     totals_frame['labelled_fraction'] = totals_frame.EdU /
34         → totals_frame.PCNA
35
36     #setup x for linear regression with y-intercept constant
37     X = totals_frame.Time
38     X = sm.add_constant(X)
39
40     model = sm.OLS(totals_frame.labelled_fraction, X).fit()
41     print(model.summary())
```

```

38     fit_results = model.params
39
40     tc = 1/fit_results[1]
41     ts = tc * fit_results[0]
42
43     sns.regplot(totals_frame.Time,totals_frame.labelled_fraction)
44
45     plt.text(7,.2,"Tc: " + f'{tc:.2f}')
46     plt.text(7,.1,"Ts: " + f'{ts:.2f}')
47
48     plt.xlabel("Time (h)")
49     plt.ylabel("Fraction of CMZ RPCs labelled by Edu")
50
51     plt.savefig("cumulative_edu.png")
52     png_memory = BytesIO()
53     plt.savefig(png_memory, format='png', dpi=600)
54     PILpng = Image.open(png_memory)
55     PILpng.save('cumulative_edu.tiff')
56     png_memory.close()
57
58     plt.show()
59
60 if __name__ == "__main__":
61     main()

```

---

### 17.1.15 /python\_fixtures/figure\_plots/He\_output\_plot.py

```

1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.patches as patches
5 from fileinput import filename
6 from statsmodels.sandbox.distributions.gof_new import a_st70_upp
7
8 from PIL import Image
9 from io import BytesIO
10
11 #PLoS formatting stuff
12 plt.rcParams['font.size'] = 10
13 plt.rcParams['font.family'] = 'sans-serif'
14 plt.rcParams['font.sans-serif'] = ['Arial']
15

```

```

16 #AIC & Plotting utility params
17 count_seeds = 10000
18 event_seeds = 1000
19 error_samples = 5000 #number of samples to draw when estimating
    ↳ plausibility interval for simulations
20
21 he_model_params = 15
22 det_model_params = 13
23
24 #stats & line_plotter utility arrays
25 bin_sequence_24_32 = np.arange(1,26,1)
26 x_sequence_24_32 = np.arange(1,25,1)
27
28 bin_sequence_48 = np.arange(1,11,1)
29 x_sequence_48 = np.arange(1,10,1)
30
31 bin_sequence_wan = np.arange(2,17,1)
32 x_sequence_wan = np.arange(2,16,1)
33
34 bin_sequence_events = np.arange(30,85,5)
35 x_sequence_events = np.arange(30,80,5)
36
37 ######
38 # HE ET AL EMPIRICAL RESULTS
39 #####
40
41 #Count probability arrays
42 raw_counts =
    ↳ np.loadtxt('/home/main/git/chaste/projects/ISP/empirical_data/empirical_counts.csv',
    ↳ skiprows=1, usecols=(3,4,5,6,7,8,9,10)) #collect the per-cell-type
    ↳ counts
43 raw_counts_24 = raw_counts[0:64,:]
44 raw_counts_32 = raw_counts[64:233,:]
45 raw_counts_48 = raw_counts[233:396,:]
46
47 prob_empirical_24,bin_edges =
    ↳ np.histogram(np.sum(raw_counts_24,axis=1),bin_sequence_24_32,density=True)
    ↳ #obtain probability density histogram for counts, retabulating by
    ↳ summing across types
48 prob_empirical_32,bin_edges =
    ↳ np.histogram(np.sum(raw_counts_32,axis=1),bin_sequence_24_32,density=True)
49 prob_empirical_48,bin_edges =
    ↳ np.histogram(np.sum(raw_counts_48,axis=1),bin_sequence_48,density=True)

```

```

50
51 #no. of lineages E0 per induction timepoint/event group
52 lineages_sampled_24 = 64
53 lineages_sampled_32 = 169
54 lineages_sampled_48 = 163
55 lineages_sampled_events = 60
56
57 #Mitotic mode rate probability arrays
58 raw_events =
  ↳ np.loadtxt('/home/main/git/chaste/projects/ISP/empirical_data/empirical_lineages.')
  ↳ skiprows=1, usecols=(3,5,8))
59 EO_events = raw_events[np.where(raw_events[:,2]==1)] #exclude any mitosis
  ↳ whose time was too early for recording
60
61 EO_PP = EO_events[np.where(EO_events[:,0]==0)]
62 EO_PD = EO_events[np.where(EO_events[:,0]==1)]
63 EO_DD = EO_events[np.where(EO_events[:,0]==2)]
64
65 prob_empirical_PP,bin_edges =
  ↳ np.histogram(EO_PP,bin_sequence_events,density=True)
66 prob_empirical_PD,bin_edges =
  ↳ np.histogram(EO_PD,bin_sequence_events,density=True)
67 prob_empirical_DD,bin_edges =
  ↳ np.histogram(EO_DD,bin_sequence_events,density=True)
68
69 empirical_fit_list =
  ↳ [prob_empirical_24,prob_empirical_32,prob_empirical_48,prob_empirical_PP,prob_emp
70
71 #mutant results
72 wt_mean_clone_size = 6
73 ath5_mean_clone_size = 8
74 wt_even_odd_ratio = 1.1
75 ath5_even_odd_ratio = 2.2
76
77 ##########
78 # WAN ET AL EMPIRICAL RESULTS
79 ##########
80 #(derived from figure)
81 prob_wan =
  ↳ [.3143,.1857,.1757,.0871,.0871,.0786,.0271,.0179,.0071,0,0,.0071];
82 lineages_sampled_wan = 40 #not reported- approximation based on 95% CI
83

```

```
84 empirical_test_list = [prob_wan, wt_mean_clone_size,
85     ↳ ath5_mean_clone_size, wt_even_odd_ratio, ath5_even_odd_ratio]
86
87 def main():
88     #LOAD & PARSE HE MODEL OUTPUT FILES
89     #original fit stochastic mitotic mode files
90     o_counts_24 =
91         ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
92     o_counts_32 =
93         ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
94     o_counts_48 =
95         ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
96     o_events =
97         ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM
98     o_events_PP = np.array(o_events[np.where(o_events[:,1]==0)])
99     o_events_PD = np.array(o_events[np.where(o_events[:,1]==1)])
100    o_events_DD = np.array(o_events[np.where(o_events[:,1]==2)])
101    o_counts_wan_no_shadow =
102        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
103        ↳ skiprows=1, usecols=3)
104    o_counts_ath5 =
105        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
106        ↳ skiprows=1, usecols=3)
107    o_counts_validate =
108        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
109        ↳ skiprows=1, usecols=3)
110
111    #refit stochastic mitotic mode files
112    s_counts_24 =
113        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
114        ↳ skiprows=1, usecols=3)
115    s_counts_32 =
116        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
117        ↳ skiprows=1, usecols=3)
118    s_counts_48 =
119        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
120        ↳ skiprows=1, usecols=3)
```

```

104 s_events =
105     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM
106     ↵ skiprows=1, usecols=(0,3))
107 s_events_PP = np.array(s_events[np.where(s_events[:,1]==0)])
108 s_events_PD = np.array(s_events[np.where(s_events[:,1]==1)])
109 s_events_DD = np.array(s_events[np.where(s_events[:,1]==2)])
110 s_counts_wan_no_shadow =
111     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
112     ↵ skiprows=1, usecols=3)
113 s_counts_ath5 =
114     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
115     ↵ skiprows=1, usecols=3)
116 s_counts_validate =
117     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
118     ↵ skiprows=1, usecols=3)
119
120 #deterministic mitotic mode files
121 d_counts_24 =
122     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
123     ↵ skiprows=1, usecols=3)
124 d_counts_32 =
125     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
126     ↵ skiprows=1, usecols=3)
127 d_counts_48 =
128     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
129     ↵ skiprows=1, usecols=3)
130 d_events =
131     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM
132     ↵ skiprows=1, usecols=(0,3))
133 d_events_PP = np.array(d_events[np.where(d_events[:,1]==0)])
134 d_events_PD = np.array(d_events[np.where(d_events[:,1]==1)])
135 d_events_DD = np.array(d_events[np.where(d_events[:,1]==2)])
136 d_counts_wan_no_shadow =
137     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
138     ↵ skiprows=1, usecols=3)
139 d_counts_ath5 =
140     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
141     ↵ skiprows=1, usecols=3)
142 d_counts_validate =
143     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
144     ↵ skiprows=1, usecols=3)
145
146 #calculate Ath5 clone size and even-odd ratios

```

```

125     o_wt_mean_clone_size = np.mean(o_counts_validate)
126     o_ath5_mean_clone_size = np.mean(o_counts_ath5)

127
128     s_wt_mean_clone_size = np.mean(s_counts_validate)
129     s_ath5_mean_clone_size = np.mean(s_counts_ath5)

130
131     d_wt_mean_clone_size = np.mean(d_counts_validate)
132     d_ath5_mean_clone_size = np.mean(d_counts_ath5)

133
134     o_wt_even_odd_ratio = len(np.where(o_counts_validate % 2 ==
135                                ↳  0)[0])/len(np.where(o_counts_validate % 2 == 1)[0])
136     o_ath5_even_odd_ratio = len(np.where(o_counts_ath5 % 2 ==
137                                ↳  0)[0])/len(np.where(o_counts_ath5 % 2 == 1)[0])

138     s_wt_even_odd_ratio = len(np.where(s_counts_validate % 2 ==
139                                ↳  0)[0])/len(np.where(s_counts_validate % 2 == 1)[0])
140     s_ath5_even_odd_ratio = len(np.where(s_counts_ath5 % 2 ==
141                                ↳  0)[0])/len(np.where(s_counts_ath5 % 2 == 1)[0])

142
143     o_fit_output_list = [o_counts_24, o_counts_32, o_counts_48,
144                           ↳  o_events_PP, o_events_PD, o_events_DD]
145     o_test_output_list = [o_counts_wan_no_shadow, o_wt_mean_clone_size,
146                           ↳  o_ath5_mean_clone_size, o_wt_even_odd_ratio,
147                           ↳  o_ath5_even_odd_ratio]

148
149     s_fit_output_list = [s_counts_24, s_counts_32, s_counts_48,
150                           ↳  s_events_PP, s_events_PD, s_events_DD]
151     s_test_output_list = [s_counts_wan_no_shadow, s_wt_mean_clone_size,
152                           ↳  s_ath5_mean_clone_size, s_wt_even_odd_ratio,
153                           ↳  s_ath5_even_odd_ratio]

154
155     d_fit_output_list = [d_counts_24, d_counts_32, d_counts_48,
156                           ↳  d_events_PP, d_events_PD, d_events_DD]
157     d_test_output_list = [d_counts_wan_no_shadow, d_wt_mean_clone_size,
158                           ↳  d_ath5_mean_clone_size, d_wt_even_odd_ratio,
159                           ↳  d_ath5_even_odd_ratio]

160
161 #SETUP FIGURES

```

```

153 #4 figures to be generated; 3 show output of individual model modes,
154   → 4th overlays stochastic refit on deterministic fit
155 o_fig, ((o_24,o_32),(o_48,o_PP),(o_PD,o_DD),(o_wan_noshad,
156   → o_ath5size),(o_ratio,o_unused)) =
157   → plt.subplots(5,2,figsize=(7.5,8.75))
158 s_fig, ((s_24,s_32),(s_48,s_PP),(s_PD,s_DD),(s_wan_noshad,
159   → s_ath5size),(s_ratio,s_unused)) =
160   → plt.subplots(5,2,figsize=(7.5,8.75))
161 d_fig, ((d_24,d_32),(d_48,d_PP),(d_PD,d_DD),(d_wan_noshad,
162   → d_ath5size),(d_ratio,d_unused)) =
163   → plt.subplots(5,2,figsize=(7.5,8.75))
164 sd_fig, ((sd_24,sd_32),(sd_48,sd_PP),(sd_PD,sd_DD),(sd_wan_noshad,
165   → sd_ath5size),(sd_ratio,sd_unused)) =
166   → plt.subplots(5,2,figsize=(7.5,8.75))

167 #turn off unused axes
168
169   → o_unused.axis('off');s_unused.axis('off');d_unused.axis('off');sd_unused.axis('off')

170 #set xlabel
171
172   → o_24.set_xlabel('Count');o_32.set_xlabel('Count');o_48.set_xlabel('Count');o_
173   → o_PP.set_xlabel('Age (hpf)');o_PD.set_xlabel('Age
174   → (hpf)');o_DD.set_xlabel('Age (hpf)')

175
176   → s_24.set_xlabel('Count');s_32.set_xlabel('Count');s_48.set_xlabel('Count');s_
177   → s_PP.set_xlabel('Age (hpf)');s_PD.set_xlabel('Age
178   → (hpf)');s_DD.set_xlabel('Age (hpf)')

179
180   → d_24.set_xlabel('Count');d_32.set_xlabel('Count');d_48.set_xlabel('Count');d_
181   → d_PP.set_xlabel('Age (hpf)');d_PD.set_xlabel('Age
182   → (hpf)');d_DD.set_xlabel('Age (hpf)')

183
184   → sd_24.set_xlabel('Count');sd_32.set_xlabel('Count');sd_48.set_xlabel('Count');
185   → sd_PP.set_xlabel('Age (hpf)');sd_PD.set_xlabel('Age
186   → (hpf)');sd_DD.set_xlabel('Age (hpf)')

187 #set ylabel
188
189   → o_24.set_ylabel('Probability');o_32.set_ylabel('Probability');o_48.set_ylabel('Probability');

```

```

177     o_PP.set_ylabel('PP Probability');o_PD.set_ylabel('PD
178         ↵ Probability');o_DD.set_ylabel('DD Probability');
179     o_ath5size.set_ylabel('Clone size');o_ratio.set_ylabel('Clone
180         ↵ even/odd ratio')

181             ↵ s_24.set_ylabel('Probability');s_32.set_ylabel('Probability');s_48.set_ylabel(
182     s_PP.set_ylabel('PP Probability');s_PD.set_ylabel('PD
183         ↵ Probability');s_DD.set_ylabel('DD Probability');
184     s_ath5size.set_ylabel('Clone size');s_ratio.set_ylabel('Clone
185         ↵ even/odd ratio')

186             ↵ d_24.set_ylabel('Probability');d_32.set_ylabel('Probability');d_48.set_ylabel(
187     d_PP.set_ylabel('PP Probability');d_PD.set_ylabel('PD
188         ↵ Probability');d_DD.set_ylabel('DD Probability');
189     d_ath5size.set_ylabel('Clone size');d_ratio.set_ylabel('Clone
190         ↵ even/odd ratio')

191
192 #PLOT DATA
193 #plot lines, CIs, and bars
194 o_objects = line_plotter(o_24, o_counts_24, bin_sequence_24_32,
195     ↵ x_sequence_24_32, count_seeds, lineages_sampled_24,
196     ↵ prob_empirical_24, 'b-', '#000080', '#0000FF', 'y+', 0)
197 line_plotter(o_32, o_counts_32, bin_sequence_24_32, x_sequence_24_32,
198     ↵ count_seeds, lineages_sampled_32, prob_empirical_32, 'b-',
199     ↵ '#000080', '#0000FF', 'y+', 0)
200 line_plotter(o_48, o_counts_48, bin_sequence_48, x_sequence_48,
201     ↵ count_seeds, lineages_sampled_48, prob_empirical_48, 'b-',
202     ↵ '#000080', '#0000FF', 'y+', 0)
203 line_plotter(o_PP, o_events_PP[:,0], bin_sequence_events,
204     ↵ x_sequence_events, event_seeds,
205     ↵ lineages_sampled_events,prob_empirical_PP,'b-', '#000080',
206     ↵ '#0000FF', 'y+', 1)

```

```

198     line_plotter(o_PD, o_events_PD[:,0], bin_sequence_events,
199                   ← x_sequence_events, event_seeds,
200                   ← lineages_sampled_events, prob_empirical_PD, 'b-', '#000080',
201                   ← '#0000FF', 'y+', 1)
202     line_plotter(o_DD, o_events_DD[:,0], bin_sequence_events,
203                   ← x_sequence_events, event_seeds,
204                   ← lineages_sampled_events, prob_empirical_DD, 'b-', '#000080',
205                   ← '#0000FF', 'y+', 1)
206     line_plotter(o_wan_noshad, o_counts_wan_no_shadow, bin_sequence_wan,
207                   ← x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
208                   ← 'b-', '#000080', '#0000FF', 'y+', 2)
209     o_ath5size.bar([0,1,2,3], [(wt_mean_clone_size/wt_mean_clone_size),
210                   ← (o_wt_mean_clone_size/wt_mean_clone_size),
211                   ← (ath5_mean_clone_size/wt_mean_clone_size),
212                   ← (o_ath5_mean_clone_size/wt_mean_clone_size)], color=['#000000', '#000080', '#0000FF'],
213                   ← 'WT', 'OSM WT', 'EO Ath5mo', 'OSM Ath5mo'])
214     o_ratio.bar([0,1,2,3], [wt_even_odd_ratio, o_wt_even_odd_ratio,
215                   ← ath5_even_odd_ratio,
216                   ← o_ath5_even_odd_ratio], color=['#000000', '#000080', '#000000', '#000080'], edgecolor='black',
217                   ← 'WT', 'OFit WT', 'EO Ath5mo', 'OFit Ath5mo'])
218     plt.setp(o_ath5size.get_xticklabels(), rotation=25, ha='right')
219     plt.setp(o_ratio.get_xticklabels(), rotation=25, ha='right')

220
221     s_objects = line_plotter(s_24, s_counts_24, bin_sequence_24_32,
222                   ← x_sequence_24_32, count_seeds, lineages_sampled_24,
223                   ← prob_empirical_24, 'm-', '#800080', '#FF00FF', 'k+', 0)
224     line_plotter(s_32, s_counts_32, bin_sequence_24_32, x_sequence_24_32,
225                   ← count_seeds, lineages_sampled_32, prob_empirical_32, 'm-',
226                   ← '#800080', '#FF00FF', 'k+', 0)
227     line_plotter(s_48, s_counts_48, bin_sequence_48, x_sequence_48,
228                   ← count_seeds, lineages_sampled_48, prob_empirical_48, 'm-',
229                   ← '#800080', '#FF00FF', 'k+', 0)
230     line_plotter(s_PP, s_events_PP[:,0], bin_sequence_events,
231                   ← x_sequence_events, event_seeds,
232                   ← lineages_sampled_events, prob_empirical_PP, 'm-', '#800080',
233                   ← '#FF00FF', 'k+', 1)
234     line_plotter(s_PD, s_events_PD[:,0], bin_sequence_events,
235                   ← x_sequence_events, event_seeds,
236                   ← lineages_sampled_events, prob_empirical_PD, 'm-', '#800080',
237                   ← '#FF00FF', 'k+', 1)

```

```

211     line_plotter(s_DD, s_events_DD[:,0], bin_sequence_events,
212                   ← x_sequence_events, event_seeds,
213                   ← lineages_sampled_events, prob_empirical_DD, 'm-', '#800080',
214                   ← '#FF00FF', 'k+', 1)
215     line_plotter(s_wan_noshad, s_counts_wan_no_shadow, bin_sequence_wan,
216                   ← x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
217                   ← 'm-', '#800080', '#FF00FF', 'k+', 2)
218     s_ath5size.bar([0,1,2,3], [(wt_mean_clone_size/wt_mean_clone_size),
219                               ← (s_wt_mean_clone_size/wt_mean_clone_size),(ath5_mean_clone_size/wt_mean_clone_
220                               ← '#000000', '#800080'],edgecolor='#000000',tick_label=['EO WT','SM
221                               ← WT','EO Ath5mo','SM Ath5mo'])
222     s_ratio.bar([0,1,2,3], [wt_even_odd_ratio, s_wt_even_odd_ratio,
223                               ← ath5_even_odd_ratio,
224                               ← s_ath5_even_odd_ratio],color=['#000000','#000080','#000000','#000080'],edgeco
225     plt.setp(s_ath5size.get_xticklabels(), rotation=25, ha='right')
226     plt.setp(s_ratio.get_xticklabels(), rotation=25, ha='right')

227
228
229     d_objects = line_plotter(d_24, d_counts_24, bin_sequence_24_32,
230                               ← x_sequence_24_32, count_seeds, lineages_sampled_24,
231                               ← prob_empirical_24, 'g-', '#008000', '#00FF00', 'k+', 0)
232     line_plotter(d_32, d_counts_32, bin_sequence_24_32, x_sequence_24_32,
233                               ← count_seeds, lineages_sampled_32, prob_empirical_32, 'g-',
234                               ← '#008000', '#00FF00', 'k+', 0)
235     line_plotter(d_48, d_counts_48, bin_sequence_48, x_sequence_48,
236                               ← count_seeds, lineages_sampled_48, prob_empirical_48, 'g-',
237                               ← '#008000', '#00FF00', 'k+', 0)
238     line_plotter(d_PP, d_events_PP[:,0], bin_sequence_events,
239                   ← x_sequence_events, event_seeds,
240                   ← lineages_sampled_events,prob_empirical_PP,'g-','#008000',
241                   ← '#00FF00', 'k+', 1)
242     line_plotter(d_PD, d_events_PD[:,0], bin_sequence_events,
243                   ← x_sequence_events, event_seeds,
244                   ← lineages_sampled_events,prob_empirical_PD,'g-','#008000',
245                   ← '#00FF00', 'k+', 1)
246     line_plotter(d_DD, d_events_DD[:,0], bin_sequence_events,
247                   ← x_sequence_events, event_seeds,
248                   ← lineages_sampled_events,prob_empirical_DD,'g-','#008000',
249                   ← '#00FF00', 'k+', 1)
250     line_plotter(d_wan_noshad, d_counts_wan_no_shadow, bin_sequence_wan,
251                   ← x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
252                   ← 'g-','#008000', '#00FF00', 'k+', 2)

```

```

226     d_ath5size.bar([0,1,2,3], [(wt_mean_clone_size/wt_mean_clone_size),
227                         ↳ (d_wt_mean_clone_size/wt_mean_clone_size),(ath5_mean_clone_size/wt_mean_clone_
228                         ↳ WT','DM WT', 'EO Ath5mo', 'DM Ath5mo')])
227 d_ratio.bar([0,1,2,3], [wt_even_odd_ratio, d_wt_even_odd_ratio,
228                         ↳ ath5_even_odd_ratio,
229                         ↳ d_ath5_even_odd_ratio],color=['#000000', '#008000', '#000000', '#008000'],edgeco
228                         ↳ WT', 'DM WT', 'EO Ath5mo', 'DM Ath5mo'])
229 plt.setp(d_ath5size.get_xticklabels(), rotation=25, ha='right')
230 plt.setp(d_ratio.get_xticklabels(), rotation=25, ha='right')

231
232 sd_objects_s = line_plotter(sd_24, s_counts_24, bin_sequence_24_32,
233                             ↳ x_sequence_24_32, count_seeds, lineages_sampled_24,
234                             ↳ prob_empirical_24, 'm-', '#800080', '#FF00FF', 'k+', 0)
233 sd_objects_d = line_plotter(sd_24, d_counts_24, bin_sequence_24_32,
234                             ↳ x_sequence_24_32, count_seeds, lineages_sampled_24,
235                             ↳ prob_empirical_24, 'g-', '#008000', '#00FF00', 'k+', 0)
234 line_plotter(sd_32, s_counts_32, bin_sequence_24_32,
235                             ↳ x_sequence_24_32, count_seeds, lineages_sampled_32,
236                             ↳ prob_empirical_32, 'm-', '#800080', '#FF00FF', 'k+', 0)
235 line_plotter(sd_32, d_counts_32, bin_sequence_24_32,
236                             ↳ x_sequence_24_32, count_seeds, lineages_sampled_32,
237                             ↳ prob_empirical_32, 'g-', '#008000', '#00FF00', 'k+', 0)
236 line_plotter(sd_48, s_counts_48, bin_sequence_48, x_sequence_48,
237                             ↳ count_seeds, lineages_sampled_48, prob_empirical_48, 'm-',
238                             ↳ '#800080', '#FF00FF', 'k+', 0)
237 line_plotter(sd_48, d_counts_48, bin_sequence_48, x_sequence_48,
238                             ↳ count_seeds, lineages_sampled_48, prob_empirical_48, 'g-',
239                             ↳ '#008000', '#00FF00', 'k+', 0)
238 line_plotter(sd_PP, s_events_PP[:,0], bin_sequence_events,
239                             ↳ x_sequence_events, event_seeds,
240                             ↳ lineages_sampled_events,prob_empirical_PP,'m-', '#800080',
241                             ↳ '#FF00FF', 'k+', 1)
239 line_plotter(sd_PP, d_events_PP[:,0], bin_sequence_events,
240                             ↳ x_sequence_events, event_seeds,
241                             ↳ lineages_sampled_events,prob_empirical_PP,'g-', '#008000',
242                             ↳ '#00FF00', 'k+', 1)
240 line_plotter(sd_PD, s_events_PD[:,0], bin_sequence_events,
241                             ↳ x_sequence_events, event_seeds,
242                             ↳ lineages_sampled_events,prob_empirical_PD,'m-', '#800080',
243                             ↳ '#FF00FF', 'k+', 1)

```

```

241     line_plotter(sd_PD, d_events_PD[:,0], bin_sequence_events,
242                   ↵ x_sequence_events, event_seeds,
243                   ↵ lineages_sampled_events,prob_empirical_PD,'g-', '#008000',
244                   ↵ '#00FF00', 'k+', 1)
245     line_plotter(sd_DD, s_events_DD[:,0], bin_sequence_events,
246                   ↵ x_sequence_events, event_seeds,
247                   ↵ lineages_sampled_events,prob_empirical_DD,'m-', '#800080',
248                   ↵ '#FF00FF', 'k+', 1)
249     line_plotter(sd_DD, d_events_DD[:,0], bin_sequence_events,
250                   ↵ x_sequence_events, event_seeds,
251                   ↵ lineages_sampled_events,prob_empirical_DD,'g-', '#008000',
252                   ↵ '#00FF00', 'k+', 1)
253     line_plotter(sd_wan_noshad, s_counts_wan_no_shadow, bin_sequence_wan,
254                   ↵ x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
255                   ↵ 'm-', '#800080', '#FF00FF', 'k+', 2)
256     line_plotter(sd_wan_noshad, d_counts_wan_no_shadow, bin_sequence_wan,
257                   ↵ x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
258                   ↵ 'g-', '#008000', '#00FF00', 'k+', 2)
259     sd_ath5size.bar([0,1,2,3,4,5], [(wt_mean_clone_size),
260                   ↵ (s_wt_mean_clone_size),(d_wt_mean_clone_size),(ath5_mean_clone_size),(s_ath5_
261                   ↵ WT','SM WT','DM WT', 'EO Ath5mo', 'SM Ath5Mo', 'DM Ath5mo')])
262     sd_ratio.bar([0,1,2,3,4,5], [wt_even_odd_ratio, s_wt_even_odd_ratio,
263                   ↵ d_wt_even_odd_ratio, ath5_even_odd_ratio, s_ath5_even_odd_ratio,
264                   ↵ d_ath5_even_odd_ratio],color=['#000000','#800080','#008000','#000000','#800080
265                   ↵ WT','SM WT','DM WT', 'EO Ath5mo', 'SM Ath5Mo', 'DM Ath5mo'])
266     plt.setp(sd_ath5size.get_xticklabels(), rotation=25, ha='right')
267     plt.setp(sd_ratio.get_xticklabels(), rotation=25, ha='right')

268
269 #calculate AICs, write table to file
270 o_fit_AIC = calculate_fit_AIC(o_fit_output_list, empirical_fit_list,
271                   ↵ he_model_params)
272 o_test_AIC = calculate_test_AIC(o_test_output_list,
273                   ↵ empirical_fit_list, he_model_params)

274
275 s_fit_AIC = calculate_fit_AIC(s_fit_output_list, empirical_fit_list,
276                   ↵ he_model_params)
277 s_test_AIC = calculate_test_AIC(s_test_output_list,
278                   ↵ empirical_test_list, he_model_params)

279
280 d_fit_AIC = calculate_fit_AIC(d_fit_output_list, empirical_fit_list,
281                   ↵ det_model_params)
282 d_test_AIC = calculate_test_AIC(d_test_output_list,
283                   ↵ empirical_test_list, det_model_params)

```

```
260
261     AIC_table('AIC_table.tex', o_fit_AIC, o_test_AIC, s_fit_AIC,
262     ↪   s_test_AIC, d_fit_AIC, d_test_AIC)
263
264     #plot annotations
265     plot_annotater(s_fig,s_objects,'Stochastic Model')
266     plot_annotater(o_fig,o_objects,'Stochastic Model (original fit)')
267     plot_annotater(d_fig,d_objects, 'Deterministic Model')
268     plot_annotater(sd_fig,sd_objects_s, 'Stochastic Model', sd_objects_d,
269     ↪   'Deterministic Model')
270
271     #export as .tiff, .png
272     o_fig.savefig('o_fig.png',dpi=600)
273     png_memory = BytesIO()
274     o_fig.savefig(png_memory, format='png', dpi=600)
275     PILpng = Image.open(png_memory)
276     PILpng.save('o_fig.tiff')
277     png_memory.close()
278
279     s_fig.savefig('s_fig.png',dpi=600)
280     png_memory = BytesIO()
281     s_fig.savefig(png_memory, format='png', dpi=600)
282     PILpng = Image.open(png_memory)
283     PILpng.save('s_fig.tiff')
284     png_memory.close()
285
286     d_fig.savefig('d_fig.png',dpi=600)
287     png_memory = BytesIO()
288     d_fig.savefig(png_memory, format='png', dpi=600)
289     PILpng = Image.open(png_memory)
290     PILpng.save('d_fig.tiff')
291     png_memory.close()
292
293     sd_fig.savefig('sd_fig.png',dpi=600)
294     png_memory = BytesIO()
295     sd_fig.savefig(png_memory, format='png', dpi=600)
296     PILpng = Image.open(png_memory)
297     PILpng.save('sd_fig.tiff')
298     png_memory.close()
299
300
301     plt.show()
```

```
301 def calculate_fit_AIC(output_list, empirical_list, number_params):
302     #function calculates AIC for He et al counts & event probability
303     #→ density (NOT per-lineage event probability)
304     #this numpy array holds the individual RSS vals for timepoints +
305     #→ events
306     rss = np.zeros(6)
307     number_comparisons = 0
308
309
310
311     for i in range(0,3):
312         output_counts = output_list[i]
313         empirical_prob = empirical_list[i]
314
315         output_histo,bin_edges = np.histogram(output_counts,
316             → np.arange(1,len(empirical_prob)+2), density = False)
317         output_prob = np.array(output_histo/count_seeds)
318
319         residual = np.array(output_prob- empirical_prob)
320         number_comparisons += len(residual)
321         rss[i] = np.sum(np.square(residual))
322
323
324     for i in range (0,3):
325         output_events = output_list[i+3]
326         empirical_prob = empirical_list[i+3]
327
328         histo_events,bin_edges = np.histogram(output_events,
329             → bin_sequence_events, density=True)
330         output_prob = np.array(histo_events/event_seeds)
331
332         residual = np.array(output_prob - empirical_prob)
333         number_comparisons += len(residual)
334         rss[i+3] = np.sum(np.square(residual))
335
336
337     AIC = 2 * number_params + number_comparisons * np.log(np.sum(rss))
338
339
340     return AIC
341
342
343 def calculate_test_AIC(output_list, empirical_list, number_params):
344     #function calculates AIC for He mutant data & Wan counts
345     rss=np.zeros(3)
346     number_comparisons = 0
347
348     #Wan data
349     for i in range (0,1):
```

```

340     output_counts = output_list[i]
341     empirical_prob = empirical_list[i]
342
343     output_histo, bin_edges = np.histogram(output_counts,
344         ↳ np.arange(2, len(empirical_prob)+3), density = False)
345     output_prob = np.array(output_histo/count_seeds)
346
347     residual = np.array(output_prob - empirical_prob)
348     number_comparisons += len(residual)
349     rss[i] = np.sum(np.square(residual))
350
351     #He data
352     for i in range(1,3):
353         output_measure = output_list[i]
354         empirical_measure = empirical_list[i]
355
356         residual = np.array(output_measure - empirical_measure)
357         number_comparisons += 1
358         rss[i] = np.sum(np.square(residual))
359
360     AIC = 2 * number_params + number_comparisons * np.log(np.sum(rss))
361
362     return AIC
363
364 #line plotter plots average counts or event event +- 2 standard
365 #    deviations (He et al's "95% probability interval")
366 # "mode" 1 gives correct hourly output for 5-hour event bins. mode 2
367 #    begins Wan plots at clone size 2, reflecting unavailable data
368 def line_plotter(subplot, data, bin_sequence, x_sequence, seeds, samples,
369     ↳ empirical_prob, line_colour_string, fill_edge_colour_string,
370     ↳ fill_face_colour_string, empirical_colour_string, mode):
371
372     legend_objects = []
373
374     prob_histo, bin_edges = np.histogram(data, bin_sequence,
375         ↳ density=True)
376
377     #estimate of 95% CI for empirical observations of model output from
378     #    repeated sampling of the data w/ the appropriate number of
379     #    empirically observed lineages
380     interval = sampler(data,samples,bin_sequence)
381     legend_objects.append(subplot.plot(x_sequence,prob_histo,
382         ↳ line_colour_string))

```

```

374     subplot.fill_between(x_sequence, (prob_histo - interval), (prob_histo
375     ↵ + interval), alpha=0.1, edgecolor=fill_edge_colour_string,
376     ↵ facecolor=fill_face_colour_string)
377
378     if mode == 0:
379
380         ↵ legend_objects.append(subplot.plot(np.arange(1,len(empirical_prob)+1,1),e
381
382     if mode == 1:
383
384         ↵ legend_objects.append(subplot.plot(np.arange(30,80,5),empirical_prob,
385         ↵ empirical_colour_string))
386
387     if mode == 2:
388
389         ↵ legend_objects.append(subplot.plot(np.arange(2,len(empirical_prob)+2,1),
390         ↵ empirical_prob, empirical_colour_string))
391
392     return legend_objects
393
394 def sampler(data,samples,bin_sequence):
395
396     if len(data) == 0: #catches edge case w/ no entries for a mitotic
397         ↵ mode
398         data=[0]
399
400     base_sample=np.zeros((error_samples,len(bin_sequence)-1))
401
402     for i in range(0,error_samples):
403         new_data_sample = np.random.choice(data,samples)
404         new_histo_prob, bin_edges = np.histogram(new_data_sample,
405             ↵ bin_sequence, density=True)
406         base_sample[i,:] = new_histo_prob
407
408     sample_95CI = np.array(2 * (np.std(base_sample,0)))
409
410     return sample_95CI
411
412 def plot_annotater(fig_to_ann, legend_objects_1, legend_objects_1_name,
413     ↵ legend_objects_2=None, legend_objects_2_name=None):
414     #subplot labels
415     labels = ['A','B','C','D','E','F','G','H','I','']

```

```
406
407     for i, ax in enumerate(fig_to_ann.axes):
408         ax.text(.01,.95, labels[i], transform=ax.transAxes,
409                 fontweight='bold', va='top')
410
411         time_rectangle_x = .70
412         time_rectangle_y = .65
413         time_rectangle_width = .2
414         time_rectangle_height = .03
415         time_caret_half_width = .02
416         time_caret_height = np.sqrt(3)*(time_caret_half_width*2)
417
418     if i < 3:
419         #add the time bar and label 72 hr caret on appropriate plots
420
421         ax.add_patch(plt.Rectangle((time_rectangle_x,time_rectangle_y),time_r
422
423         if i == 0:
424             #add 24hr caret
425
426             ax.add_patch(plt.Polygon(((time_rectangle_x,time_rectangle_y+time_rec
427
428         if i == 1:
429             #add 32hr caret
430
431             ax.add_patch(plt.Polygon(((time_rectangle_x+(8/48)*time_rectangle_wid
```

```
431     ↪ ax.text(time_rectangle_x+(8/48)*time_rectangle_width,time_rectangle_y
        ↪ transform=ax.transAxes,
        ↪ fontsize='8',va='bottom',ha='center')
432
433 if i == 2:
434
435     ↪ ax.add_patch(plt.Polygon(((time_rectangle_x+(24/48)*time_rectangle_width,
        ↪ time_rectangle_y),),
        ↪ fill=False, transform=ax.transAxes))
436
437 parent_circle_coord = (.85,.8)
438 child_1_coord = (.8,.6)
439 child_2_coord = (.9,.6)
440 ellipse_width = .03
441 ellipse_height = 2.5*ellipse_width
442
443 if i == 3:
444
445     ↪ ax.add_patch(plt.Polygon((parent_circle_coord),child_1_coord),color='k')
446     ↪ ax.add_patch(plt.Polygon((parent_circle_coord),child_2_coord),color='k')
447
448     ↪ ax.add_patch(patches.Ellipse(parent_circle_coord,ellipse_width,ellipse_height,
        ↪ color='k', transform=ax.transAxes))
449     ↪ ax.add_patch(patches.Ellipse(child_1_coord,ellipse_width,ellipse_height,
        ↪ color='k', transform=ax.transAxes))
450     ↪ ax.add_patch(patches.Ellipse(child_2_coord,ellipse_width,ellipse_height,
        ↪ color='k', transform=ax.transAxes))
451
452 if i == 4:
453     ↪ ax.add_patch(plt.Polygon((parent_circle_coord),child_1_coord),color='k')
        ↪ ax.add_patch(plt.Polygon((parent_circle_coord),child_2_coord),color='k')
```

```
454     ↵ ax.add_patch(patches.Ellipse(parent_circle_coord,ellipse_width,ellipse_
455     ↵ color='k', transform=ax.transAxes))
456
457     ↵ ax.add_patch(patches.Ellipse(child_1_coord,ellipse_width,ellipse_heig
458
459     ↵ ax.add_patch(patches.Ellipse(child_2_coord,ellipse_width,ellipse_heig
460     ↵ edgecolor='k', facecolor='w', transform=ax.transAxes))
461     ↵ ax.text(.91, .7, 'PD', transform=ax.transAxes, va='center',
462     ↵ ha='left')
463
464
465     ↵ ax.add_patch(plt.Polygon(((parent_circle_coord),child_1_coord),color=
466     ↵ ax.add_patch(plt.Polygon(((parent_circle_coord),child_2_coord),color=
467
468
469     ↵ if i == 5:
470         ↵ ax.add_patch(patches.Ellipse(parent_circle_coord,ellipse_width,ellipse_
471         ↵ color='k', transform=ax.transAxes))
472
473         ↵ ax.add_patch(patches.Ellipse(child_1_coord,ellipse_width,ellipse_heig
474         ↵ edgecolor='k', facecolor='w', transform=ax.transAxes))
475
476         ↵ ax.add_patch(patches.Ellipse(child_2_coord,ellipse_width,ellipse_heig
477         ↵ edgecolor='k', facecolor='w', transform=ax.transAxes))
478         ↵ ax.text(.91, .7, 'DD', transform=ax.transAxes, va='center',
479         ↵ ha='left')
480
481
482     ↵ if i == 6:
483         ↵ ax.text(.80,.85,'CMZ', transform=ax.transAxes,
484         ↵ fontweight='bold', va='top')
485
486     ↵ if legend_objects_2 == None:
487
488         ↵ fig_to_ann.legend((legend_objects_1[1][0],legend_objects_1[0][0]),('Obser
489         ↵ bbox_to_anchor=(.59,.10,.1,.1), loc='upper left')
490
491 else:
```

```

475     ↵ fig_to_ann.legend((legend_objects_1[1][0],legend_objects_1[0][0],legend_o
        ↵ legend_objects_2_name), bbox_to_anchor=(.59,.1,.1,.1),
        ↵ loc='upper left')

476
477 #data group box annotations
478 fig_to_ann.patches.extend([plt.Rectangle((0.005,0.41),0.99,0.58,
479                               fill=False, color='k', linestyle='--',
480                               transform=fig_to_ann.transFigure,
481                               ↵ figure=fig_to_ann)])
481 fig_to_ann.patches.extend([plt.Polygon(((0.005, .4),(.995,.4),
482                               ↵ (.995,.21), (.5,.21), (.5, .01), (.005, .01)),
483                               fill=False, color='k', linestyle=':',
484                               transform=fig_to_ann.transFigure,
485                               ↵ figure=fig_to_ann)])
486
487 fig_to_ann.text(.605, .075, 'Training Data', fontsize=12, transform =
488     ↵ fig_to_ann.transFigure, figure=fig_to_ann)
489 fig_to_ann.patches.extend([plt.Rectangle((0.6,0.07),0.143,0.029,
490                               fill=False, color='k', linestyle='--',
491                               transform=fig_to_ann.transFigure,
492                               ↵ figure=fig_to_ann)])
493
494 fig_to_ann.text(.605, .025, 'Test Data', fontsize=12, transform =
495     ↵ fig_to_ann.transFigure, figure=fig_to_ann)
496 fig_to_ann.patches.extend([plt.Rectangle((0.6,0.02),0.111,0.029,
497                               fill=False, color='k', linestyle=':',
498                               transform=fig_to_ann.transFigure,
499                               ↵ figure=fig_to_ann)])
500
501 fig_to_ann.subplots_adjust(wspace=0.4, hspace=0.3)
502 fig_to_ann.tight_layout()

503 def AIC_table (filename, o_training_AIC, o_test_AIC, s_training_AIC,
504     ↵ s_test_AIC, d_training_AIC, d_test_AIC):
505     #writes LaTeX tabular snippet with AIC values
506     table_file = open(filename, 'w')
507     table_file.write(r'\begin{tabular}{l|l|l}'+r'\n')
508     table_file.write(r'\hline'+r'\n')
509     table_file.write(r'{\bf Model} & {\bf Training AIC} & {\bf Test AIC}
510     ↵ \\\thickhline'+r'\n')

```

```

505     table_file.write(r'Stochastic (He fit) &
506         +' + f'{o_training_AIC:.2f}' + r'$ & $' + f'{o_test_AIC:.2f}' + r'$\\'
507         + '\hline' + '\n')
508     table_file.write(r'Stochastic (SPSA fit) &
509         +' + f'{s_training_AIC:.2f}' + r'$ & $' + f'{s_test_AIC:.2f}' + r'$\\'
510         + '\hline' + '\n')
511     table_file.write(r'Deterministic (SPSA fit) &
512         +' + f'{d_training_AIC:.2f}' + r'$ & $' + f'{d_test_AIC:.2f}' + r'$\\'
513         + '\hline' + '\n')
514     table_file.write(r'\end{tabular}' + '\n')

509
510 if __name__ == "__main__":
511     main()

```

---

### 17.1.16 /python\_fixtures/figure\_plots/Kolmogorov\_plot.py

```

1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 #AIC & Plotting utility params
6 event_seeds = 1000
7 error_samples = 5000 #number of samples to draw when estimating
    → plausibility interval for simulations
8
9 def main():
10     #LOAD & PARSE KOLMOGOROV COMPLEXITY FILES
11     gomes_kol =
12         → np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/Kolmogorov_gomes.txt',
13                     → skiprows=1, usecols=1)
14     he_kol =
15         → np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/Kolmogorov_he.txt',
16                     → skiprows=1, usecols=1)
17     he_refit_kol =
18         → np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/Kolmogorov_he_refit.txt',
19                     → skiprows=1, usecols=1)
20     deterministic_kol =
21         → np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/Kolmogorov_deterministic.txt',
22                     → skiprows=1, usecols=1)
23     boije_kol =
24         → np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/Kolmogorov_boije.txt',
25                     → skiprows=1, usecols=1)

```

```

16     eo_kol =
17         ↳ np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/Kol'
18             ↳ skiprows=1, usecols=1)
19
20     data_sequence =
21         ↳ [gomes_kol,he_kol,he_refit_kol,deterministic_kol,boije_kol,eo_kol]
22
23     plt.violinplot(data_sequence, showextrema=True)
24
25     plt.ylabel ('Estimated Kolmogorov Complexity')
26
27     plt.show()
28
29 if __name__ == "__main__":
30     main()

```

---

### 17.1.17 /python\_fixtures/figure\_plots/Mitotic\_rate\_plot.py

```

1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 from PIL import Image
6 from io import BytesIO
7
8 #PLoS formatting stuff
9 plt.rcParams['font.size'] = 12
10 plt.rcParams['font.family'] = 'sans-serif'
11 plt.rcParams['font.sans-serif'] = ['Arial']
12
13 #AIC & Plotting utility params
14 event_seeds = 1000
15 error_samples = 5000 #number of samples to draw when estimating
16             ↳ plausibility interval for simulations
17
18 bin_sequence_events = np.arange(30,85,5)
19 x_sequence_events = np.arange(30,80,5)
20 #####
21 # HE ET AL EMPIRICAL RESULTS
22 #####
23

```

```

24 rate_bin_sequence = np.arange(30,85,5)
25 rate_x_sequence = np.arange(30,80,5)
26 rate_trim_value = 10
27
28 #no. of lineages EO per induction timepoint/event group
29 lineages_sampled_events = 60
30
31 #Mitotic mode rate probability arrays
32 raw_events =
    ↳ np.loadtxt('/home/main/git/chaste/projects/ISP/empirical_data/empirical_lineages.txt')
    ↳ skiprows=1, usecols=(3,5,8))
33 EO_events = raw_events[np.where(raw_events[:,2]==1)] #exclude any mitosis
    ↳ whose time was too early for recording
34
35 event_counts, bin_edges =
    ↳ np.histogram(EO_events,bin_sequence_events,density=False)
36 #hourly values
37 empirical_events_per_lineage =
    ↳ np.array(event_counts/(lineages_sampled_events*5))
38
39 def main():
40     #LOAD & PARSE HE MODEL OUTPUT FILES
41     #original fit stochastic mitotic mode files
42     o_events =
        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM000080.txt")
        ↳ skiprows=1, usecols=(0))
43
44     #refit stochastic mitotic mode files
45     s_events =
        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM0000FF.txt")
        ↳ skiprows=1, usecols=(0))
46
47     #deterministic mitotic mode files
48     d_events =
        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM000080.txt")
        ↳ skiprows=1, usecols=(0))
49
50     original = line_plotter(o_events,bin_sequence_events,
        ↳ x_sequence_events, event_seeds, lineages_sampled_events, 'b--',
        ↳ '#000080', '#0000FF')
51     refit = line_plotter(s_events,bin_sequence_events, x_sequence_events,
        ↳ event_seeds, lineages_sampled_events, 'm-', '#800080', '#FF00FF')

```

```

52     deterministic = line_plotter(d_events,bin_sequence_events,
53         ↵ x_sequence_events, event_seeds, lineages_sampled_events, 'g-',
54         ↵ '#008000', '#00FF00')
55
56     empirical = plt.plot(np.arange(30,80,5),empirical_events_per_lineage,
57         ↵ 'k+')
58
59     plt.legend((original[0], refit[0], deterministic[0], empirical[0]),
60         ↵ ('SM (He fit)', 'SM (SPSA fit)', 'DM', 'Observed'))
61
62     plt.xlabel ('Age (hpf)')
63     plt.ylabel ('Probability of mitotic event per lineage per hour')
64
65
66
67     plt.savefig("per_lineage_mitotic_rate.png")
68     png_memory = BytesIO()
69     plt.savefig(png_memory, format='png', dpi=600)
70     PILpng = Image.open(png_memory)
71     PILpng.save('per_lineage_mitotic_rate.tiff')
72     png_memory.close()
73
74     plt.show()
75
76
77
78
79
80
81
82

```

71 #line plotter plots average counts or event event +- 2 standard  
 ↵ deviations (He et al's "95% probability interval")  
 72 # "mode" 1 gives correct hourly output for 5-hour event bins. mode 2  
 ↵ begins Wan plots at clone size 2, reflecting unavailable data  
 73 def line\_plotter(data, bin\_sequence, x\_sequence, seeds, samples,  
 ↵ line\_colour\_string, fill\_edge\_colour\_string,  
 ↵ fill\_face\_colour\_string):  
 74  
 75 histo, bin\_edges = np.histogram(data, bin\_sequence, density=False)  
 76 prob\_histo = np.array(histo / (seeds \* 5))  
 77  
 78 #estimate of 95% CI for empirical observations of model output from  
 ↵ repeated sampling of the data w/ the appropriate number of  
 ↵ empirically observed lineages  
 79 interval = sampler(data,samples,bin\_sequence)  
 80 line = plt.plot(x\_sequence,prob\_histo, line\_colour\_string)  
 81 plt.fill\_between(x\_sequence, (prob\_histo - interval), (prob\_histo +  
 ↵ interval), alpha=0.1, edgecolor=fill\_edge\_colour\_string,  
 ↵ facecolor=fill\_face\_colour\_string)

```

83     return line
84
85 def sampler(data,samples,bin_sequence):
86
87     if len(data) == 0: #catches edge case w/ no entries for a mitotic
88         → mode
89         data=[0]
90
91     base_sample=np.zeros((error_samples,len(bin_sequence)-1))
92
93     for i in range(0,error_samples):
94         new_data_sample = np.random.choice(data,samples)
95         new_histo, bin_edges = np.histogram(new_data_sample,
96             → bin_sequence, density=False)
97         new_histo_prob = np.array(new_histo/samples)
98         base_sample[i,:] = new_histo_prob
99
100    sample_95CI = np.array(2 * (np.std(base_sample,0)))
101
102 if __name__ == "__main__":
103     main()

```

---

### 17.1.18 /python\_fixtures/figure\_plots/Wan\_output\_plot.py

---

```

1 import os
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import matplotlib.ticker as ticker
6
7 from PIL import Image
8 from io import BytesIO
9
10 #PLoS formatting stuff
11 plt.rcParams['font.size'] = 12
12 plt.rcParams['font.family'] = 'sans-serif'
13 plt.rcParams['font.sans-serif'] = ['Arial']
14
15 def main():

```

```

16     wan_output_dir =
17         '/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/WanOutput'
18     wan_results_list = []
19     for root,dirs,files in os.walk(wan_output_dir):
20         for name in files:
21             if name == 'celltypes.dat':
22                 results = np.loadtxt(os.path.join(root,name), usecols=2)
23                 if len(results) == 8569:
24                     wan_results_list.append(results)
25
26     wan_sim_results = np.array(wan_results_list)
27     wan_sim_mean = np.mean(wan_sim_results, axis=0)
28     wan_sim_95CI = 2*np.std(wan_sim_results, axis=0)
29     wan_sim_x_seq = np.arange(72,8641,1)/24
30
31     empirical_pop_mean = np.array([792.0, 768.4, 906.1, 1159.7, 1630.0,
32         ↵ 3157.9, 3480.2, 4105.1, 1003.0, 477.2, 438.8088611111])
33     empirical_pop_95CI = 2*np.array([160.1, 200.1, 244.5, 477.6, 444.3,
34         ↵ 1414.3, 472.1, 1169.7, 422.8, 367.5, 294.8])
35     empirical_x_seq =
36         ↵ np.array([72,120,192,288,408,552,720,1440,2160,4320,8640])/24
37
38     fig, ax = plt.subplots(1,1)
39
40     empirical = plt.plot(empirical_x_seq, empirical_pop_mean, 'g-')
41     plt.fill_between(empirical_x_seq, (empirical_pop_mean -
42         ↵ empirical_pop_95CI), (empirical_pop_mean + empirical_pop_95CI),
43         ↵ alpha=0.1, edgecolor='#008000', facecolor='#00FF00')
44
45     wan_simulator = plt.plot(wan_sim_x_seq, wan_sim_mean, 'm-')
46     plt.fill_between(wan_sim_x_seq, (wan_sim_mean - wan_sim_95CI),
47         ↵ (wan_sim_mean + wan_sim_95CI), alpha=0.1, edgecolor='#800080',
48         ↵ facecolor='#FF00FF')
49
50     plt.ylim(bottom=0)
51     plt.xlim(left=0)
52
53     plt.ylabel("CMZ population size")
54     plt.xlabel("Retina age (dpf)")
55     ax.xaxis.set_major_locator(ticker.MultipleLocator(40))
56     ax.legend((empirical[0], wan_simulator[0]), ('Observed CMZ
57         ↵ population', '"Wan-type" simulated CMZ population'))
```

```

50     png_memory = BytesIO()
51     fig.savefig(png_memory, format='png', dpi=600)
52     PILpng = Image.open(png_memory)
53     PILpng.save('wan_fig.tiff')
54     png_memory.close()
55
56     plt.show()
57
58 if __name__ == "__main__":
59     main()

```

---

### 17.1.19 /src/BoijeCellCycleModel.cpp

```

1 #include "BoijeCellCycleModel.hpp"
2
3 BoijeCellCycleModel::BoijeCellCycleModel() :
4     AbstractSimpleCellCycleModel(), mOutput(false),
5     ↪ mEventStartTime(), mSequenceSampler(false),
6     ↪ mSeqSamplerLabelSister(
7         false), mDebug(false), mTimeID(), mVarIDs(),
8         ↪ mDebugWriter(), mGeneration(0), mPhase2gen(3),
9         ↪ mPhase3gen(
10            5), mprobAtoh7(0.32), mprobPtf1a(0.30), mprobng(0.80),
11            ↪ mAtoh7Signal(false), mPtf1aSignal(false), mNgSignal(
12              false), mMitoticMode(0), mSeed(0), mp_PostMitoticType(),
13              ↪ mp_RGC_Type(), mp_AC_HC_Type(), mp_PR_BC_Type(),
14              ↪ mp_label_Type()
15 {
16 }
17
18 BoijeCellCycleModel::BoijeCellCycleModel(const BoijeCellCycleModel&
19     ↪ rModel) :
20     AbstractSimpleCellCycleModel(rModel), mOutput(rModel.mOutput),
21     ↪ mEventStartTime(rModel.mEventStartTime), mSequenceSampler(
22         rModel.mSequenceSampler),
23         ↪ mSeqSamplerLabelSister(rModel.mSeqSamplerLabelSister),
24         ↪ mDebug(rModel.mDebug), mTimeID(
25             rModel.mTimeID), mVarIDs(rModel.mVarIDs),
26             ↪ mDebugWriter(rModel.mDebugWriter), mGeneration(
27                 rModel.mGeneration), mPhase2gen(rModel.mPhase2gen),
28                 ↪ mPhase3gen(rModel.mPhase3gen), mprobAtoh7(
29

```

```

16             rModel.mprobAtoh7), mprobPtf1a(rModel.mprobPtf1a),
17             ↳ mprobng(rModel.mprobng), mAtoh7Signal(
18             rModel.mAtoh7Signal), mPtf1aSignal(rModel.mPtf1aSignal),
19             ↳ mNgSignal(rModel.mNgSignal), mMitoticMode(
20             rModel.mMitoticMode), mSeed(rModel.mSeed),
21             ↳ mp_PostMitoticType(rModel.mp_PostMitoticType),
22             ↳ mp_RGC_Type(
23             rModel.mp_RGC_Type), mp_AC_HC_Type(rModel.mp_AC_HC_Type),
24             ↳ mp_PR_BC_Type(rModel.mp_PR_BC_Type), mp_label_Type(
25             rModel.mp_label_Type)
26     {
27 }
28
29 AbstractCellCycleModel* BoijeCellCycleModel::CreateCellCycleModel()
30 {
31     return new BoijeCellCycleModel(*this);
32 }
33
34 void BoijeCellCycleModel::SetCellCycleDuration()
35 {
36     if
37         ↳ (mpCell->GetCellProliferativeType()->IsType<DifferentiatedCellProliferativeTy
38     {
39         mCellCycleDuration = DBL_MAX;
40     }
41     else
42     {
43         mCellCycleDuration = 1.0;
44     }
45 }
46
47 void BoijeCellCycleModel::ResetForDivision()
48 {
49     mGeneration++; //increment generation counter
50     //the first division is ascribed to generation "1"
51     RandomNumberGenerator* p_random_number_generator =
52         ↳ RandomNumberGenerator::Instance();
53
54     mMitoticMode = 0; //0=PP;1=PD;2=DD
55 }
```

```

52  ****
53  * TRANSCRIPTION FACTOR RANDOM VARIABLES & RULES
54  * 1st phase: only PP divisions (symmetric proliferative) permitted
55  * 2nd phase: all division modes permitted, all TF signals avail
56  * 3rd phase: only PP/DD modes permitted, only ng has nonzero signal
57  ****
58
59 //reset RVs and signals
60 double atoh7RV = 1, ptf1aRV = 1, ngRV = 1;
61 mAtoh7Signal = mPtf1aSignal = mNgSignal = false;
62
63
64 //PHASE & TF SIGNAL RULES
65 if (mGeneration > mPhase2gen && mGeneration ≤ mPhase3gen) //if the
   ↳ cell is in the 2nd model phase, all signals have nonzero
   ↳ probabilities at each division
66 {
67     //RVs take values evenly distributed across 0-1
68     atoh7RV = p_random_number_generator→ranf();
69     ptf1aRV = p_random_number_generator→ranf();
70     ngRV = p_random_number_generator→ranf();
71
72     if (atoh7RV < mprobAtoh7)
73     {
74         mAtoh7Signal = true;
75     }
76     if (ptf1aRV < mprobPtf1a)
77     {
78         mPtf1aSignal = true;
79     }
80     if (ngRV < mprobng)
81     {
82         mNgSignal = true;
83     }
84 }
85
86 if (mGeneration > mPhase3gen) //if the cell is in the 3rd model
   ↳ phase, only ng signal has a nonzero probability
87 {
88     ngRV = p_random_number_generator→ranf();
     //roll a probability die for the ng signal
89     if (ngRV < mprobng)
90     {

```

```

92         mNgSignal = true;
93     }
94 }
95
96 /*****
97 * MITOTIC MODE & SPECIFICATION RULES
98 * -(additional asymmetric postmitotic rules specified in
99 ← InitialiseDaughterCell();)
100 * *****/
101
102 if(mAtoh7Signal == true)
103 {
104     mMitoticMode = 1;
105 }
106
107 if (mPtf1aSignal == true && mAtoh7Signal == false) //Ptf1A alone
108     → gives a symmetrical postmitotic AC/HC division
109 {
110     mMitoticMode = 2;
111     mpCell→SetCellProliferativeType(mp_PostMitoticType);
112     mpCell→AddCellProperty(mp_AC_HC_Type);
113 }
114
115 if (mPtf1aSignal == false && mAtoh7Signal == false && mNgSignal ==
116     → true) //ng alone gives a symmetrical postmitotic PR/BC division
117 {
118     mMitoticMode = 2;
119     mpCell→SetCellProliferativeType(mp_PostMitoticType);
120     mpCell→AddCellProperty(mp_PR_BC_Type);
121
122 /*****
123 * Write mitotic event to file if appropriate
124 * mDebug: many files: detailed per-lineage info switch; intended for
125 ← TestHeInductionCountFixture
126     * mOutput: 1 file: time, seed, cellID, mitotic mode, intended for
127 ← TestHeMitoticModeRateFixture
128     * *****/
129
130 if (mDebug)
131 {
132     WriteDebugData(atoh7RV, ptf1aRV, ngRV);
133 }

```

```

130
131     if (mOutput)
132     {
133         WriteModeEventOutput();
134     }
135
136     AbstractSimpleCellCycleModel::ResetForDivision();
137
138     /*****  

139     * SEQUENCE SAMPLER  

140     *****/
141     //if the sequence sampler has been turned on, check for the label &
142     //→ write mitotic mode to log
143     //50% chance of each daughter cell from a mitosis inheriting the
144     //→ label
145     if (mSequenceSampler)
146     {
147         if (mpCell→HasCellProperty<CellLabel>())
148         {
149             (*LogFile::Instance()) << mMitoticMode;
150             double labelRV = p_random_number_generator→ranf();
151             if (labelRV ≤ .5)
152             {
153                 mSeqSamplerLabelSister = true;
154                 mpCell→RemoveCellProperty<CellLabel>();
155             }
156             else
157             {
158                 mSeqSamplerLabelSister = false;
159             }
160         }
161         //prevents lost-label cells from labelling their progeny
162         mSeqSamplerLabelSister = false;
163     }
164 }
165 }
166
167 void BoijeCellCycleModel::InitialiseDaughterCell()
168 {
169     //Asymmetric specification rules
170

```

```
171     if (mAtoh7Signal == true)
172     {
173         if (mPtf1aSignal == true)
174         {
175             mpCell→SetCellProliferativeType(mp_PostMitoticType);
176             mpCell→AddCellProperty(mp_AC_HC_Type);
177         }
178         else
179         {
180             mpCell→SetCellProliferativeType(mp_PostMitoticType);
181             mpCell→AddCellProperty(mp_RGC_Type);
182         }
183     }
184
185     /*****
186     * SEQUENCE SAMPLER
187     *****/
188     if (mSequenceSampler)
189     {
190         if (mSeqSamplerLabelSister)
191         {
192             mpCell→AddCellProperty(mp_label_Type);
193             mSeqSamplerLabelSister = false;
194         }
195         else
196         {
197             mpCell→RemoveCellProperty<CellLabel>();
198         }
199     }
200 }
201
202 void BoijeCellCycleModel::SetGeneration(unsigned generation)
203 {
204     mGeneration = generation;
205 }
206
207 unsigned BoijeCellCycleModel::GetGeneration() const
208 {
209     return mGeneration;
210 }
211
```

```
212 void
213     → BoijeCellCycleModel::SetPostMitoticType(boost::shared_ptr<AbstractCellProperty>
214         → p_PostMitoticType)
215 {
216     mp_PostMitoticType = p_PostMitoticType;
217
218     → boost::shared_ptr<AbstractCellProperty>
219     → p_AC_HC_Type,
220
221     → boost::shared_ptr<AbstractCellProperty>
222     → p_PR_BC_Type)
223 {
224     mp_RGC_Type = p_RGC_Type;
225     mp_AC_HC_Type = p_AC_HC_Type;
226     mp_PR_BC_Type = p_PR_BC_Type;
227 }
228
229
230
231
232
233
234
235 void BoijeCellCycleModel::SetModelParameters(unsigned phase2gen, unsigned
236     → phase3gen, double probAtoh7, double probPtf1a,
237                                         double probng)
238 {
239     mPhase2gen = phase2gen;
240     mPhase3gen = phase3gen;
241     mprobAtoh7 = probAtoh7;
242     mprobPtf1a = probPtf1a;
243     mprobng = probng;
244 }
245
246
247
248
249
250
251
252
253
254
255 void BoijeCellCycleModel::EnableModeEventOutput(double eventStart,
256     → unsigned seed)
257 {
258     mOutput = true;
259     mEventStartTime = eventStart;
260     mSeed = seed;
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294 }
```

```
245     double currentTime = SimulationTime::Instance()→GetTime() +
246         → mEventStartTime;
247     CellPtr currentCell = GetCell();
248     double currentCellID = (double) currentCell→GetCellId();
249     (*LogFile::Instance()) << currentTime << "\t" << mSeed << "\t" <<
250         → currentCellID << "\t" << mMitoticMode << "\n";
251 }
252
253 void
254     ← BoijeCellCycleModel :: EnableSequenceSampler(boost :: shared_ptr<AbstractCellProperty
255         → label)
256 {
257     mSequenceSampler = true;
258     mp_label_Type = label;
259 }
260
261 void
262     ← BoijeCellCycleModel :: EnableModelDebugOutput(boost :: shared_ptr<ColumnDataWriter>
263         → debugWriter)
264 {
265     mDebug = true;
266     mDebugWriter = debugWriter;
267
268     mTimeID = mDebugWriter→DefineUnlimitedDimension("Time", "h");
269     mVarIDs.push_back(mDebugWriter→DefineVariable("CellID", "No"));
270     mVarIDs.push_back(mDebugWriter→DefineVariable("Generation", "No"));
271     mVarIDs.push_back(mDebugWriter→DefineVariable("MitoticMode",
272         → "Mode"));
273     mVarIDs.push_back(mDebugWriter→DefineVariable("atoh7Set",
274         → "Percentile"));
275     mVarIDs.push_back(mDebugWriter→DefineVariable("atoh7RV",
276         → "Percentile"));
277     mVarIDs.push_back(mDebugWriter→DefineVariable("ptf1aSet",
278         → "Percentile"));
279     mVarIDs.push_back(mDebugWriter→DefineVariable("ptf1aRV",
280         → "Percentile"));
281     mVarIDs.push_back(mDebugWriter→DefineVariable("ngSet",
282         → "Percentile"));
283     mVarIDs.push_back(mDebugWriter→DefineVariable("ngRV",
284         → "Percentile"));
285
286     mDebugWriter→EndDefineMode();
287 }
```

```
275
276 void BoijeCellCycleModel::WriteDebugData(double atoh7RV, double ptf1aRV,
277   → double ngRV)
278 {
279     double currentTime = SimulationTime::Instance()→GetTime();
280     CellPtr currentCell = GetCell();
281     double currentCellID = (double) currentCell→GetCellId();
282
283     mDebugWriter→PutVariable(mTimeID, currentTime);
284     mDebugWriter→PutVariable(mVarIDs[0], currentCellID);
285     mDebugWriter→PutVariable(mVarIDs[1], mGeneration);
286     mDebugWriter→PutVariable(mVarIDs[2], mMitoticMode);
287     mDebugWriter→PutVariable(mVarIDs[3], mprobAtoh7);
288     mDebugWriter→PutVariable(mVarIDs[4], atoh7RV);
289     mDebugWriter→PutVariable(mVarIDs[5], mprobPtf1a);
290     mDebugWriter→PutVariable(mVarIDs[6], ptf1aRV);
291     mDebugWriter→PutVariable(mVarIDs[7], mprobng);
292     mDebugWriter→PutVariable(mVarIDs[8], ngRV);
293     mDebugWriter→AdvanceAlongUnlimitedDimension();
294 }
295 /*****
296 * UNUSED FUNCTIONS (required for class nonvirtuality, do not remove)
297 *****/
298
299 double BoijeCellCycleModel::GetAverageTransitCellCycleTime()
300 {
301     return mCellCycleDuration;
302 }
303
304 double BoijeCellCycleModel::GetAverageStemCellCycleTime()
305 {
306     return mCellCycleDuration;
307 }
308
309 void BoijeCellCycleModel::OutputCellCycleModelParameters(out_stream&
310   → rParamsFile)
311 {
312     *rParamsFile << "\t\t\t<CellCycleDuration>" << mCellCycleDuration <<
313     → "</CellCycleDuration>\n";
314
315     // Call method on direct parent class
```

```

314     → AbstractSimpleCellCycleModel::OutputCellCycleModelParameters(rParamsFile);
315 }
316
317 // Serialization for Boost ≥ 1.36
318 #include "SerializationExportWrapperForCpp.hpp"
319 CHASTE_CLASS_EXPORT(BoijeCellCycleModel)

```

---

### 17.1.20 /src/BoijeCellCycleModel.hpp

```

1 #ifndef BOIJECELLCYCLEMODEL_HPP_
2 #define BOIJECELLCYCLEMODEL_HPP_
3
4 #include "AbstractSimpleCellCycleModel.hpp"
5 #include "RandomNumberGenerator.hpp"
6 #include "Cell.hpp"
7 #include "DifferentiatedCellProliferativeType.hpp"
8 #include "SmartPointers.hpp"
9 #include "ColumnDataWriter.hpp"
10 #include "LogFile.hpp"
11 #include "CellLabel.hpp"
12
13 #include "BoijeRetinalNeuralFates.hpp"
14
15
16 /*****
17 * BOIJE CELL CYCLE MODEL
18 * As described in Boije et al. 2015 [Boije2015] doi:
19 *   → 10.1016/j.devcel.2015.08.011
20 * USE: By default, BoijeCellCycleModels are constructed with the
21 *   → parameters reported in [Boijke2015].
22 * In normal use, the model steps through three phases of probabilistic
23 *   → transcription factor signalling.
24 * These signals determine the mitotic mode and the fate of offspring.
25 * NB: the Boije model is purely generational and assumes a generation
26 *   → time of 1; time in Boije simulations
27 * thus represents generation number rather than proper time. Refactor for
28 *   → use with simulations that refer
29 * to clocktime!!
30 *
31 */

```

```
28 * 2 per-model-event output modes:
29 * EnableModeEventOutput() enables mitotic mode event logging-all cells
30   ↳ will write to the singleton log file
31 * EnableModelDebugOutput() enables more detailed debug output, each seed
32   ↳ will have its own file written to
33 *
34 * 1 mitotic-event-sequence sampler (only samples one "path" through the
35   ↳ lineage):
36 * EnableSequenceSampler() - one "sequence" of progenitors writes mitotic
37   ↳ event type to a string in the singleton log file
38 *
39 *****/
40
41 class BoijeCellCycleModel : public AbstractSimpleCellCycleModel
42 {
43     friend class TestSimpleCellCycleModels;
44
45     /** Needed for serialization.*/
46     friend class boost::serialization::access;
47     /**
48      * Archive the cell-cycle model and random number generator, never
49      ↳ used directly - boost uses this.
50      *
51      * @param archive the archive
52      * @param version the current version of this class
53      */
54     template<class Archive>
55     void serialize(Archive & archive, const unsigned int version)
56     {
57         archive &
58             boost::serialization::base_object<AbstractSimpleCellCycleModel>(*this);
59
60         SerializableSingleton<RandomNumberGenerator>* p_wrapper =
61             RandomNumberGenerator::Instance()→GetSerializationWrapper();
62         archive & p_wrapper;
63         archive & mCellCycleDuration;
64         archive & mGeneration;
65     }
66 }
```

```
64 //Private write functions for models
65 void WriteModeEventOutput();
66 void WriteDebugData(double atoh7RV, double ptf1aRV, double ngRV);
67
68 protected:
69     //mode/output variables
70     bool mOutput;
71     double mEventStartTime;
72     bool mSequenceSampler;
73     bool mSeqSamplerLabelSister;
74     //debug writer stuff
75     bool mDebug;
76     int mTimeID;
77     std::vector<int> mVarIDs;
78     boost::shared_ptr<ColumnDataWriter> mDebugWriter;
79     //model parameters and state memory vars
80     unsigned mGeneration;
81     unsigned mPhase2gen;
82     unsigned mPhase3gen;
83     double mprobAtoh7;
84         double mprobPtfa;
85         double mprobng;
86         bool mAtoh7Signal;
87         bool mPtfaSignal;
88         bool mNgSignal;
89     unsigned mMitoticMode;
90     unsigned mSeed;
91     boost::shared_ptr<AbstractCellProperty> mp_PostMitoticType;
92     boost::shared_ptr<AbstractCellProperty> mp_RGC_Type;
93     boost::shared_ptr<AbstractCellProperty> mp_AC_HC_Type;
94     boost::shared_ptr<AbstractCellProperty> mp_PR_BC_Type;
95     boost::shared_ptr<AbstractCellProperty> mp_label_Type;
96
97 /**
98 * Protected copy-constructor for use by CreateCellCycleModel().
99 *
100 * The only way for external code to create a copy of a cell cycle
101 model
102     * is by calling that method, to ensure that a model of the correct
103     * subclass is created.
104     * This copy-constructor helps subclasses to ensure that all member
105     * variables are correctly copied when this happens.
106     *
```

```
104     * This method is called by child classes to set member variables for
105     * a daughter cell upon cell division.
106     * Note that the parent cell cycle model will have had
107     * ResetForDivision() called just before CreateCellCycleModel() is
108     * called,
109     * so performing an exact copy of the parent is suitable behaviour.
110     * Any daughter-cell-specific initialisation
111     * can be done in InitialiseDaughterCell().
112     *
113     * @param rModel the cell cycle model to copy.
114     */
115 BoijeCellCycleModel(const BoijeCellCycleModel& rModel);
116
117 public:
118
119 /**
120  * Constructor - just a default, mBirthTime is set in the
121  * AbstractCellCycleModel class.
122  */
123 BoijeCellCycleModel();
124
125 /**
126  * SetCellCycleDuration() method to set length of cell cycle (default
127  * 1.0 as Boije's model is purely generational)
128  */
129 void SetCellCycleDuration();
130
131 /**
132  * Overridden builder method to create new copies of
133  * this cell-cycle model.
134  *
135  * @return new cell-cycle model
136  */
137 AbstractCellCycleModel* CreateCellCycleModel();
138
139 /**
140  * Overridden ResetForDivision() method.
141  * Contains general mitotic mode logic
142  */
143 void ResetForDivision();
144
145 /**
146  * Overridden InitialiseDaughterCell() method.
```

```

141     * Used to implement asymmetric mitotic mode
142     */
143     void InitialiseDaughterCell();
144
145     /**
146      * Sets the cell's generation.
147      *
148      * @param generation the cell's generation
149      */
150     void SetGeneration(unsigned generation);
151
152     /**
153      * @return the cell's generation.
154      */
155     unsigned GetGeneration() const;
156
157     /*****
158      * Model setup functions:
159      * Set TF signal probabilities with SetModelParameters
160      * Set Postmitotic and specification AbstractCellProperties w/ the
161      * relevant functions
162      * By default these are available in BoijeRetinalNeuralFates.hpp
163      *
164      * NB: The Boije model lumps together amacrine & horizontal cells,
165      * photoreceptors & bipolar neurons
166      *****/
167
168     void SetModelParameters(unsigned phase2gen = 3, unsigned phase3gen =
169     → 5, double probAtoh7 = 0.32, double probPtf1a = 0.3, double probng
170     → = 0.8);
171     void SetPostMitoticType(boost::shared_ptr<AbstractCellProperty>
172     → p_PostMitoticType);
173     void SetSpecifiedTypes(boost::shared_ptr<AbstractCellProperty>
174     → p_RGC_Type, boost::shared_ptr<AbstractCellProperty> p_AC_HC_Type,
175     → boost::shared_ptr<AbstractCellProperty> p_PR_BC_Type);
176
177     //Functions to enable per-cell mitotic mode logging for mode rate &
178     → sequence sampling fixtures
179     //Uses singleton logfile
180     void EnableModeEventOutput(double eventStart, unsigned seed);
181     void EnableSequenceSampler(boost::shared_ptr<AbstractCellProperty>
182     → label);
183
184

```

```

175     //More detailed debug output. Needs a ColumnDataWriter passed to it
176     //Only declare ColumnDataWriter directory, filename, etc; do not set
177     //up otherwise
178     void EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
179     ↪ debugWriter);
180
181     /**
182      * Overridden GetAverageTransitCellCycleTime() method.
183      *
184      * @return the average of mMinCellCycleDuration and
185      ↪ mMaxCellCycleDuration
186      */
187     double GetAverageTransitCellCycleTime();
188
189     /**
190      * Overridden GetAverageStemCellCycleTime() method.
191      *
192      * @return the average of mMinCellCycleDuration and
193      ↪ mMaxCellCycleDuration
194      */
195     double GetAverageStemCellCycleTime();
196
197     /**
198      * Overridden OutputCellCycleModelParameters() method.
199      *
200      * @param rParamsFile the file stream to which the parameters are
201      ↪ output
202      */
203     virtual void OutputCellCycleModelParameters(out_stream& rParamsFile);
204
205 #endif /*BOIJECELLCYCLEMODEL_HPP_*/

```

---

### 17.1.21 /src/BoijeRetinalNeuralFates.cpp

---

```

1 #include " ../../../../../projects/ISP/src/BoijeRetinalNeuralFates.hpp"
2
3 RetinalGanglion::RetinalGanglion(unsigned colour)

```

```
4     : AbstractCellProperty(),
5     mColour(colour)
6 {
7 }
8
9 RetinalGanglion::~RetinalGanglion()
10 {
11 }
12
13 unsigned RetinalGanglion::GetColour() const
14 {
15     return mColour;
16 }
17
18 AmacrineHorizontal::AmacrineHorizontal(unsigned colour)
19     : AbstractCellProperty(),
20     mColour(colour)
21 {
22 }
23
24 AmacrineHorizontal::~AmacrineHorizontal()
25 {
26 }
27
28 unsigned AmacrineHorizontal::GetColour() const
29 {
30     return mColour;
31 }
32
33 ReceptorBipolar::ReceptorBipolar(unsigned colour)
34     : AbstractCellProperty(),
35     mColour(colour)
36 {
37 }
38
39 ReceptorBipolar::~ReceptorBipolar()
40 {
41 }
42
43 unsigned ReceptorBipolar::GetColour() const
44 {
45     return mColour;
46 }
```

```

47
48 #include "SerializationExportWrapperForCpp.hpp"
49 // Declare identifier for the serializer
50 CHASTE_CLASS_EXPORT(RetinalGanglion)
51 CHASTE_CLASS_EXPORT(AmacrineHorizontal)
52 CHASTE_CLASS_EXPORT(ReceptorBipolar)

```

---

### 17.1.22 /src/BoijeRetinalNeuralFates.hpp

```

1 #ifndef BOIJERETINALNEURALFATES_HPP_
2 #define BOIJERETINALNEURALFATES_HPP_

3
4 #include <boost/shared_ptr.hpp>
5 #include "AbstractCellProperty.hpp"
6 #include "ChasteSerialization.hpp"
7 #include <boost/serialization/base_object.hpp>

8
9 class RetinalGanglion : public AbstractCellProperty
10 {
11 private:
12
13     /**
14      * Colour for use by visualizer.
15      */
16     unsigned mColour;

17     /** Needed for serialization.*/
18     friend class boost::serialization::access;
19     /**
20      * Archive the member variables.
21      *
22      * @param archive the archive
23      * @param version the current version of this class
24      */
25     template<class Archive>
26     void serialize(Archive & archive, const unsigned int version)
27     {
28         archive &
29             boost::serialization::base_object<AbstractCellProperty>(*this);
30         archive & mColour;
31     }
32

```

```
33 public:
34
35     /**
36      * Constructor.
37      *
38      * @param colour what colour cells with this property should be in
39      * the visualizer (defaults to 6)
40      */
41     RetinalGanglion(unsigned colour=3);
42
43     /**
44      * Destructor.
45      */
46     virtual ~RetinalGanglion();
47
48     /**
49      * @return #mColour.
50      */
51     unsigned GetColour() const;
52
53 class AmacrineHorizontal : public AbstractCellProperty
54 {
55 private:
56
57     /**
58      * Colour for use by visualizer.
59      */
60     unsigned mColour;
61
62     /** Needed for serialization.*/
63     friend class boost::serialization::access;
64
65     /**
66      * Archive the member variables.
67      *
68      * @param archive the archive
69      * @param version the current version of this class
70      */
71     template<class Archive>
72     void serialize(Archive & archive, const unsigned int version)
73     {
74         archive &
75             boost::serialization::base_object<AbstractCellProperty>(*this);
```

```
74         archive & mColour;
75     }
76
77 public:
78
79 /**
80 * Constructor.
81 *
82 * @param colour what colour cells with this property should be in
83 * the visualizer (defaults to 6)
84 */
85 AmacrineHorizontal(unsigned colour=4);
86
87 /**
88 * Destructor.
89 */
90 virtual ~AmacrineHorizontal();
91
92 /**
93 * @return #mColour.
94 */
95 unsigned GetColour() const;
96 };
97
98 class ReceptorBipolar : public AbstractCellProperty
99 {
100 private:
101 /**
102 * Colour for use by visualizer.
103 */
104 unsigned mColour;
105
106 /** Needed for serialization.*/
107 friend class boost::serialization::access;
108 /**
109 * Archive the member variables.
110 *
111 * @param archive the archive
112 * @param version the current version of this class
113 */
114 template<class Archive>
115 void serialize(Archive & archive, const unsigned int version)
```

```

116     {
117         archive &
118             → boost::serialization::base_object<AbstractCellProperty>(*this);
119         archive & mColour;
120     }
121
122 public:
123 /**
124 * Constructor.
125 *
126 * @param colour what colour cells with this property should be in
127 → the visualizer (defaults to 6)
128 */
129 ReceptorBipolar(unsigned colour=5);
130
131 /**
132 * Destructor.
133 */
134 virtual ~ReceptorBipolar();
135
136 /**
137 * @return #mColour.
138 */
139 unsigned GetColour() const;
140
141 #include "SerializationExportWrapper.hpp"
142 // Declare identifier for the serializer
143 CHASTE_CLASS_EXPORT(RetinalGanglion)
144 CHASTE_CLASS_EXPORT(AmacrineHorizontal)
145 CHASTE_CLASS_EXPORT(ReceptorBipolar)
146
147 #endif /* BOIJERETINALNEURALFATES_HPP_ */

```

---

### 17.1.23 /src/GomesCellCycleModel.cpp

---

```

1 #include "GomesCellCycleModel.hpp"
2 #include "GomesRetinalNeuralFates.hpp"
3
4 GomesCellCycleModel::GomesCellCycleModel() :

```

```

5     AbstractSimpleCellCycleModel(), mOutput(false),
6     ↳ mEventStartTime(), mSequenceSampler(false),
7     ↳ mSeqSamplerLabelSister(
8         false), mDebug(false), mTimeID(), mVarIDs(),
9         ↳ mDebugWriter(), mNormalMu(3.9716),
10        ↳ mNormalSigma(0.32839), mPP(
11            .055), mPD(.221), mpBC(.128), mpAC(.106), mpMG(.028),
12            ↳ mMitoticMode(), mSeed(), mp_PostMitoticType(),
13            ↳ mp_RPh_Type(), mp_BC_Type(), mp_AC_Type(),
14            ↳ mp_MG_Type(), mp_label_Type()
15    {
16    }
17
18 GomesCellCycleModel::GomesCellCycleModel(const GomesCellCycleModel&
19     rModel) :
20     AbstractSimpleCellCycleModel(rModel), mOutput(rModel.mOutput),
21     ↳ mEventStartTime(rModel.mEventStartTime), mSequenceSampler(
22         rModel.mSequenceSampler),
23         ↳ mSeqSamplerLabelSister(rModel.mSeqSamplerLabelSister),
24         ↳ mDebug(rModel.mDebug), mTimeID(
25             rModel.mTimeID), mVarIDs(rModel.mVarIDs),
26             ↳ mDebugWriter(rModel.mDebugWriter), mNormalMu(
27                 rModel.mNormalMu), mNormalSigma(rModel.mNormalSigma),
28                 ↳ mPP(rModel.mPP), mPD(rModel.mPD), mpBC(
29                     rModel.mpBC), mpAC(rModel.mpAC), mpMG(rModel.mpMG),
30                     ↳ mMitoticMode(rModel.mMitoticMode), mSeed(
31                         rModel.mSeed),
32                         ↳ mp_PostMitoticType(rModel.mp_PostMitoticType),
33                         ↳ mp_RPh_Type(rModel.mp_RPh_Type), mp_BC_Type(
34                             rModel.mp_BC_Type), mp_AC_Type(rModel.mp_AC_Type),
35                             ↳ mp_MG_Type(rModel.mp_MG_Type), mp_label_Type(
36                                 rModel.mp_label_Type)
37    {
38    }
39
40 AbstractCellCycleModel* GomesCellCycleModel::CreateCellCycleModel()
41 {
42     return new GomesCellCycleModel(*this);
43 }
44
45 void GomesCellCycleModel::SetCellCycleDuration()
46 {
47     /*****
```

```

31 * CELL CYCLE DURATION RANDOM VARIABLE
32 *****/
33
34 RandomNumberGenerator* p_random_number_generator =
35     → RandomNumberGenerator::Instance();
36
37 //Gomes cell cycle length determined by lognormal distribution with
38 // default mean 56 hr, std 18.9 hrs.
39 mCellCycleDuration =
40     → exp(p_random_number_generator→NormalRandomDeviate(mNormalMu,
41     → mNormalSigma));
42 }
43
44 void GomesCellCycleModel::ResetForDivision()
45 {
46     *****
47     * Mitotic mode rules
48     * *****/
49 RandomNumberGenerator* p_random_number_generator =
50     → RandomNumberGenerator::Instance();
51
52 //Check time in lineage and determine current mitotic mode phase
53 mMitoticMode = 0; //0=PP;1=PD;2=DD
54
55 *****
56     * MITOTIC MODE RANDOM VARIABLE
57 *****/
58 //initialise mitoticmode random variable, set mitotic mode
59 // appropriately after comparing to mode probability array
60 double mitoticModeRV = p_random_number_generator→ranf();
61
62 if (mitoticModeRV > mPP && mitoticModeRV ≤ mPP + mPD)
63 {
64     mMitoticMode = 1;
65 }
66 if (mitoticModeRV > mPP + mPD)
67 {
68     mMitoticMode = 2;
69 }
70
71 *****
72     * Write mitotic event to file if appropriate

```

```

67      * mDebug: many files: detailed per-lineage info switch; intended for
68  ↳ TestHeInductionCountFixture
69      * mOutput: 1 file: time, seed, cellID, mitotic mode, intended for
70  ↳ TestHeMitoticModeRateFixture
71      * *****/
72
73
74      if (mDebug)
75      {
76          WriteDebugData(mitoticModeRV);
77      }
78
79      if (mOutput)
80      {
81          WriteModeEventOutput();
82      }
83
84      //set new cell cycle length (will be overwritten with DBL_MAX for DD
85  ↳ divisions)
86      AbstractSimpleCellCycleModel::ResetForDivision();
87
88      *****
89      * Symmetric postmitotic specification rule
90      * -(asymmetric postmitotic rule specified in
91  ↳ InitialiseDaughterCell());
92      * *****/
93
94      if (mMitoticMode == 2)
95      {
96          mpCell→SetCellProliferativeType(mp_PostMitoticType);
97          mCellCycleDuration = DBL_MAX;
98          *****
99          * SPECIFICATION RANDOM VARIABLE
100         *****/
101         double specificationRV = p_random_number_generator→ranf();
102         if (specificationRV ≤ mpMG)
103         {
104             mpCell→AddCellProperty(mp_MG_Type);
105         }
106         if (specificationRV > mpMG && specificationRV ≤ mpMG + mpAC)
107         {
108             mpCell→AddCellProperty(mp_AC_Type);
109         }
110     }

```

```

105     if (specificationRV > mpMG + mpAC && specificationRV ≤ mpMG +
106         → mpAC + mpBC)
107     {
108         mpCell→AddCellProperty(mp_BC_Type);
109     }
110     if (specificationRV > mpMG + mpAC + mpBC)
111     {
112         mpCell→AddCellProperty(mp_RPh_Type);
113     }
114
115 /*****  

116 * SEQUENCE SAMPLER  

117 *****/
118 //if the sequence sampler has been turned on, check for the label &
119 //→ write mitotic mode to log
120 //50% chance of each daughter cell from a mitosis inheriting the
121 //→ label
122 if (mSequenceSampler)
123 {
124     if (mpCell→HasCellProperty<CellLabel>())
125     {
126         (*LogFile::Instance()) << mMitoticMode;
127         double labelRV = p_random_number_generator→ranf();
128         if (labelRV ≤ .5)
129         {
130             mSeqSamplerLabelSister = true;
131             mpCell→RemoveCellProperty<CellLabel>();
132         }
133         else
134         {
135             mSeqSamplerLabelSister = false;
136         }
137     }
138     else
139     {
140         //prevents lost-label cells from labelling their progeny
141         mSeqSamplerLabelSister = false;
142     }
143 }
144 void GomesCellCycleModel::InitialiseDaughterCell()

```

```

145 {
146     if (mMitoticMode == 0)
147     {
148         //daughter cell's mCellCycleDuration is copied from parent; reset
149         // to new value from gamma PDF here
150         SetCellCycleDuration();
151     }
152
153     *****
154     * PD-type division
155     *****/
156
157     if (mMitoticMode == 1)
158     {
159         RandomNumberGenerator* p_random_number_generator =
160             RandomNumberGenerator::Instance();
161         mpCell->SetCellProliferativeType(mp_PostMitoticType);
162         mCellCycleDuration = DBL_MAX;
163
164         *****
165         * SPECIFICATION RULES
166         *****/
167         double specificationRV = p_random_number_generator->ranf();
168         if (specificationRV <= mpMG)
169         {
170             mpCell->AddCellProperty(mp_MG_Type);
171         }
172         if (specificationRV > mpMG && specificationRV <= mpMG + mpAC)
173         {
174             mpCell->AddCellProperty(mp_AC_Type);
175         }
176         if (specificationRV > mpMG + mpAC && specificationRV <= mpMG +
177             mpAC + mpBC)
178         {
179             mpCell->AddCellProperty(mp_BC_Type);
180         }
181     }
182
183     if (mMitoticMode == 2)
184     {

```

```
185     RandomNumberGenerator* p_random_number_generator =
186         ↳ RandomNumberGenerator::Instance();
187     //remove the fate assigned to the parent cell in
188     ↳ ResetForDivision, then assign the sister fate as usual
189     mpCell→RemoveCellProperty<AbstractCellProperty>();
190     mpCell→SetCellProliferativeType(mp_PostMitoticType);
191
192     /*****
193     * SPECIFICATION RULES
194     *****/
195     double specificationRV = p_random_number_generator→ranf();
196     if (specificationRV ≤ mpMG)
197     {
198         mpCell→AddCellProperty(mp_MG_Type);
199     }
200     if (specificationRV > mpMG && specificationRV ≤ mpMG + mpAC)
201     {
202         mpCell→AddCellProperty(mp_AC_Type);
203     }
204     if (specificationRV > mpMG + mpAC && specificationRV ≤ mpMG +
205         ↳ mpAC + mpBC)
206     {
207         mpCell→AddCellProperty(mp_BC_Type);
208     }
209     if (specificationRV > mpMG + mpAC + mpBC)
210     {
211         mpCell→AddCellProperty(mp_RPh_Type);
212     }
213
214     /*****
215     * SEQUENCE SAMPLER
216     *****/
217     if (mSequenceSampler)
218     {
219         if (mSeqSamplerLabelSister)
220         {
221             mpCell→AddCellProperty(mp_label_Type);
222             mSeqSamplerLabelSister = false;
223         }
224         else
225         {
226             mpCell→RemoveCellProperty<CellLabel>();
```

```

225         }
226     }
227 }
228
229 void GomesCellCycleModel::SetModelParameters(const double normalMu, const
230   ↵  double normalSigma, const double PP,
231   ↵  const double PD, const
232   ↵  double pBC, const double
233   ↵  pAC, const double pMG)
234 {
235     mNormalMu = normalMu;
236     mNormalSigma = normalSigma;
237     mPP = PP;
238     mPD = PD;
239     mpBC = pBC;
240     mpAC = pAC;
241     mpMG = pMG;
242 }
243
244 void
245   ↵  GomesCellCycleModel::SetModelProperties(boost::shared_ptr<AbstractCellProperty>
246   ↵  p_RPh_Type,
247   ↵  boost::shared_ptr<AbstractCellProperty>
248   ↵  p_AC_Type,
249   ↵  boost::shared_ptr<AbstractCellProperty>
250   ↵  p_BC_Type,
251   ↵  boost::shared_ptr<AbstractCellProperty>
252   ↵  p_MG_Type)
253 {
254     mp_RPh_Type = p_RPh_Type;
255     mp_AC_Type = p_AC_Type;
256     mp_BC_Type = p_BC_Type;
257     mp_MG_Type = p_MG_Type;
258 }
259
260 void
261   ↵  GomesCellCycleModel::SetPostMitoticType(boost::shared_ptr<AbstractCellProperty>
262   ↵  p_PostMitoticType)
263 {

```

```

255     mp_PostMitoticType = p_PostMitoticType;
256 }
257
258 void GomesCellCycleModel::EnableModeEventOutput(double eventStart,
259     ↪ unsigned seed)
260 {
261     mOutput = true;
262     mEventStartTime = eventStart;
263     mSeed = seed;
264 }
265
266 void GomesCellCycleModel::WriteModeEventOutput()
267 {
268     double currentTime = SimulationTime::Instance()->GetTime() +
269         ↪ mEventStartTime;
270     CellPtr currentCell = GetCell();
271     double currentCellID = (double) currentCell->GetCellId();
272     (*LogFile::Instance()) << currentTime << "\t" << mSeed << "\t" <<
273         ↪ currentCellID << "\t" << mMitoticMode << "\n";
274 }
275
276 void
277     ↪ GomesCellCycleModel::EnableSequenceSampler(boost::shared_ptr<AbstractCellProperty>
278         ↪ label)
279 {
280     mSequenceSampler = true;
281     mp_label_Type = label;
282 }
283
284 void
285     ↪ GomesCellCycleModel::EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
286         ↪ debugWriter)
287 {
288     mDebug = true;
289     mDebugWriter = debugWriter;
290
291     mTimeID = mDebugWriter->DefineUnlimitedDimension("Time", "h");
292     mVarIDs.push_back(mDebugWriter->DefineVariable("CellID", "No"));
293     mVarIDs.push_back(mDebugWriter->DefineVariable("CycleDuration",
294         ↪ "h"));
295     mVarIDs.push_back(mDebugWriter->DefineVariable("PP", "Percentile"));
296     mVarIDs.push_back(mDebugWriter->DefineVariable("PD", "Percentile"));

```

```
290     mVarIDs.push_back(mDebugWriter->DefineVariable("Dieroll",
291         "Percentile"));
292     mVarIDs.push_back(mDebugWriter->DefineVariable("MitoticMode",
293         "Mode"));
294 }
295
296 void GomesCellCycleModel::WriteDebugData(double percentileRoll)
297 {
298     double currentTime = SimulationTime::Instance()->GetTime();
299     CellPtr currentCell = GetCell();
300     double currentCellID = (double) currentCell->GetCellId();
301
302     mDebugWriter->PutVariable(mTimeID, currentTime);
303     mDebugWriter->PutVariable(mVarIDs[0], currentCellID);
304     mDebugWriter->PutVariable(mVarIDs[1], mCellCycleDuration);
305     mDebugWriter->PutVariable(mVarIDs[2], mPP);
306     mDebugWriter->PutVariable(mVarIDs[3], mPD);
307     mDebugWriter->PutVariable(mVarIDs[4], percentileRoll);
308     mDebugWriter->PutVariable(mVarIDs[5], mMitoticMode);
309     mDebugWriter->AdvanceAlongUnlimitedDimension();
310 }
311
312 /*****
313 * UNUSED FUNCTIONS (required for class nonvirtuality, do not remove)
314 *****/
315
316 double GomesCellCycleModel::GetAverageTransitCellCycleTime()
317 {
318     return (0.0);
319 }
320
321 double GomesCellCycleModel::GetAverageStemCellCycleTime()
322 {
323     return (0.0);
324 }
325
326 void GomesCellCycleModel::OutputCellCycleModelParameters(out_stream&
327     rParamsFile)
328 {
329     *rParamsFile << "\t\t\t<CellCycleDuration>" << mCellCycleDuration <<
330     "</CellCycleDuration>\n";
```

```

329
330     // Call method on direct parent class
331
332     ↳ AbstractSimpleCellCycleModel::OutputCellCycleModelParameters(rParamsFile);
333
334 }
335 // Serialization for Boost ≥ 1.36
336 #include "SerializationExportWrapperForCpp.hpp"
337 CHASTE_CLASS_EXPORT(GomesCellCycleModel)

```

---

### 17.1.24 /src/GomesCellCycleModel.hpp

```

1 #ifndef GOMESCELLCYCLEMODEL_HPP_
2 #define GOMESCELLCYCLEMODEL_HPP_
3
4 #include "AbstractSimpleCellCycleModel.hpp"
5 #include "RandomNumberGenerator.hpp"
6 #include "Cell.hpp"
7 #include "DifferentiatedCellProliferativeType.hpp"
8 #include "GomesRetinalNeuralFates.hpp"
9 #include "SmartPointers.hpp"
10 #include "ColumnDataWriter.hpp"
11 #include "LogFile.hpp"
12 #include "CellLabel.hpp"
13
14 /*****
15 * GOMES CELL CYCLE MODEL
16 * As described in Gomes et al. 2011 [Gomes2011] doi: 10.1242/dev.059683
17 *
18 * USE: By default, GomesCellCycleModels are constructed with the
19 * parameter fit reported in [Gomes2011].
20 * Cell cycle length, mitotic mode, and postmitotic fate of cells are
21 * determined by independent random variables
22 * PP = symmetric proliferative mitotic mode, both progeny remain mitotic
23 * PD = asymmetric proliferative mitotic mode, one progeny exits the cell
24 *       cycle and differentiates
25 * DD = symmetric differentiative mitotic mode, both progeny exit the
26 *       cell cycle and differentiate
27 *
28 * Change default model parameters with SetModelParameters(<params>);
29 * Set AbstractCellProperties for differentiated neural types with
30 * SetModelProperties();

```

```
26  *
27  * 2 per-model-event output modes:
28  * EnableModeEventOutput() enables mitotic mode event logging-all cells
29  *   ↳ will write to the singleton log file
30  * EnableModelDebugOutput() enables more detailed debug output, each seed
31  *   ↳ will have its own file written to
32  * by a ColumnDataWriter passed to it from the test
33  * (eg. by the SetupDebugOutput helper function in the project simulator)
34  *
35  *
36  * ****
37
38 class GomesCellCycleModel : public AbstractSimpleCellCycleModel
39 {
40     friend class TestSimpleCellCycleModels;
41
42 private:
43
44     /** Needed for serialization.*/
45     friend class boost::serialization::access;
46
47     * Archive the cell-cycle model and random number generator, never
48     * used directly - boost uses this.
49     *
50     * @param archive
51     * @param version the current version of this class
52     */
53     template<class Archive>
54     void serialize(Archive & archive, const unsigned int version)
55     {
56         archive &
57             ↳ boost::serialization::base_object<AbstractSimpleCellCycleModel>(*this);
58
59         SerializableSingleton<RandomNumberGenerator>* p_wrapper =
60
61             ↳ RandomNumberGenerator::Instance()→GetSerializationWrapper();
62         archive & p_wrapper;
63         archive & mCellCycleDuration;
64     }
65 }
```

```
62 //Private write functions for models
63 void WriteModeEventOutput();
64 void WriteDebugData(double percentile);
65
66 protected:
67     //mode/output variables
68     bool mOutput;
69     double mEventStartTime;
70     bool mSequenceSampler;
71     bool mSeqSamplerLabelSister;
72     //debug writer stuff
73     bool mDebug;
74     int mTimeID;
75     std::vector<int> mVarIDs;
76     boost::shared_ptr<ColumnDataWriter> mDebugWriter;
77     //model parameters and state memory vars
78     double mNormalMu;
79     double mNormalSigma;
80     double mPP;
81     double mPD;
82     double mpBC;
83     double mpAC;
84     double mpMG;
85     unsigned mMitoticMode;
86     unsigned mSeed;
87     boost::shared_ptr<AbstractCellProperty> mp_PostMitoticType;
88     boost::shared_ptr<AbstractCellProperty> mp_RPh_Type;
89     boost::shared_ptr<AbstractCellProperty> mp_BC_Type;
90     boost::shared_ptr<AbstractCellProperty> mp_AC_Type;
91     boost::shared_ptr<AbstractCellProperty> mp_MG_Type;
92     boost::shared_ptr<AbstractCellProperty> mp_label_Type;
93
94 /**
95 * Protected copy-constructor for use by CreateCellCycleModel().
96 *
97 * The only way for external code to create a copy of a cell cycle
98 model
99     * is by calling that method, to ensure that a model of the correct
100    * subclass is created.
101     * This copy-constructor helps subclasses to ensure that all member
102    * variables are correctly copied when this happens.
103 *
```

```
102     * This method is called by child classes to set member variables for
103     * a daughter cell upon cell division.
104     * Note that the parent cell cycle model will have had
105     * ResetForDivision() called just before CreateCellCycleModel() is
106     * called,
107     * so performing an exact copy of the parent is suitable behaviour.
108     * Any daughter-cell-specific initialisation
109     * can be done in InitialiseDaughterCell().
110     *
111     * @param rModel the cell cycle model to copy.
112     */
113     GomesCellCycleModel(const GomesCellCycleModel& rModel);
114
115 public:
116
117     /**
118     * Constructor - just a default, mBirthTime is set in the
119     * AbstractCellCycleModel class.
120     */
121     GomesCellCycleModel();
122
123     /**
124     * SetCellCycleDuration() method to set length of cell cycle
125     * (lognormal distribution as specified in [Gomes2011])
126     */
127     void SetCellCycleDuration();
128
129     /**
130     * Overridden builder method to create new copies of
131     * this cell-cycle model.
132     *
133     * @return new cell-cycle model
134     */
135     AbstractCellCycleModel* CreateCellCycleModel();
136
137     /**
138     * Overridden ResetForDivision() method.
139     * Contains general mitotic mode logic
140     */
141     void ResetForDivision();
142
143     /**
144     * Overridden InitialiseDaughterCell() method. Used to implement
145     * asymmetric mitotic mode*/
```

```

138 void InitialiseDaughterCell();
139
140 /* Model setup functions
141 * Set lognormal cell cycle curve properties with *mean and std of
142 * corresponding NORMAL curve*
143 * Model requires valid AbstractCellProperties to assign postmitotic
144 * fate;
145 * Defaults are found in GomesRetinalNeuralFates.hpp
146 * (RodPhotoreceptor, AmacrineCell, BipolarCell, MullerGlia)
147 */
148
149 void SetModelParameters(const double normalMu = 3.9716, const double
150   ↵ normalSigma = .32839, const double PP = .055,
151   ↵ const double PD = .221, const double pBC =
152   ↵ .128, const double pAC = .106, const
153   ↵ double pMG =
154   .028);
155 void SetModelProperties(boost::shared_ptr<AbstractCellProperty>
156   ↵ p_RPh_Type,
157   ↵ boost::shared_ptr<AbstractCellProperty>
158   ↵ p_AC_Type,
159   ↵ boost::shared_ptr<AbstractCellProperty>
160   ↵ p_BC_Type,
161   ↵ boost::shared_ptr<AbstractCellProperty>
162   ↵ p_MG_Type);
163
164 //This should normally be a DifferentiatedCellProliferativeType
165 void SetPostMitoticType(boost::shared_ptr<AbstractCellProperty>
166   ↵ p_PostMitoticType);
167
168 //Functions to enable per-cell mitotic mode logging for mode rate &
169   ↵ sequence sampling fixtures
170 //Uses singleton logfile
171 void EnableModeEventOutput(double eventStart, unsigned seed);
172 void EnableSequenceSampler(boost::shared_ptr<AbstractCellProperty>
173   ↵ label);
174
175 //More detailed debug output. Needs a ColumnDataWriter passed to it
176 //Only declare ColumnDataWriter directory, filename, etc; do not set
177   ↵ up otherwise
178 void EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
179   ↵ debugWriter);

```

```

167     //Not used, but must be overwritten lest GomesCellCycleModels be
168     ↵ abstract
169     double GetAverageTransitCellCycleTime();
170     double GetAverageStemCellCycleTime();
171
172     /**
173      * Overridden OutputCellCycleModelParameters() method.
174      *
175      * @param rParamsFile the file stream to which the parameters are
176      ↵ output
177      */
178     virtual void OutputCellCycleModelParameters(out_stream& rParamsFile);
179
180 #include "SerializationExportWrapper.hpp"
181 // Declare identifier for the serializer
182 CHASTE_CLASS_EXPORT(GomesCellCycleModel)
183 #endif /*GOMESCELLCYCLEMODEL_HPP_*/

```

---

### 17.1.25 /src/GomesRetinalNeuralFates.cpp

```

1 #include "GomesRetinalNeuralFates.hpp"
2
3 RodPhotoreceptor::RodPhotoreceptor(unsigned colour)
4     : AbstractCellProperty(),
5      mColour(colour)
6 {
7 }
8
9 RodPhotoreceptor::~RodPhotoreceptor()
10 {
11 }
12
13 unsigned RodPhotoreceptor::GetColour() const
14 {
15     return mColour;
16 }
17
18 AmacrineCell::AmacrineCell(unsigned colour)
19     : AbstractCellProperty(),
20      mColour(colour)

```

```
21 {
22 }
23
24 AmacrineCell::~AmacrineCell()
25 {
26 }
27
28 unsigned AmacrineCell::GetColour() const
29 {
30     return mColour;
31 }
32
33 BipolarCell::BipolarCell(unsigned colour)
34     : AbstractCellProperty(),
35     mColour(colour)
36 {
37 }
38
39 BipolarCell::~BipolarCell()
40 {
41 }
42
43 unsigned BipolarCell::GetColour() const
44 {
45     return mColour;
46 }
47
48 MullerGlia::MullerGlia(unsigned colour)
49     : AbstractCellProperty(),
50     mColour(colour)
51 {
52 }
53
54 MullerGlia::~MullerGlia()
55 {
56 }
57
58 unsigned MullerGlia::GetColour() const
59 {
60     return mColour;
61 }
62
63 #include "SerializationExportWrapperForCpp.hpp"
```

```
64 // Declare identifier for the serializer
65 CHASTE_CLASS_EXPORT(RodPhotoreceptor)
66 CHASTE_CLASS_EXPORT(AmacrineCell)
67 CHASTE_CLASS_EXPORT(BipolarCell)
68 CHASTE_CLASS_EXPORT(MullerGlia)
```

---

### 17.1.26 /src/GomesRetinalNeuralFates.hpp

```
1 #ifndef GOMESRETINALNEURALFATES_HPP_
2 #define GOMESRETINALNEURALFATES_HPP_
3
4 #include <boost/shared_ptr.hpp>
5 #include "AbstractCellProperty.hpp"
6 #include "ChasteSerialization.hpp"
7 #include <boost/serialization/base_object.hpp>
8
9 class RodPhotoreceptor : public AbstractCellProperty
10 {
11 private:
12
13     /**
14      * Colour for use by visualizer.
15      */
16     unsigned mColour;
17
18     /** Needed for serialization.*/
19     friend class boost::serialization::access;
20     /**
21      * Archive the member variables.
22      *
23      * @param archive the archive
24      * @param version the current version of this class
25      */
26     template<class Archive>
27     void serialize(Archive & archive, const unsigned int version)
28     {
29         archive &
30             boost::serialization::base_object<AbstractCellProperty>(*this);
31         archive & mColour;
32     }
33 public:
```

```
34
35  /**
36   * Constructor.
37   *
38   * @param colour what colour cells with this property should be in
39   * the visualizer (defaults to 6)
40   */
41   RodPhotoreceptor(unsigned colour=3);
42
43  /**
44   * Destructor.
45   */
46  virtual ~RodPhotoreceptor();
47
48  /**
49   * @return #mColour.
50   */
51  unsigned GetColour() const;
52 };
53
54 class AmacrineCell : public AbstractCellProperty
55 {
56 private:
57  /**
58   * Colour for use by visualizer.
59   */
60  unsigned mColour;
61
62  /** Needed for serialization.*/
63  friend class boost::serialization::access;
64  /**
65   * Archive the member variables.
66   *
67   * @param archive the archive
68   * @param version the current version of this class
69   */
70  template<class Archive>
71  void serialize(Archive & archive, const unsigned int version)
72  {
73      archive &
74      boost::serialization::base_object<AbstractCellProperty>(*this);
75      archive & mColour;
```

```
75     }
76
77 public:
78
79     /**
80      * Constructor.
81      *
82      * @param colour what colour cells with this property should be in
83      * the visualizer (defaults to 6)
84      */
85     AmacrineCell(unsigned colour=6);
86
87     /**
88      * Destructor.
89      */
90     virtual ~AmacrineCell();
91
92     /**
93      * @return #mColour.
94      */
95     unsigned GetColour() const;
96 };
97
98 class BipolarCell : public AbstractCellProperty
99 {
100 private:
101
102     /**
103      * Colour for use by visualizer.
104      */
105     unsigned mColour;
106
107     /** Needed for serialization. */
108     friend class boost::serialization::access;
109
110     /**
111      * Archive the member variables.
112      *
113      * @param archive the archive
114      * @param version the current version of this class
115      */
116     template<class Archive>
117     void serialize(Archive & archive, const unsigned int version)
118     {
```

```
117     archive &
118     → boost::serialization::base_object<AbstractCellProperty>(*this);
119     archive & mColour;
120 }
121
122 public:
123 /**
124 * Constructor.
125 *
126 * @param colour what colour cells with this property should be in
127 * the visualizer (defaults to 6)
128 */
129 BipolarCell(unsigned colour=5);
130
131 /**
132 * Destructor.
133 */
134 virtual ~BipolarCell();
135
136 /**
137 * @return #mColour.
138 */
139 unsigned GetColour() const;
140
141 class MullerGlia : public AbstractCellProperty
142 {
143 private:
144
145 /**
146 * Colour for use by visualizer.
147 */
148 unsigned mColour;
149
150 /** Needed for serialization.*/
151 friend class boost::serialization::access;
152 /**
153 * Archive the member variables.
154 *
155 * @param archive the archive
156 * @param version the current version of this class
157 */
```

```

158     template<class Archive>
159     void serialize(Archive & archive, const unsigned int version)
160     {
161         archive &
162             → boost::serialization::base_object<AbstractCellProperty>(*this);
163         archive & mColour;
164     }
165
166 public:
167
168     /**
169      * Constructor.
170      *
171      * @param colour what colour cells with this property should be in
172      * the visualizer (defaults to 6)
173      */
174     MullerGlia(unsigned colour=6);
175
176     /**
177      * Destructor.
178      */
179     virtual ~MullerGlia();
180
181     /**
182      * @return #mColour.
183      */
184     unsigned GetColour() const;
185 };
186 #include "SerializationExportWrapper.hpp"
187 // Declare identifier for the serializer
188 CHASTE_CLASS_EXPORT(RodPhotoreceptor)
189 CHASTE_CLASS_EXPORT(AmacrineCell)
190 CHASTE_CLASS_EXPORT(BipolarCell)
191 CHASTE_CLASS_EXPORT(MullerGlia)
192
193 #endif /* GOMESRETINALNEURALFATES_HPP_ */

```

---

### 17.1.27 /src/HeAth5Mo.cpp

---

```

1 #include " ../../../../../projects/ISP/src/HeAth5Mo.hpp"
2
3 Ath5Mo::Ath5Mo(unsigned colour)

```

```

4     : AbstractCellProperty(),
5     mColour(colour)
6 {
7 }
8
9 Ath5Mo::~Ath5Mo()
10 {
11 }
12
13 unsigned Ath5Mo::GetColour() const
14 {
15     return mColour;
16 }
17
18 #include "SerializationExportWrapperForCpp.hpp"
19 // Declare identifier for the serializer
20 CHASTE_CLASS_EXPORT(Ath5Mo)

```

---

### 17.1.28 /src/HeAth5Mo.hpp

```

1 #ifndef HEATH5MO_HPP_
2 #define HEATH5MO_HPP_
3
4 #include <boost/shared_ptr.hpp>
5 #include "AbstractCellProperty.hpp"
6 #include "ChasteSerialization.hpp"
7 #include <boost/serialization/base_object.hpp>
8
9 class Ath5Mo : public AbstractCellProperty
10 {
11 private:
12
13     /**
14      * Colour for use by visualizer.
15      */
16     unsigned mColour;
17
18     /** Needed for serialization.*/
19     friend class boost::serialization::access;
20     /**
21      * Archive the member variables.
22      */

```

```

23     * @param archive the archive
24     * @param version the current version of this class
25     */
26     template<class Archive>
27     void serialize(Archive & archive, const unsigned int version)
28     {
29         archive &
30             → boost::serialization::base_object<AbstractCellProperty>(*this);
31         archive & mColour;
32     }
33
34 public:
35 /**
36 * Constructor.
37 *
38 * @param colour what colour cells with this property should be in
39 * the visualizer (defaults to 6)
40 */
41 Ath5Mo(unsigned colour=3);
42 /**
43 * Destructor.
44 */
45 virtual ~Ath5Mo();
46
47 /**
48 * @return #mColour.
49 */
50 unsigned GetColour() const;
51 };
52
53
54 #include "SerializationExportWrapper.hpp"
55 // Declare identifier for the serializer
56 CHASTE_CLASS_EXPORT(Ath5Mo)
57 #endif /* HEATH5MO_HPP */

```

---

### 17.1.29 /src/HeCellCycleModel.cpp

---

```

1 #include "HeCellCycleModel.hpp"
2

```



```

23             rModel.mPhase3PD), mMitoticMode(rModel.mMitoticMode),
24             ↳ mSeed(rModel.mSeed), mTimeDependentCycleDuration(
25             rModel.mTimeDependentCycleDuration),
26             ↳ mPeakRateTime(rModel.mPeakRateTime),
27             ↳ mIncreasingRateSlope(
28             rModel.mIncreasingRateSlope),
29             ↳ mDecreasingRateSlope(rModel.mDecreasingRateSlope),
30             ↳ mBaseGammaScale(
31             rModel.mBaseGammaScale)
32 {
33 }
34
35 AbstractCellCycleModel* HeCellCycleModel::CreateCellCycleModel()
36 {
37     return new HeCellCycleModel(*this);
38 }
39
40 void HeCellCycleModel::SetCellCycleDuration()
41 {
42     RandomNumberGenerator* p_random_number_generator =
43         ↳ RandomNumberGenerator::Instance();
44
45     /***** CELL CYCLE DURATION RANDOM VARIABLE *****/
46
47     if (!mTimeDependentCycleDuration) //Normal operation, cell cycle
48         ↳ length stays constant
49     {
50         //He cell cycle length determined by shifted gamma distribution
51         ↳ reflecting 4 hr refractory period followed by gamma pdf
52         mCellCycleDuration = mGammaShift +
53             ↳ p_random_number_generator->GammaRandomDeviate(mGammaShape,
54             ↳ mGammaScale);
55     }
56
57     ****
58     * Variable cycle length
59     * Give -ve mIncreasingRateSlope and +ve mDecreasingRateSlope,
60     * cell cycle length linearly declines (increasing rate), then
61     ↳ increases, switching at mPeakRateTime
62     ****/
63     else
64 }
```

```

55    {
56        double currTime = SimulationTime::Instance()→GetTime();
57        if (currTime ≤ mPeakRateTime)
58        {
59            mGammaScale = std::max((mBaseGammaScale - currTime *
60                → mIncreasingRateSlope), .000000000001);
61        }
62        if (currTime > mPeakRateTime)
63        {
64            mGammaScale = std::max(
65                ((mBaseGammaScale - mPeakRateTime *
66                    → mIncreasingRateSlope)
67                     + (mBaseGammaScale + (currTime -
68                        → mPeakRateTime) * mDecreasingRateSlope)),
69                .000000000001);
70        }
71    }
72
73 void HeCellCycleModel::ResetForDivision()
74 {
75     /*****
76     * TIME IN LINEAGE DEPENDENT MITOTIC MODE PHASE RULES
77     * *****/
78     RandomNumberGenerator* p_random_number_generator =
79         → RandomNumberGenerator::Instance();
80
81     double currentTiL = SimulationTime::Instance()→GetTime() +
82         → mTiLOffset;
83
84     /*Rule logic defaults to phase 1 behaviour, checks for currentTiL >
85      → phaseBoundaries and changes
86      currentPhase and subsequently mMitoticMode as appropriate*/
87     unsigned currentPhase = 1;
88     mMitoticMode = 0;
89
90     //Check time in lineage and determine current mitotic mode phase
91     *****
92     * Phase boundary & deterministic mitotic mode rules

```

```

90     *****/
91     if (currentTiL > mMitoticModePhase2 && currentTiL <
92         → mMitoticModePhase3)
93     {
94         //if current TiL is > phase 2 boundary time, set the currentPhase
95         → appropriately
96         currentPhase = 2;
97
98         //if deterministic mode is enabled, PD divisions are guaranteed
99         → unless this is an Ath5 morphant
100        if (mDeterministic)
101        {
102            mMitoticMode = 1; //0=PP;1=PD;2=DD
103            if (mpCell→HasCellProperty<Ath5Mo>()) //Ath5 morphants
104                → undergo PP rather than PD divisions in 80% of cases
105            {
106                double ath5RV = p_random_number_generator→ranf();
107                if (ath5RV ≤ .8)
108                {
109                    mMitoticMode = 0;
110                }
111            }
112
113            if (currentTiL > mMitoticModePhase3)
114            {
115                //if current TiL is > phase 3 boundary time, set the currentPhase
116                → appropriately
117                currentPhase = 3;
118                if (mDeterministic)
119                {
120                    //if deterministic mode is enabled, DD divisions are
121                    → guaranteed
122                    mMitoticMode = 2;
123                }
124
125            /*****
126             * MITOTIC MODE RANDOM VARIABLE
127             *****/
128             //initialise mitoticmode random variable, set mitotic mode
129             → appropriately after comparing to mode probability matrix

```

```

126     double mitoticModeRV = p_random_number_generator→ranf(); //0-1
127     ↪ evenly distributed RV
128
129     if (!mDeterministic)
130     {
131         //construct 3×2 matrix of mode probabilities arranged by phase
132         double modeProbabilityMatrix[3][2] = { { mPhase1PP, mPhase1PD },
133             ↪ { mPhase2PP, mPhase2PD }, { mPhase3PP,
134
135             ↪
136             //if the RV is > currentPhasePP && ≤ currentPhasePD, change
137             ↪ mMitoticMode from PP to PD
138             if (mitoticModeRV > modeProbabilityMatrix[currentPhase - 1][0]
139                 && mitoticModeRV
140                     ≤ modeProbabilityMatrix[currentPhase - 1][0] +
141                         ↪ modeProbabilityMatrix[currentPhase - 1][1])
142             {
143                 mMitoticMode = 1;
144                 if (mpCell→HasCellProperty<Ath5Mo>()) //Ath5 morphants
145                     ↪ undergo PP rather than PD divisions in 80% of cases
146                 {
147                     double ath5RV = p_random_number_generator→ranf();
148                     if (ath5RV ≤ .8)
149                     {
150                         mMitoticMode = 0;
151                     }
152                 }
153             }
154         }
155
156         ****
157         * Write mitotic event to relevant files
158         * ****

```

```

159     if (mDebug)
160     {
161         WriteDebugData(currentTiL, currentPhase, mitoticModeRV);
162     }
163
164     if (mOutput)
165     {
166         WriteModeEventOutput();
167     }
168
169     //set new cell cycle length (will be overwritten with DBL_MAX for DD
170     //→ divisions)
171     AbstractSimpleCellCycleModel::ResetForDivision();
172
173     /*****
174     * Symmetric postmitotic specification rule
175     * -(asymmetric postmitotic rule specified in
176     //→ InitialiseDaughterCell());
177     * *****/
178     if (mMitoticMode == 2)
179     {
180         boost::shared_ptr<AbstractCellProperty> p_PostMitoticType =
181
182             //→ mpCell→rGetCellPropertyCollection().GetCellPropertyRegistry()→G
183             mpCell→SetCellProliferativeType(p_PostMitoticType);
184             mCellCycleDuration = DBL_MAX;
185
186             if(mKillSpecified)
187             {
188                 mpCell→Kill();
189             }
190
191             /*****
192             * SEQUENCE SAMPLER
193             *****/
194             //if the sequence sampler has been turned on, check for the label &
195             //→ write mitotic mode to log
196             //50% chance of each daughter cell from a mitosis inheriting the
197             //→ label
198             if (mSequenceSampler)
199             {
200                 if (mpCell→HasCellProperty<CellLabel>())

```

```

197     {
198         (*LogFile::Instance()) << mMitoticMode;
199         double labelRV = p_random_number_generator->ranf();
200         if (labelRV <= .5)
201         {
202             mSeqSamplerLabelSister = true;
203             mpCell->RemoveCellProperty<CellLabel>();
204         }
205         else
206         {
207             mSeqSamplerLabelSister = false;
208         }
209     }
210     else
211     {
212         //prevents lost-label cells from labelling their progeny
213         mSeqSamplerLabelSister = false;
214     }
215 }
216 }

217 void HeCellCycleModel::Initialise()
218 {
219     boost::shared_ptr<AbstractCellProperty> p_Transit =
220
221     → mpCell->rGetCellPropertyCollection().GetCellPropertyRegistry()->Get<T>
222     mpCell->SetCellProliferativeType(p_Transit);
223
224     if (mTiLOffset == 0) //the "regular" case, set cycle duration
225     → normally
226     {
227         SetCellCycleDuration();
228     }
229
230     if (mTiLOffset < 0) //these cells are offspring of Wan stem cells
231     {
232         mReadyToDivide = false;
233         SetCellCycleDuration();
234     }
235
236     if (mTiLOffset > 0.0) //if the TiL is > 0, the first division has
237     → already occurred
238     {

```

```

237     mReadyToDivide = false;
238
239     RandomNumberGenerator* p_random_number_generator =
240         → RandomNumberGenerator::Instance();
241
242     /**
243      * This calculation "runs time forward" by subtracting
244      → appropriately generated cell lengths from TiLOffset
245      * Ultimately c is subtracted from a final cell length
246      → calculation to give the appropriate reduced cycle length
247      */
248
249     double c = mTiLOffset;
250     while (c > 0)
251     {
252         c = c - (mGammaShift +
253             → p_random_number_generator→GammaRandomDeviate(mGammaShape,
254             → mGammaScale));
255     }
256
257     mCellCycleDuration = (mGammaShift +
258         → p_random_number_generator→GammaRandomDeviate(mGammaShape,
259         → mGammaScale))
260         + c;
261
262     }
263
264     void HeCellCycleModel::InitialiseDaughterCell()
265     {
266         RandomNumberGenerator* p_random_number_generator =
267             → RandomNumberGenerator::Instance();
268
269         /**
270          * PD-type division & shifted sister cycle length & boundary
271          → adjustments
272          *****/
273
274         if (mMitoticMode == 1) //RPC becomes specified retinal neuron in
275             → asymmetric PD mitosis
276         {
277             boost::shared_ptr<AbstractCellProperty> p_PostMitoticType =

```

```
269             → mpCell→rGetCellPropertyCollection().GetCellPropertyRegistry()→G
270             mpCell→SetCellProliferativeType(p_PostMitoticType);
271             mCellCycleDuration = DBL_MAX;
272
273         if(mKillSpecified)
274         {
275             mpCell→Kill();
276         }
277     }
278
279     //daughter cell's mCellCycleDuration is copied from parent; modified
280     → by a normally distributed shift if it remains proliferative
281     if (mMitoticMode == 0)
282     {
283         double sisterShift =
284             → p_random_number_generator→NormalRandomDeviate(0,
285             → mSisterShiftWidth); //random variable mean 0 SD 1 by default
286         mCellCycleDuration = std::max(mGammaShift, mCellCycleDuration +
287             → sisterShift); // sister shift respects 4 hour refractory
288             → period
289     }
290
291
292     //deterministic model phase boundary division shift for daughter
293     → cells
294     if (mDeterministic)
295     {
296         //shift phase boundaries to reflect error in "timer" after
297         → division
298         double phaseShift =
299             → p_random_number_generator→NormalRandomDeviate(0,
300             → mPhaseShiftWidth);
301         mMitoticModePhase2 = mMitoticModePhase2 + phaseShift;
302         mMitoticModePhase3 = mMitoticModePhase3 + phaseShift;
303     }
304
305     ****
306     * SEQUENCE SAMPLER
307     ****
308     if (mSequenceSampler)
309     {
310         if (mSeqSamplerLabelSister)
311         {
```

```

302     boost::shared_ptr<AbstractCellProperty> p_label_type =
303
304         → mpCell→rGetCellPropertyCollection().GetCellPropertyRegistry(
305             mpCell→AddCellProperty(p_label_type);
306             mSeqSamplerLabelSister = false;
307         }
308     else
309     {
310         mpCell→RemoveCellProperty<CellLabel>();
311     }
312
313     if (mMitoticMode == 2 && mKillSpecified) mpCell→Kill();
314 }
315
316 void HeCellCycleModel::SetModelParameters(double tiLOffset, double
317     → mitoticModePhase2, double mitoticModePhase3,
318
319     → double phase1PP, double
320         → phase1PD, double phase2PP,
321         → double phase2PD,
322         → double phase3PP, double
323             → phase3PD, double
324             → gammaShift, double
325             → gammaShape,
326             → double gammaScale, double
327                 → sisterShift)
328 {
329     mTiLOffset = tiLOffset;
330     mMitoticModePhase2 = mitoticModePhase2;
331     mMitoticModePhase3 = mitoticModePhase3;
332     mPhase1PP = phase1PP;
333     mPhase1PD = phase1PD;
334     mPhase2PP = phase2PP;
335     mPhase2PD = phase2PD;
336     mPhase3PP = phase3PP;
337     mPhase3PD = phase3PD;
338     mGammaShift = gammaShift;
339     mGammaShape = gammaShape;
340     mGammaScale = gammaScale;
341     mSisterShiftWidth = sisterShift;
342 }
343
344
345

```

```
336 void HeCellCycleModel::SetDeterministicMode(double tiLOffset, double
→ mitoticModePhase2, double mitoticModePhase3,
337
338
339 {
340     mDeterministic = true;
341     mTiLOffset = tiLOffset;
342     mMitoticModePhase2 = mitoticModePhase2;
343     mMitoticModePhase3 = mitoticModePhase3;
344     mPhaseShiftWidth = phaseShiftWidth;
345     mGammaShift = gammaShift;
346     mGammaShape = gammaShape;
347     mGammaScale = gammaScale;
348     mSisterShiftWidth = sisterShift;
349 }
350
351 void HeCellCycleModel::SetTimeDependentCycleDuration(double peakRateTime,
→ double increasingSlope,
352
353 {
354     mTimeDependentCycleDuration = true;
355     mPeakRateTime = peakRateTime;
356     mIncreasingRateSlope = increasingSlope;
357     mDecreasingRateSlope = decreasingSlope;
358     mBaseGammaScale = mGammaScale;
359 }
360
361 void HeCellCycleModel::EnableKillSpecified()
362 {
363     mKillSpecified = true;
364 }
365
366 void HeCellCycleModel::EnableModeEventOutput(double eventStart, unsigned
→ seed)
367 {
368     mOutput = true;
369     mEventStartTime = eventStart;
370     mSeed = seed;
371 }
```

```

372
373 void HeCellCycleModel::WriteModeEventOutput()
374 {
375     double currentTime = SimulationTime::Instance()→GetTime() +
376         → mEventStartTime;
377     CellPtr currentCell = GetCell();
378     double currentCellID = (double) currentCell→GetCellId();
379     (*LogFile::Instance()) << currentTime << "\t" << mSeed << "\t" <<
380         → currentCellID << "\t" << mMitoticMode << "\n";
381 }
382
383 void HeCellCycleModel::EnableSequenceSampler()
384 {
385     mSequenceSampler = true;
386     boost::shared_ptr<AbstractCellProperty> p_label_type =
387         → mpCell→rGetCellPropertyCollection().GetCellPropertyRegistry()→Get<C
388     mpCell→AddCellProperty(p_label_type);
389 }
390
391 void
392     → HeCellCycleModel::PassDebugWriter(boost::shared_ptr<ColumnDataWriter>
393     → debugWriter, int timeID,
394                                         std::vector<int> varIDs)
395 {
396     mDebug = true;
397     mDebugWriter = debugWriter;
398     mTimeID = timeID;
399     mVarIDs = varIDs;
400 }
401
402 void
403     → HeCellCycleModel::EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
404     → debugWriter)
405 {
406     mDebug = true;
407     mDebugWriter = debugWriter;
408
409     mTimeID = mDebugWriter→DefineUnlimitedDimension("Time", "h");
410
411     mVarIDs.push_back(mDebugWriter→DefineVariable("CellID", "No"));
412     mVarIDs.push_back(mDebugWriter→DefineVariable("TiL", "h"));

```

```

407     mVarIDs.push_back(mDebugWriter->DefineVariable("CycleDuration",
408         ↵ "h"));
409     mVarIDs.push_back(mDebugWriter->DefineVariable("Phase2Boundary",
410         ↵ "h"));
411     mVarIDs.push_back(mDebugWriter->DefineVariable("Phase3Boundary",
412         ↵ "h"));
413     mVarIDs.push_back(mDebugWriter->DefineVariable("Phase", "No"));
414     mVarIDs.push_back(mDebugWriter->DefineVariable("MitoticModeRV",
415         ↵ "Percentile"));
416     mVarIDs.push_back(mDebugWriter->DefineVariable("MitoticMode",
417         ↵ "Mode"));
418     mVarIDs.push_back(mDebugWriter->DefineVariable("Label", "binary"));

419 }
420
421 void HeCellCycleModel::WriteDebugData(double currentTiL, unsigned phase,
422                                     ↵ double mitoticModeRV)
423 {
424     double currentTime = SimulationTime::Instance()->GetTime();
425     double currentCellID = mpCell->GetCellId();
426     unsigned label = 0;
427     if (mpCell->HasCellProperty<CellLabel>()) label = 1;
428
429     mDebugWriter->PutVariable(mTimeID, currentTime);
430     mDebugWriter->PutVariable(mVarIDs[0], currentCellID);
431     mDebugWriter->PutVariable(mVarIDs[1], currentTiL);
432     mDebugWriter->PutVariable(mVarIDs[2], mCellCycleDuration);
433     mDebugWriter->PutVariable(mVarIDs[3], mMitoticModePhase2);
434     mDebugWriter->PutVariable(mVarIDs[4], mMitoticModePhase3);
435     mDebugWriter->PutVariable(mVarIDs[5], phase);
436     if (!mDeterministic)
437     {
438         mDebugWriter->PutVariable(mVarIDs[6], mitoticModeRV);
439     }
440     mDebugWriter->PutVariable(mVarIDs[7], mMitoticMode);
441     if (mSequenceSampler)
442     {
443         mDebugWriter->PutVariable(mVarIDs[8], label);
444     }
445     mDebugWriter->AdvanceAlongUnlimitedDimension();
446 }
447

```

```

444 /*****
445 * UNUSED FUNCTIONS (required for class nonvirtuality, do not remove)
446 *****/
447
448 double HeCellCycleModel::GetAverageTransitCellCycleTime()
449 {
450     return (0.0);
451 }
452
453 double HeCellCycleModel::GetAverageStemCellCycleTime()
454 {
455     return (0.0);
456 }
457
458 void HeCellCycleModel::OutputCellCycleModelParameters(out_stream&
459             rParamsFile)
460 {
461     *rParamsFile << "\t\t\t<CellCycleDuration>" << mCellCycleDuration <<
462     "</CellCycleDuration>\n";
463
464     // Call method on direct parent class
465
466     ↵ AbstractSimpleCellCycleModel::OutputCellCycleModelParameters(rParamsFile);
467
468 // Serialization for Boost ≥ 1.36
469 #include "SerializationExportWrapperForCpp.hpp"
470 CHASTE_CLASS_EXPORT(HeCellCycleModel)

```

---

### 17.1.30 /src/HeCellCycleModel.hpp

---

```

1 #ifndef HECELLCYCLEMODEL_HPP_
2 #define HECELLCYCLEMODEL_HPP_
3
4 #include "AbstractSimpleCellCycleModel.hpp"
5 #include "RandomNumberGenerator.hpp"
6 #include "Cell.hpp"
7 #include "TransitCellProliferativeType.hpp"
8 #include "DifferentiatedCellProliferativeType.hpp"
9 #include "SmartPointers.hpp"
10 #include "ColumnDataWriter.hpp"
11 #include "LogFile.hpp"

```

```

12 #include "CellLabel.hpp"
13 #include "HeAth5Mo.hpp"

14 /*****
15  * HE CELL CYCLE MODEL
16  * As described in He et al. 2012 [He2012] doi:
17  *   → 10.1016/j.neuron.2012.06.033
18  *
19  * USE: By default, HeCellCycleModels are constructed with the parameter
20  *   fit reported in [He2012].
21  * In normal use, the model steps through three phases of mitotic mode
22  *   probability parameterisation.
23  * PP = symmetric proliferative mitotic mode, both progeny remain mitotic
24  * PD = asymmetric proliferative mitotic mode, one progeny exits the cell
25  *   cycle and differentiates
26  * DD = symmetric differentiative mitotic mode, both progeny exit the
27  *   cell cycle and differentiate
28  *
29  * Change default model parameters with SetModelParameters(<params>);
30  * Enable deterministic model alternative with
31  *   → EnableDeterministicMode(<params>);
32  *
33  * 2 per-model-event output modes:
34  * EnableModeEventOutput() enables mitotic mode event logging-all cells
35  *   → will write to the singleton log file
36  * EnableModelDebugOutput() enables more detailed debug output, each seed
37  *   → will have its own file written to
38  * by a ColumnDataWriter passed to it from the test
39  * (eg. by the SetupDebugOutput helper function in the project simulator)
40  *
41  * 1 mitotic-event-sequence sampler (only samples one "path" through the
42  *   lineage):
43  * EnableSequenceSampler() - one "sequence" of progenitors writes mitotic
44  *   event type to a string in the singleton log file
45  */
46
47 class HeCellCycleModel : public AbstractSimpleCellCycleModel
48 {
49     friend class TestSimpleCellCycleModels;
50
51 private:
52
53

```

```

45  /** Needed for serialization.*/
46  friend class boost::serialization::access;
47  /**
48   * Archive the cell-cycle model and random number generator, never
49   * used directly - boost uses this.
50   *
51   * @param archive the archive
52   * @param version the current version of this class
53   */
54  template<class Archive>
55  void serialize(Archive & archive, const unsigned int version)
56  {
57      archive &
58          boost::serialization::base_object<AbstractSimpleCellCycleModel>(*this);
59
60      SerializableSingleton<RandomNumberGenerator>* p_wrapper =
61
62          RandomNumberGenerator::Instance()→GetSerializationWrapper();
63
64      archive & p_wrapper;
65      archive & mCellCycleDuration;
66
67
68  protected:
69      //mode/output variables
70      bool mKillSpecified;
71      bool mDeterministic;
72      bool mOutput;
73      double mEventStartTime;
74      bool mSequenceSampler;
75      bool mSeqSamplerLabelSister;
76      //debug writer stuff
77      bool mDebug;
78      int mTimeID;
79      std::vector<int> mVarIDs;
80      boost::shared_ptr<ColumnDataWriter> mDebugWriter;
81      //model parameters and state memory vars
82      double mTiLOffset;
83      double mGammaShift;

```

```

84     double mGammaShape;
85     double mGammaScale;
86     double mSisterShiftWidth;
87     double mMitoticModePhase2;
88     double mMitoticModePhase3;
89     double mPhaseShiftWidth;
90     double mPhase1PP;
91     double mPhase1PD;
92     double mPhase2PP;
93     double mPhase2PD;
94     double mPhase3PP;
95     double mPhase3PD;
96     unsigned mMitoticMode;
97     unsigned mSeed;
98     bool mTimeDependentCycleDuration;
99     double mPeakRateTime;
100    double mIncreasingRateSlope;
101    double mDecreasingRateSlope;
102    double mBaseGammaScale;

103
104 /**
105  * Protected copy-constructor for use by CreateCellCycleModel().
106  *
107  * The only way for external code to create a copy of a cell cycle
108  * model
109  * is by calling that method, to ensure that a model of the correct
110  * subclass is created.
111  * This copy-constructor helps subclasses to ensure that all member
112  * variables are correctly copied when this happens.
113  *
114  * This method is called by child classes to set member variables for
115  * a daughter cell upon cell division.
116  * Note that the parent cell cycle model will have had
117  * ResetForDivision() called just before CreateCellCycleModel() is
118  * called,
119  * so performing an exact copy of the parent is suitable behaviour.
120  * Any daughter-cell-specific initialisation
121  * can be done in InitialiseDaughterCell().
122  *
123  * @param rModel the cell cycle model to copy.
124  */
125 HeCellCycleModel(const HeCellCycleModel& rModel);
126
127
128
129

```

```
120 public:  
121  
122     /**  
123      * Constructor - just a default, mBirthTime is set in the  
124      ← AbstractCellCycleModel class.  
125      */  
126     HeCellCycleModel();  
127  
128     /**  
129      * SetCellCycleDuration() method to set length of cell cycle  
130      */  
131     void SetCellCycleDuration();  
132  
133     /**  
134      * Overridden builder method to create new copies of  
135      * this cell-cycle model.  
136      *  
137      * @return new cell-cycle model  
138      */  
139     AbstractCellCycleModel* CreateCellCycleModel();  
140  
141     /**  
142      * Overridden ResetForDivision() method.  
143      * Contains general mitotic mode logic  
144      ***/  
145     void ResetForDivision();  
146  
147     /**  
148      * Overridden Initialise() method  
149      * Used to give an appropriate mCellCycleDuration to cells w/ TiL  
150      ← offsets  
151      * sets mReadytoDivide to false as appropriate  
152      * Initialises as transit proliferative type  
153      ***/  
154     void Initialise();  
155  
156     /**  
157      * Overridden InitialiseDaughterCell() method.  
158      * Used to apply sister-cell time shifting (cell cycle duration,  
159      ← deterministic phase boundaries)  
159      * Used to implement asymmetric mitotic mode  
160      * */  
161     void InitialiseDaughterCell();
```

```

160
161     /*Model setup functions for standard He (SetModelParameters) and
162      → deterministic alternative (SetDeterministicMode) models
163      * Default parameters are from refits of He et al + deterministic
164      → alternatives
165      * He 2012 params: mitoticModePhase2 = 8, mitoticModePhase3 = 15,
166      → p1PP = 1, p1PD = 0, p2PP = .2, p2PD = .4, p3PP = .2, p3PD = 0
167      * gammaShift = 4, gammaShape = 2, gammaScale = 1, sisterShift = 1
168      */
169
170     void SetModelParameters(double tiLOffset = 0, double
171         → mitoticModePhase2 = 8, double mitoticModePhase3 = 15,
172             → double phase1PP = 1, double phase1PD = 0,
173                 → double phase2PP = .2, double phase2PD =
174                     → .4,
175                     double phase3PP = .2, double phase3PD = 0,
176                         → double gammaShift = 4, double gammaShape
177                             → = 2,
178                             double gammaScale = 1, double sisterShift =
179                                 → 1);
180
181     void SetDeterministicMode(double tiLOffset = 0, double
182         → mitoticModePhase2 = 8, double mitoticModePhase3 = 15,
183             → double phaseShiftWidth = 1, double
184                 → gammaShift = 4, double gammaShape = 2,
185                 double gammaScale = 1, double sisterShift =
186                     → 1);
187
188     void SetTimeDependentCycleDuration(double peakRateTime, double
189         → increasingSlope, double decreasingSlope);
190
191
192     //Function to set mKillSpecified = true; marks specified neurons for
193     → death and removal from population
194     //Intended to help w/ resource consumption for WanSimulator
195     void EnableKillSpecified();
196
197
198     //Functions to enable per-cell mitotic mode logging for mode rate &
199     → sequence sampling fixtures
200     //Uses singleton logfile
201     void EnableModeEventOutput(double eventStart, unsigned seed);
202     void EnableSequenceSampler();
203
204
205     //More detailed debug output. Needs a ColumnDataWriter passed to it
206     //Only declare ColumnDataWriter directory, filename, etc; do not set
207     → up otherwise

```

```
186 //Use PassDebugWriter if the writer is already enabled elsewhere (ie.  
187     ↵  in a Wan stem cell cycle model)  
188 void EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>  
189     ↵  debugWriter);  
190 void PassDebugWriter(boost::shared_ptr<ColumnDataWriter> debugWriter,  
191     ↵  int timeID, std::vector<int> varIDs);  
192  
193 //Not used, but must be overwritten lest HeCellCycleModels be  
194     ↵  abstract  
195 double GetAverageTransitCellCycleTime();  
196 double GetAverageStemCellCycleTime();  
197  
198 /**  
199     * Overridden OutputCellCycleModelParameters() method.  
200     *  
201     * @param rParamsFile the file stream to which the parameters are  
202     ↵  output  
203     */  
204 virtual void OutputCellCycleModelParameters(out_stream& rParamsFile);  
205 };  
206  
207 #include "SerializationExportWrapper.hpp"  
208 // Declare identifier for the serializer  
209 CHASTE_CLASS_EXPORT(HeCellCycleModel)  
210  
211 #endif /*HECELLCYCLEMODEL_HPP_*/
```

### 17.1.31 /src/OffLatticeSimulationPropertyStop.cpp

```
1 #include "OffLatticeSimulationPropertyStop.hpp"
2
3 #include <boost/make_shared.hpp>
4
5 #include "CellBasedEventHandler.hpp"
6 #include "ForwardEulerNumericalMethod.hpp"
7 #include "StepSizeException.hpp"
8
9 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
10 OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::OffLatticeSimulationProperty
   ↵ rCellPopulation,
11
   ↵ bool
   ↵   deleteCellPopulationInDestructor,
```

```

12                     bool initialiseCells
13                         )
14 : AbstractCellBasedSimulation<ELEMENT_DIM,SPACE_DIM>(rCellPopulation,
15   ~ deleteCellPopulationInDestructor, initialiseCells),
16 p_property()
17 {
18     if
19       ~ (!dynamic_cast<AbstractOffLatticeCellPopulation<ELEMENT_DIM,SPACE_DIM>*>(&rCe
20     {
21         EXCEPTION("OffLatticeSimulations require a subclass of
22           ~ AbstractOffLatticeCellPopulation.");
23     }
24 }
25
26 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
27 bool
28   ~ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::StoppingEventHasOccurred
29 {
30     if(p_property->GetCellCount()<1){
31         return true;
32     }
33     else{
34         return false;
35     }
36 }
37
38 //Public access to Stopping Event bool
39 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
40 bool
41   ~ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::HasStoppingEventOccurred
42 {
43     return StoppingEventHasOccurred();
44 }
45
46 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
47 void
48   ~ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::SetStopProperty(boost::s
49   ~ stopPropertySetting)
50 {
51     p_property = stopPropertySetting;
52 }
53
54
55 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>

```

```
48 void
  ↵ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::AddForce(boost::shared_p
  ↵ > pForce)
49 {
50     mForceCollection.push_back(pForce);
51 }
52
53 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
54 void
  ↵ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::RemoveAllForces()
55 {
56     mForceCollection.clear();
57 }
58
59 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
60 void
  ↵ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::AddCellPopulationBoundary
  ↵ > pBoundaryCondition)
61 {
62     mBoundaryConditions.push_back(pBoundaryCondition);
63 }
64
65 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
66 void
  ↵ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::RemoveAllCellPopulationBou
67 {
68     mBoundaryConditions.clear();
69 }
70
71 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
72 void
  ↵ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::SetNumericalMethod(boost
  ↵ > SPACE_DIM> > pNumericalMethod)
73 {
74     mpNumericalMethod = pNumericalMethod;
75 }
76
77 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
78 const boost::shared_ptr<AbstractNumericalMethod<ELEMENT_DIM, SPACE_DIM> >
  ↵ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::GetNumericalMethod()
  ↵ const
79 {
80     return mpNumericalMethod;
```

```

81  }
82
83 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
84 const std::vector<boost::shared_ptr<AbstractForce<ELEMENT_DIM, SPACE_DIM>
85   > >&
86   OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::rGetForceCollection()
87   const
88 {
89     return mForceCollection;
90 }
91
92
93 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
94 void
95   OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::UpdateCellLocationsAndTo
96 {
97   CellBasedEventHandler::BeginEvent(CellBasedEventHandler::POSITION);
98
99   double time_advanced_so_far = 0;
100  double target_time_step = this->mDt;
101  double present_time_step = this->mDt;
102
103  while (time_advanced_so_far < target_time_step)
104  {
105      // Store the initial node positions (these may be needed when
106      // applying boundary conditions)
107      std::map<Node<SPACE_DIM>*, c_vector<double, SPACE_DIM> >
108      old_node_locations;
109
110      for (typename AbstractMesh<ELEMENT_DIM, SPACE_DIM>::NodeIterator
111        node_iter =
112        this->mrCellPopulation.rGetMesh().GetNodeIteratorBegin();
113        node_iter !=
114        this->mrCellPopulation.rGetMesh().GetNodeIteratorEnd();
115        ++node_iter)
116      {
117          old_node_locations[&(*node_iter)] =
118          (node_iter)->rGetLocation();
119      }
120
121      // Try to update node positions according to the numerical method
122      try
123      {
124          mpNumericalMethod->UpdateAllNodePositions(present_time_step);

```

```

114         ApplyBoundaries(old_node_locations);
115
116         // Successful time step! Update time_advanced_so_far
117         time_advanced_so_far += present_time_step;
118
119         // If using adaptive timestep, then increase the
120         // → present_time_step (by 1% for now)
121         if (mpNumericalMethod→HasAdaptiveTimestep())
122         {
123             ///\todo #2087 Make this a settable member variable
124             double timestep_increase = 0.01;
125             present_time_step =
126                 → std::min((1+timestep_increase)*present_time_step,
127                 → target_time_step - time_advanced_so_far);
128         }
129     }
130
131     catch (StepSizeException& e)
132     {
133         // Detects if a node has travelled too far in a single time
134         // → step
135         if (mpNumericalMethod→HasAdaptiveTimestep())
136         {
137             // If adaptivity is switched on, revert node locations
138             // → and choose a suitably smaller time step
139             RevertToOldLocations(old_node_locations);
140             present_time_step = std::min(e.GetSuggestedNewStep(),
141                 → target_time_step - time_advanced_so_far);
142         }
143     }
144
145     CellBasedEventHandler::EndEvent(CellBasedEventHandler::POSITION);
146 }
147
148 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>

```

```

149 void
150   → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::RevertToOldLocations(std
151   → c_vector<double, SPACE_DIM> > oldNodeLoctions)
150 {
151   for (typename AbstractMesh<ELEMENT_DIM, SPACE_DIM>::NodeIterator
152     → node_iter =
153     → this→mrCellPopulation.rGetMesh().GetNodeIteratorBegin();
152     node_iter ≠
153     → this→mrCellPopulation.rGetMesh().GetNodeIteratorEnd();
153     ++node_iter)
154   {
155     (node_iter)→rGetModifiableLocation() =
156     → oldNodeLoctions[&(*node_iter)];
156   }
157 }

158
159 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
160 void
161   → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::ApplyBoundaries(std::map
162   → SPACE_DIM> > oldNodeLoctions)
161 {
162   // Apply any boundary conditions
163   for (typename
164     → std::vector<boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT
165     → >>::iterator bcs_iter = mBoundaryConditions.begin();
164     bcs_iter ≠ mBoundaryConditions.end();
165     ++bcs_iter)
166   {
167     (*bcs_iter)→ImposeBoundaryCondition(oldNodeLoctions);
168   }

169
170   // Verify that each boundary condition is now satisfied
171   for (typename
172     → std::vector<boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT
173     → >>::iterator bcs_iter = mBoundaryConditions.begin();
172     bcs_iter ≠ mBoundaryConditions.end();
173     ++bcs_iter)
174   {
175     if (!((*bcs_iter)→VerifyBoundaryCondition()))
176     {
177       EXCEPTION("The cell population boundary conditions are
178       → incompatible.");
178     }

```

```

179     }
180 }
181
182 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
183 void
184      $\hookrightarrow$  OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::WriteVisualizerSetupFile
185 {
186     if (PetscTools::AmMaster())
187     {
188         for (unsigned i=0; i<this $\rightarrow$ mForceCollection.size(); i++)
189         {
190              $\hookrightarrow$  this $\rightarrow$ mForceCollection[i] $\rightarrow$ WriteDataToVisualizerSetupFile(this $\rightarrow$ mpViz
191         }
192
193          $\hookrightarrow$  this $\rightarrow$ mrCellPopulation.WriteDataToVisualizerSetupFile(this $\rightarrow$ mpVizSetupFile
194     }
195
196 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
197 void
198      $\hookrightarrow$  OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::SetupSolve()
199 {
200     // Clear all forces
201     for (typename AbstractMesh<ELEMENT_DIM, SPACE_DIM>::NodeIterator
202         node_iter =
203          $\hookrightarrow$  this $\rightarrow$ mrCellPopulation.rGetMesh().GetNodeIteratorBegin();
204         node_iter !=
205          $\hookrightarrow$  this $\rightarrow$ mrCellPopulation.rGetMesh().GetNodeIteratorEnd();
206         ++node_iter)
207     {
208         node_iter $\rightarrow$ ClearAppliedForce();
209     }
210
211     // Use a forward Euler method by default, unless a numerical method
212      $\hookrightarrow$  has been specified already
213     if (mpNumericalMethod == nullptr)
214     {
215         mpNumericalMethod =
216             boost::make_shared<ForwardEulerNumericalMethod<ELEMENT_DIM,
217             SPACE_DIM> >();
218     }

```

```
212     → mpNumericalMethod→SetCellPopulation(dynamic_cast<AbstractOffLatticeCellPopul
213     mpNumericalMethod→SetForceCollection(&mForceCollection);
214 }
215
216 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
217 void
218 {
219     // Loop over forces
220     *rParamsFile << "\n\t<Forces>\n";
221     for (typename
222         → std::vector<boost::shared_ptr<AbstractForce<ELEMENT_DIM,SPACE_DIM>
223         >>::iterator iter = mForceCollection.begin();
224         iter ≠ mForceCollection.end();
225         ++iter)
226     {
227         // Output force details
228         (*iter)→OutputForceInfo(rParamsFile);
229     }
230     *rParamsFile << "\t</Forces>\n";
231
232     // Loop over cell population boundary conditions
233     *rParamsFile << "\n\t<CellPopulationBoundaryConditions>\n";
234     for (typename
235         → std::vector<boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT_
236         >>::iterator iter = mBoundaryConditions.begin();
237         iter ≠ mBoundaryConditions.end();
238         ++iter)
239     {
240         // Output cell boundary condition details
241         (*iter)→OutputCellPopulationBoundaryConditionInfo(rParamsFile);
242     }
243     *rParamsFile << "\t</CellPopulationBoundaryConditions>\n";
244
245     // Output numerical method details
246     *rParamsFile << "\n\t<NumericalMethod>\n";
247     mpNumericalMethod→OutputNumericalMethodInfo(rParamsFile);
248     *rParamsFile << "\t</NumericalMethod>\n";
249 }
250
251
252 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
```

```

248 void
249   → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::OutputSimulationParameter
250   → rParamsFile)
251 {
252     // No new parameters to output, so just call method on direct parent
253     → class
254
255     → AbstractCellBasedSimulation<ELEMENT_DIM,SPACE_DIM>::OutputSimulationParameter
256 }
257
258 // Explicit instantiation
259 template class OffLatticeSimulationPropertyStop<1,1>;
260 template class OffLatticeSimulationPropertyStop<1,2>;
261 template class OffLatticeSimulationPropertyStop<2,2>;
262 template class OffLatticeSimulationPropertyStop<1,3>;
263 template class OffLatticeSimulationPropertyStop<2,3>;
264 template class OffLatticeSimulationPropertyStop<3,3>;
265
266 // Serialization for Boost ≥ 1.36
267 #include "SerializationExportWrapperForCpp.hpp"
268 EXPORT_TEMPLATE_CLASS_ALL_DIMS(OffLatticeSimulationPropertyStop)

```

---

### 17.1.32 /src/OffLatticeSimulationPropertyStop.hpp

```

1 #ifndef OFFLATTICESIMULATIONPROPERTYSTOP_HPP_
2 #define OFFLATTICESIMULATIONPROPERTYSTOP_HPP_
3
4 #include "AbstractCellBasedSimulation.hpp"
5 #include "AbstractForce.hpp"
6 #include "AbstractCellPopulationBoundaryCondition.hpp"
7 #include "AbstractNumericalMethod.hpp"
8
9 #include "ChasteSerialization.hpp"
10 #include <boost/serialization/base_object.hpp>
11 #include <boost/serialization/set.hpp>
12 #include <boost/serialization/vector.hpp>
13
14 /**
15  * Run an off-lattice 2D or 3D cell-based simulation using an off-lattice
16  * cell population.
17  *
18  * In cell-centre-based cell populations, each cell is represented by a

```

```

19 * single node (corresponding to its centre), and connectivity is defined
20 * either by a Delaunay triangulation or a radius of influence. In
21 * vertex-
22 * based cell populations, each cell is represented by a polytope
23 * (corresponding to its membrane) with a variable number of vertices.
24 * Alternative cell populations may be defined by the user.
25 *
26 * The OffLatticeSimulation is constructed with a CellPopulation, which
27 * updates the correspondence between each Cell and its spatial
28 * representation
29 * and handles cell division (governed by the CellCycleModel associated
30 * with each cell). Once constructed, one or more Force laws may be
31 * passed
32 * to the OffLatticeSimulation object, to define the mechanical
33 * properties
34 * of the CellPopulation. Similarly, one or more CellKillers may be
35 * passed
36 * to the OffLatticeSimulation object to specify conditions in which
37 * Cells
38 * may die, and one or more CellPopulationBoundaryConditions to specify
39 * regions in space beyond which Cells may not move.
40 */
41 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM = ELEMENT_DIM>
42 class OffLatticeSimulationPropertyStop : public
43     AbstractCellBasedSimulation<ELEMENT_DIM, SPACE_DIM>
44 {
45     private:
46         /** Needed for serialization.*/
47         friend class boost::serialization::access;
48         friend class TestOffLatticeSimulation;
49         friend class TestOffLatticeSimulationWithNodeBasedCellPopulation;
50
51         /**
52             * Save or restore the simulation.
53             *
54             * @param archive the archive
55             * @param version the current version of this class
56             */
57         template<class Archive>
58         void serialize(Archive & archive, const unsigned int version)
59     {

```

```

54     archive &
55     → boost::serialization::base_object<AbstractCellBasedSimulation<ELEMENT_DIM,
56     → SPACE_DIM>>(*this);
56     archive & mForceCollection;
57     archive & mBoundaryConditions;
57     archive & mpNumericalMethod;
58   }
59
60 protected:
61   boost::shared_ptr<AbstractCellProperty> p_property;
62   /** The mechanics used to determine the new location of the cells, a
63   → list of the forces. */
63   std::vector<boost::shared_ptr<AbstractForce<ELEMENT_DIM, SPACE_DIM>>
64   → > mForceCollection;
64
65   /** List of boundary conditions. */
66
66   → std::vector<boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT_
67   → SPACE_DIM>>> mBoundaryConditions;
67
68   /** The numerical method to use in this simulation. Defaults to the
69   → explicit forward Euler method. */
69   boost::shared_ptr<AbstractNumericalMethod<ELEMENT_DIM, SPACE_DIM>>
70   → mpNumericalMethod;
70
71   /**
72    * Overridden UpdateCellLocationsAndTopology() method.
73    *
74    * Calculate forces and update node positions.
75    */
75
76   virtual void UpdateCellLocationsAndTopology();
77
77   /**
78    * Sends nodes back to the positions given in the input map. Used
79    → after a failed step
80    * when adaptivity is turned on.
81    *
82    * @param oldNodeLoctions A map linking nodes to their old positions.
83    */
83
84   void RevertToOldLocations(std::map<Node<SPACE_DIM>*, c_vector<double,
85   → SPACE_DIM>> oldNodeLoctions);
85
86   /**

```

```

87     * Applies any boundary conditions.
88
89     *
90     * @param oldNodeLoctions Mapping between node indices and old node
91     → locations
92     */
93
94     void ApplyBoundaries(std::map<Node<SPACE_DIM>*, c_vector<double,
95     → SPACE_DIM> > oldNodeLoctions);
96
97
98     /**
99      * Overridden SetupSolve() method to clear the forces applied to the
100     → nodes.
101     */
102
103     virtual void SetupSolve();
104
105     /**
106      * Overridden WriteVisualizerSetupFile() method.
107      */
108
109     virtual void WriteVisualizerSetupFile();
110
111     bool StoppingEventHasOccurred();
112
113 public:
114
115     /**
116      * Constructor.
117      *
118      * @param rCellPopulation Reference to a cell population object
119      * @param deleteCellPopulationInDestructor Whether to delete the cell
120      → population on destruction to
121      *      free up memory (defaults to false)
122      * @param initialiseCells Whether to initialise cells (defaults to
123      → true, set to false when loading
124      *      from an archive)
125      */
126
127     OffLatticeSimulationPropertyStop(AbstractCellPopulation<ELEMENT_DIM,
128     → SPACE_DIM>& rCellPopulation,
129
130                                         bool
131                                         → deleteCellPopulationInDestructor
132                                         → = false, bool initialiseCells =
133                                         → true);
134
135
136     void SetStopProperty(boost::shared_ptr<AbstractCellProperty>
137     → stopPropertySetting);

```

```
120
121     bool HasStoppingEventOccurred();
122
123     /**
124      * Add a force to be used in this simulation (use this to set the
125      * mechanics system).
126      *
127      * @param pForce pointer to a force law
128      */
129
130     /**
131      * Remove all the forces.
132      */
133     void RemoveAllForces();
134
135     /**
136      * Add a cell population boundary condition to be used in this
137      * simulation.
138      *
139      * @param pBoundaryCondition pointer to a boundary condition
140      */
141     void AddCellPopulationBoundaryCondition(
142
143         boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT_DIM,
144         SPACE_DIM> > pBoundaryCondition);
145
146     /**
147      * Method to remove all the cell population boundary conditions
148      */
149     void RemoveAllCellPopulationBoundaryConditions();
150
151     /**
152      * Set the numerical method to be used in this simulation (use this
153      * to solve the mechanics system).
154      *
155      * @param pNumericalMethod pointer to a numerical method object
156      */
157     void
158         SetNumericalMethod(boost::shared_ptr<AbstractNumericalMethod<ELEMENT_DIM,
159         SPACE_DIM> > pNumericalMethod);
```

```
155  /**
156   * @return the current numerical method.
157   */
158  const boost::shared_ptr<AbstractNumericalMethod<ELEMENT_DIM,
159    → SPACE_DIM> > GetNumericalMethod() const;
160
161  /**
162   * Overridden OutputAdditionalSimulationSetup() method.
163   *
164   * Output any force, boundary condition or numerical method
165   * information.
166   *
167   * @param rParamsFile the file stream to which the parameters are
168   * output
169   */
170  void OutputAdditionalSimulationSetup(out_stream& rParamsFile);
171
172  /**
173   * Overridden OutputSimulationParameters() method.
174   *
175   * @param rParamsFile the file stream to which the parameters are
176   * output
177   */
178  virtual void OutputSimulationParameters(out_stream& rParamsFile);
179
180  /**
181   * Directly access the forces attached to this simulation, to allow
182   * their manipulation after archiving.
183   *
184   * @return mForceCollection the vector of pointers to forces attached
185   * to this simulation
186   */
187  const std::vector<boost::shared_ptr<AbstractForce<ELEMENT_DIM,
188    → SPACE_DIM> > >& rGetForceCollection() const;
189};

190 // Serialization for Boost ≥ 1.36
191 #include "SerializationExportWrapper.hpp"
192 EXPORT_TEMPLATE_CLASS_ALL_DIMS(OffLatticeSimulationPropertyStop)
193
194 namespace boost
195 {
196 namespace serialization
```

```

191 {
192 /**
193 * Serialize information required to construct an OffLatticeSimulation.
194 */
195 template<class Archive, unsigned ELEMENT_DIM, unsigned SPACE_DIM>
196 inline void save_construct_data(Archive & ar, const
197   → OffLatticeSimulationPropertyStop<ELEMENT_DIM, SPACE_DIM> * t,
198   → const unsigned int file_version)
199 {
200     // Save data required to construct instance
201     const AbstractCellPopulation<ELEMENT_DIM, SPACE_DIM>*
202       → p_cell_population = &(t→rGetCellPopulation());
203     ar & p_cell_population;
204 }
205 /**
206 * De-serialize constructor parameters and initialise an
207   → OffLatticeSimulation.
208 */
209 template<class Archive, unsigned ELEMENT_DIM, unsigned SPACE_DIM>
210 inline void load_construct_data(Archive & ar,
211   → OffLatticeSimulationPropertyStop<ELEMENT_DIM, SPACE_DIM> * t,
212   → const unsigned int file_version)
213 {
214   // Retrieve data from archive required to construct new instance
215   AbstractCellPopulation<ELEMENT_DIM, SPACE_DIM>* p_cell_population;
216   ar >> p_cell_population;
217
218   // Invoke inplace constructor to initialise instance, middle two
219   // variables set extra
220   // member variables to be deleted as they are loaded from archive and
221   // to not initialise cells.
222   ::new (t) OffLatticeSimulationPropertyStop<ELEMENT_DIM,
223   → SPACE_DIM>(*p_cell_population, true, false);
224 }
225 }
226 } // namespace
227
228 #endif /*OFFLATTICESIMULATIONPROPERTYSTOP_HPP_*/

```

---

### 17.1.33 /src/WanStemCellCycleModel.cpp

---

```

1 #include "WanStemCellCycleModel.hpp"
2
3 WanStemCellCycleModel::WanStemCellCycleModel() :
4     AbstractSimpleCellCycleModel(), mExpandingStemPopulation(false),
5     mPopulation(), mOutput(false), mEventStartTime(
6         72.0), mDebug(false), mTimeID(), mVarIDs(),
7         mDebugWriter(), mBasePopulation(), mGammaShift(4.0),
8         mGammaShape(
9             2.0), mGammaScale(1.0), mMitoticMode(0), mSeed(0),
10        mTimeDependentCycleDuration(false), mPeakRateTime(),
11        mIncreasingRateSlope(), mDecreasingRateSlope(),
12        mBaseGammaScale(), mHeParamVector(
13            { 8, 15, 1, 0, .2, .4, .2, 0, 4, 2, 1, 1 }))
14    {
15    }
16
17 WanStemCellCycleModel::WanStemCellCycleModel(const WanStemCellCycleModel&
18     rModel) :
19     AbstractSimpleCellCycleModel(rModel),
20     mExpandingStemPopulation(rModel.mExpandingStemPopulation),
21     mPopulation(
22         rModel.mPopulation), mOutput(rModel.mOutput),
23         mEventStartTime(rModel.mEventStartTime), mDebug(
24         rModel.mDebug), mTimeID(rModel.mTimeID),
25         mVarIDs(rModel.mVarIDs),
26         mDebugWriter(rModel.mDebugWriter), mBasePopulation(
27         rModel.mBasePopulation), mGammaShift(rModel.mGammaShift),
28         mGammaShape(rModel.mGammaShape), mGammaScale(
29         rModel.mGammaScale), mMitoticMode(rModel.mMitoticMode),
30         mSeed(rModel.mSeed), mTimeDependentCycleDuration(
31         rModel.mTimeDependentCycleDuration),
32         mPeakRateTime(rModel.mPeakRateTime),
33         mIncreasingRateSlope(
34             rModel.mIncreasingRateSlope),
35         mDecreasingRateSlope(rModel.mDecreasingRateSlope),
36         mBaseGammaScale(
37             rModel.mBaseGammaScale),
38         mHeParamVector(rModel.mHeParamVector)
39    {
40    }
41
42

```

```

23 AbstractCellCycleModel* WanStemCellCycleModel::CreateCellCycleModel()
24 {
25     return new WanStemCellCycleModel(*this);
26 }
27
28 void WanStemCellCycleModel::SetCellCycleDuration()
29 {
30     RandomNumberGenerator* p_random_number_generator =
31         RandomNumberGenerator::Instance();
32
33     mCellCycleDuration = mGammaShift +
34         p_random_number_generator->GammaRandomDeviate(mGammaShape,
35             mGammaScale);
36
37     /*****
38     * CELL CYCLE DURATION RANDOM VARIABLE
39     *****/
40
41     //Wan stem cell cycle length determined by the same formula as He
42     //RPCS
43     /*
44         if (!mTimeDependentCycleDuration) //Normal operation, cell cycle
45         length stays constant
46         {
47             //He cell cycle length determined by shifted gamma distribution
48             //reflecting 4 hr refractory period followed by gamma pdf
49             mCellCycleDuration = mGammaShift +
50             p_random_number_generator->GammaRandomDeviate(mGammaShape,
51                 mGammaScale);
52         }/*
53
54     ****
55     * Variable cycle length
56     * Give -ve mIncreasingRateSlope and +ve mDecreasingRateSlope,
57     * cell cycle length linearly declines (increasing rate), then
58     // increases, switching at mPeakRateTime
59     ****/
60     /*
61     else
62     {
63         double currTime = SimulationTime::Instance()->GetTime();
64         if (currTime <= mPeakRateTime)
65         {

```

```

56     mGammaScale = std::max((mBaseGammaScale - currTime *
57     ↳ mIncreasingRateSlope), .000000000001);
58     }
59     if (currTime > mPeakRateTime)
60     {
61         mGammaScale = std::max(((mBaseGammaScale - mPeakRateTime *
62         ↳ mIncreasingRateSlope)
63         + (mBaseGammaScale + (currTime - mPeakRateTime) *
64         ↳ mDecreasingRateSlope)),.000000000001);
65     }
66     Timer::Print("mGammaScale: " + std::to_string(mGammaScale));
67     mCellCycleDuration = mGammaShift +
68     ↳ p_random_number_generator→GammaRandomDeviate(mGammaShape,
69     ↳ mGammaScale);
70     }
71     */
72 }
73
74 void WanStemCellCycleModel::ResetForDivision()
75 {
76     mMitoticMode = 1; //by default, asymmetric division giving rise to He
77     ↳ cell (mode 1)
78
79     if (mExpandingStemPopulation)
80     {
81         double currRetinaAge = SimulationTime::Instance()→GetTime() +
82         ↳ mEventStartTime;
83         double lensGrowthFactor = .09256 * pow(currRetinaAge, .52728); ////
84         ↳ power law model fit for lens growth
85         unsigned currentPopulationTarget = int(std::round(mBasePopulation
86         ↳ * lensGrowthFactor));
87
88         unsigned currentStemPopulation =
89         ↳ (mPopulation→GetCellProliferativeTypeCount())[0];
90         if (currentStemPopulation < currentPopulationTarget)
91         {
92             mMitoticMode = 0; //if the current population is < target,
93             ↳ symmetrical stem-stem division occurs (mode 0)
94         }
95     }
96
97     ****

```

```

88     * Write mitotic event to relevant files
89     * *****/
90     if (mDebug)
91     {
92         WriteDebugData();
93     }
94
95     if (mOutput)
96     {
97         WriteModeEventOutput();
98     }
99
100    AbstractSimpleCellCycleModel::ResetForDivision();
101
102 }
103
104 void WanStemCellCycleModel::Initialise()
105 {
106
107     boost::shared_ptr<AbstractCellProperty> p_Stem =
108
109         → mpCell→rGetCellPropertyCollection().GetCellPropertyRegistry()→Get<S
110     mpCell→SetCellProliferativeType(p_Stem);
111
112     SetCellCycleDuration();
113 }
114
115 void WanStemCellCycleModel::InitialiseDaughterCell()
116 {
117
118     if (mMitoticMode == 1)
119     {
120         /*****
121          * RPC-fated cells are given HeCellCycleModel
122          *****/
123
124         double tiLOffset = -(SimulationTime::Instance()→GetTime());
125         //Initialise a HeCellCycleModel and set it up with appropriate
126         → TiL value & parameters
127         HeCellCycleModel* p_cycle_model = new HeCellCycleModel;
128         p_cycle_model→SetModelParameters(tiLOffset, mHeParamVector[0],
129         → mHeParamVector[1], mHeParamVector[2],

```

```

127                                     mHeParamVector[3],
128                                     ↵ mHeParamVector[4],
129                                     ↵ mHeParamVector[5],
130                                     ↵ mHeParamVector[6],
131                                     mHeParamVector[7],
132                                     ↵ mHeParamVector[8],
133                                     ↵ mHeParamVector[9],
134                                     ↵ mHeParamVector[10],
135                                     mHeParamVector[11]);
136
137     p_cycle_model→EnableKillSpecified();
138
139     //if debug output is enabled for the stem cell, enable it for its
140     //progenitor offspring
141     if (mDebug)
142     {
143         p_cycle_model→PassDebugWriter(mDebugWriter, mTimeID,
144                                         ↵ mVarIDs);
145     }
146
147
148     mpCell→SetCellCycleModel(p_cycle_model);
149     p_cycle_model→Initialise();
150 }
151
152     SetCellCycleDuration();
153
154 }
155
156
157 void WanStemCellCycleModel::SetModelParameters(double gammaShift, double
158                                                 ↵ gammaShape, double gammaScale,
159                                                 std::vector<double>
160                                                 ↵ heParamVector)
161 {
162     mGammaShift = gammaShift;
163     mGammaShape = gammaShape;
164     mGammaScale = gammaScale;
165     mHeParamVector = heParamVector;
166 }
167
168 void WanStemCellCycleModel::EnableExpandingStemPopulation(int
169                 ↵ basePopulation,

```

```

158
159 {
160     mExpandingStemPopulation = true;
161     mBasePopulation = basePopulation;
162     mPopulation = p_population;
163 }
164
165 void WanStemCellCycleModel::SetTimeDependentCycleDuration(double
166     ↵ peakRateTime, double increasingSlope,
167
168     ↵ double
169     ↵ decreasingSlope)
170
171 {
172     mTimeDependentCycleDuration = true;
173     mPeakRateTime = peakRateTime;
174     mIncreasingRateSlope = increasingSlope;
175     mDecreasingRateSlope = decreasingSlope;
176     mBaseGammaScale = mGammaScale;
177 }
178
179
180
181
182 void WanStemCellCycleModel::EnableModeEventOutput(double eventStart,
183     ↵ unsigned seed)
184 {
185     mOutput = true;
186     mEventStartTime = eventStart;
187     mSeed = seed;
188 }
189
190
191 void
192     ↵ WanStemCellCycleModel::EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
193     ↵ debugWriter)
194 {

```

```

192     mDebug = true;
193     mDebugWriter = debugWriter;
194
195     mTimeID = mDebugWriter->DefineUnlimitedDimension("Time", "h");
196     mVarIDs.push_back(mDebugWriter->DefineVariable("CellID", "No"));
197     mVarIDs.push_back(mDebugWriter->DefineVariable("TIL", "h"));
198     mVarIDs.push_back(mDebugWriter->DefineVariable("CycleDuration",
199         "h"));
200     mVarIDs.push_back(mDebugWriter->DefineVariable("Phase2Boundary",
201         "h"));
202     mVarIDs.push_back(mDebugWriter->DefineVariable("Phase3Boundary",
203         "h"));
204     mVarIDs.push_back(mDebugWriter->DefineVariable("Phase", "No"));
205     mVarIDs.push_back(mDebugWriter->DefineVariable("MitoticModeRV",
206         "Percentile"));
207     mVarIDs.push_back(mDebugWriter->DefineVariable("MitoticMode",
208         "Mode"));
209     mVarIDs.push_back(mDebugWriter->DefineVariable("Label", "binary"));
210
211     mDebugWriter->EndDefineMode();
212 }
213
214 void WanStemCellCycleModel::WriteDebugData()
215 {
216     double currentTime = SimulationTime::Instance()->GetTime();
217     double currentCellID = mpCell->GetCellId();
218
219     mDebugWriter->PutVariable(mTimeID, currentTime);
220     mDebugWriter->PutVariable(mVarIDs[0], currentCellID);
221     mDebugWriter->PutVariable(mVarIDs[1], 0);
222     mDebugWriter->PutVariable(mVarIDs[2], mCellCycleDuration);
223     mDebugWriter->PutVariable(mVarIDs[3], 0);
224     mDebugWriter->PutVariable(mVarIDs[4], 0);
225     mDebugWriter->PutVariable(mVarIDs[5], 0);
226     mDebugWriter->PutVariable(mVarIDs[7], mMitoticMode);
227     mDebugWriter->AdvanceAlongUnlimitedDimension();
228
229 /*****
230 * UNUSED FUNCTIONS (required for class nonvirtuality, do not remove)
231 *****/
232
233 double WanStemCellCycleModel::GetAverageTransitCellCycleTime()

```

```

230 {
231     return (0.0);
232 }
233
234 double WanStemCellCycleModel :: GetAverageStemCellCycleTime()
235 {
236     return (0.0);
237 }
238
239 void WanStemCellCycleModel :: OutputCellCycleModelParameters(out_stream&
240   ↪ rParamsFile)
241 {
242     *rParamsFile << "\t\t\t<CellCycleDuration>" << mCellCycleDuration <<
243       ↪ "</CellCycleDuration>\n";
244
245 // Call method on direct parent class
246
247   ↪ AbstractSimpleCellCycleModel :: OutputCellCycleModelParameters(rParamsFile);
248 }
249
250 // Serialization for Boost ≥ 1.36
251 #include "SerializationExportWrapperForCpp.hpp"
252 CHASTE_CLASS_EXPORT(WanStemCellCycleModel)

```

---

### 17.1.34 /src/WanStemCellCycleModel.hpp

```

1 #ifndef WANSTEMCELLCYCLEMODEL_HPP_
2 #define WANSTEMCELLCYCLEMODEL_HPP_
3
4 #include "AbstractCellPopulation.hpp"
5 #include "AbstractSimpleCellCycleModel.hpp"
6 #include "RandomNumberGenerator.hpp"
7 #include "Cell.hpp"
8 #include "StemCellProliferativeType.hpp"
9 #include "SmartPointers.hpp"
10 #include "ColumnDataWriter.hpp"
11 #include "LogFile.hpp"
12
13 #include "HeCellCycleModel.hpp"
14
15 /***** *****/
16 * WAN STEM CELL CYCLE MODEL

```

```

17 * A simple model standing in for the CMZ "stem cells" described in'
18 * Wan et al. 2016 [Wan2016]
19 *
20 * USE: By default, WanStemCellCycleModels consistently divide
21 *       ↳ asymmetrically.
22 * InitialiseDaughterCell() marks offspring for RPC fate
23 * So-marked cells are given HeCellCycleModels for their next division.
24 * Change default model parameters with SetModelParameters(<params>);
25 *
26 * 2 per-model-event output modes:
27 * EnableModeEventOutput() enables mitotic mode event logging-all cells
28 *       ↳ will write to the singleton log file
29 * EnableModelDebugOutput() enables more detailed debug output, each seed
29 *       ↳ will have its own file written to
30 * by a ColumnDataWriter passed to it from the test
31 * (eg. by the SetupDebugOutput helper function in the project simulator)
32 *
33 * 1 mitotic-event-sequence sampler (only samples one "path" through the
34 *       ↳ lineage):
35 * EnableSequenceSampler() - one "sequence" of progenitors writes mitotic
35 *       ↳ event type to a string in the singleton log file
36 *
37 *****/
38
39 class WanStemCellCycleModel : public AbstractSimpleCellCycleModel
40 {
41     friend class TestSimpleCellCycleModels;
42
43     /** Needed for serialization. */
44     friend class boost::serialization::access;
45     /**
46      * Archive the cell-cycle model and random number generator, never
47      * used directly - boost uses this.
48      *
49      * @param archive the archive
50      * @param version the current version of this class
51      */
52     template<class Archive>
53     void serialize(Archive & archive, const unsigned int version)
54     {

```

```
54     archive &
55     → boost::serialization::base_object<AbstractSimpleCellCycleModel>(*this);
56
57     SerializableSingleton<RandomNumberGenerator>* p_wrapper =
58
59         → RandomNumberGenerator::Instance()→GetSerializationWrapper();
60     archive & p_wrapper;
61     archive & mCellCycleDuration;
62 }
63
64 //Private write functions for models
65 void WriteModeEventOutput();
66 void WriteDebugData();
67
68 protected:
69     //mode/output variables
70     bool mExpandingStemPopulation;
71     boost::shared_ptr<AbstractCellPopulation<2>> mPopulation;
72     bool mOutput;
73     double mEventStartTime;
74     //debug writer stuff
75     bool mDebug;
76     int mTimeID;
77     std::vector<int> mVarIDs;
78     boost::shared_ptr<ColumnDataWriter> mDebugWriter;
79     //model parameters and state memory vars
80     int mBasePopulation;
81     double mGammaShift;
82     double mGammaShape;
83     double mGammaScale;
84     unsigned mMitoticMode;
85     unsigned mSeed;
86     bool mTimeDependentCycleDuration;
87     double mPeakRateTime;
88     double mIncreasingRateSlope;
89     double mDecreasingRateSlope;
90     double mBaseGammaScale;
91     std::vector<double> mHeParamVector;
92
93 /**
 * Protected copy-constructor for use by CreateCellCycleModel().
 *
```

```
94     * The only way for external code to create a copy of a cell cycle
95     → model
96     * is by calling that method, to ensure that a model of the correct
97     → subclass is created.
98     * This copy-constructor helps subclasses to ensure that all member
99     → variables are correctly copied when this happens.
100    *
101    * This method is called by child classes to set member variables for
102    → a daughter cell upon cell division.
103    * Note that the parent cell cycle model will have had
104    → ResetForDivision() called just before CreateCellCycleModel() is
105    → called,
106    * so performing an exact copy of the parent is suitable behaviour.
107    → Any daughter-cell-specific initialisation
108    * can be done in InitialiseDaughterCell().
109    *
110    * @param rModel the cell cycle model to copy.
111    */
112 WanStemCellCycleModel(const WanStemCellCycleModel& rModel);

113
114 public:
115
116 /**
117  * Constructor - just a default, mBirthTime is set in the
118  → AbstractCellCycleModel class.
119  */
120 WanStemCellCycleModel();

121 /**
122  * SetCellCycleDuration() method to set length of cell cycle
123  */
124 void SetCellCycleDuration();

125 /**
126  * Overridden builder method to create new copies of
127  * this cell-cycle model.
128  *
129  * @return new cell-cycle model
130  */
131 AbstractCellCycleModel* CreateCellCycleModel();

132 /**
133  * Overridden ResetForDivision() method.
```

```

129     */
130     void ResetForDivision();
131
132     /**
133      * Overridden Initialise() method.
134      * Sets proliferative type as stem cell
135      */
136
137     void Initialise();
138
139     /**
140      * Overridden InitialiseDaughterCell() method.
141      * Used to implement asymmetric mitotic mode
142      * Daughter cells are initialised w/ He cell cycle model
143      */
144     void InitialiseDaughterCell();
145
146     /*Model setup functions for standard He (SetModelParameters) and
147      → deterministic alternative (SetDeterministicMode) models
148      * Default parameters are from refits of He et al + deterministic
149      → alternatives
150      * He 2012 params: mitoticModePhase2 = 8, mitoticModePhase3 = 15,
151      → p1PP = 1, p1PD = 0, p2PP = .2, p2PD = .4, p3PP = .2, p3PD = 0
152      * gammaShift = 4, gammaShape = 2, gammaScale = 1, sisterShift = 1
153      */
154
155     void SetModelParameters(double gammaShift = 4, double gammaShape = 2,
156     → double gammaScale = 1, std::vector<double> heParamVector = { 8,
157     → 15, 1, 0, .2, .4, .2, 0, 4, 2, 1, 1 });
158
159     void EnableExpandingStemPopulation(int basePopulation,
160     → boost::shared_ptr<AbstractCellPopulation<2>> p_population);
161
162     void SetTimeDependentCycleDuration(double peakRateTime, double
163     → increasingSlope, double decreasingSlope);
164
165     //Functions to enable per-cell mitotic mode logging for mode rate &
166     → sequence sampling fixtures
167     //Uses singleton logfile
168     void EnableModeEventOutput(double eventStart, unsigned seed);
169
170     //More detailed debug output. Needs a ColumnDataWriter passed to it
171     //Only declare ColumnDataWriter directory, filename, etc; do not set
172     → up otherwise
173     void EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
174     → debugWriter);

```

```

162
163 //Not used, but must be overwritten lest WanStemCellCycleModels be
164   ↳ abstract
165   double GetAverageTransitCellCycleTime();
166   double GetAverageStemCellCycleTime();
167
168 /**
169  * Overridden OutputCellCycleModelParameters() method.
170  *
171  * @param rParamsFile the file stream to which the parameters are
172  ↳ output
173  */
174   virtual void OutputCellCycleModelParameters(out_stream& rParamsFile);
175
176 #include "SerializationExportWrapper.hpp"
177 // Declare identifier for the serializer
178 CHASTE_CLASS_EXPORT(WanStemCellCycleModel)
179 #endif /*WANSTEMCELLCYCLEMODEL_HPP_*/

```

---

## 17.2 NGRefTools

Github repository: <https://github.com/mmattocks/NGRefTools.jl>

### 17.2.1 /src/LogNormalUtils.jl

---

```

1 function get_lognormal_params(desired_μ, desired_σ)
2   μ²=desired_μ^2
3   σ²=desired_σ^2
4   ln_μ=log(μ²/sqrt(μ²+σ²))
5   ln_σ=sqrt(log(1+(σ²/μ²)))
6   return ln_μ, ln_σ
7 end
8
9 function get_lognormal_desc(d::LogNormal)
10   return mean(d), std(d)
11 end

```

---

### 17.2.2 /src/MarginalTDist.jl

---

```

1 """
2     MarginalTDist(v,μ,σ)
3
4 Return a shifted, scaled T Distribution with degrees of freedom `v`, mean
→   `μ`, and standard deviation `σ`.
5
6 This is the marginal distribution of the posterior mean for an
→   uninformative (reference) Normal Gamma prior distribution. It is also
→   the distribution of the posterior predictive for m=1 new
→   observations.
7
8     MarginalTDist(v)           #Standard TDist with v dof
9     MarginalTDist(v,μ,σ)      #TDist with v dof shifted by μ, scaled by σ
10
11    params(d)                 #Return d.v, d.μ, d.σ
12    mean(d)                   #Return d.μ (median, mode are identical)
13    std(d)                    #Return d.σ
14    quantile(d,p)            #Return quantile at probability p
15
16 Reference: Kevin P. Murphy, Conjugate Bayesian Analysis of the Gaussian
→   Distribution. 2007.
→   https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf
17 """
18 MarginalTDist=LocationScale{Float64,TDist{Float64}}
19
20 Distributions.dof(d::MarginalTDist) = d.ρ.v
21
22 Distributions.std(d::MarginalTDist) = d.σ
23
24 """
25     fit(MarginalTDist, x; PP=false)
26
27 Assuming an uninformative (reference) Normal Gamma prior, return the
→   marginal posterior distribution of the mean of a Normal model of
→   observations `x`.
28
29 Equivalent to the frequentist sampling distribution of the MLE mean. If
→   `PP=true`, return the posterior predictive distribution for m=1
→   additional observations sampled from the marginal posterior mean
→   instead.
30

```

```

31 Example:
32
33 julia> PMM_MarginalTDist(rand(10))
34 MarginalTDist{Float64}(
35   t: TDist{Float64}(v=9.0)
36   μ: 0.4789484996068401
37   σ: 0.08491142725619677
38 )
39
40 Reference: Kevin P. Murphy, Conjugate Bayesian Analysis of the Gaussian
41   Distribution. 2007.
42   ↳ https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf
43 """
44 function Distributions.fit(::Type{MarginalTDist},
45   ↳ x::AbstractVector{<:Real}; PP=false)
46   PP ? PPM_MTDist(x) : PMM_MTDist(x)
47 end
48
49 #Posterior marginal mean distribution
50 function PMM_MTDist(x)
51   n=length(x)
52   μ=mean(x)
53   ssr=sum((x.-μ).^2)
54   σ=sqrt(ssr/(n*(n-1)))
55   return MarginalTDist(μ,σ,TDist(n-1))
56 end
57
58 #Posterior predictive mean distribution
59 function PPM_MTDist(x)
60   n=length(x)
61   μ=mean(x)
62   ssr=sum((x.-μ).^2)
63   αn=(n-1)/2
64   βn=.5*ssr
65   return MarginalTDist(μ,sqrt((βn*(n+1))/(αn*n)),TDist(2*αn))
66 end
67 """
68 MTDist_MC_func(func, xs... ; lower=.025, upper=.975, mc_its=1e6,
69   ↳ summary=false)

```

```

68 Monte Carlo execute `func` with a random sample from a MarginalTDist
   → fitted to each x in xs, over `mc_its` iterates, returning the
   → calculated results (if `summary` is `true`) or the mean and quantiles
   → specified by `lower` and `upper`.

69
70 Example:
71
72     julia> NGRefTools.MTDist_MC_func(*,[rand(10),rand(10)],summary=true)
73     (0.09491370397229557, 0.20712818357976473, 0.3385677491994645)

74
75 See also: [`fit(MarginalTDist,x)`](@ref)
76 """
77 function MTDist_MC_func(func::Function, xs; lower=.025, upper=.975,
   → mc_its=1e7, summary=false)
78     dists=[fit(MarginalTDist,x) for x in xs]
79     results=Vector{Float64}()
80     for it in 1:mc_its
81         push!(results, func(rand.(dists)...))
82     end
83     summary ? (return quantile(results, lower), mean(results),
   → quantile(results,upper)) : (return results)
84 end
85
86 function plot_logn_MTDist(xs, colors, markers, labels, xlabel, ylabel;
   → args...)
87     plt=plot(;args...)
88     for (x,color) in zip(xs,colors)
89         min=floor(minimum(x)-.15*mean(x))
90         max=ceil(maximum(x)+.20*mean(x))
91         X=collect(min:max)
92         pmdist=fit(MarginalTDist, log.(x))
93         y=pdf(pmdist,log.(X))
94
   → plot!(plt,X,y,ribbon=(y,zeros(length(y))),label=:none,color=color,xlabel=
95 end
96     for (x,color,marker,label) in zip(xs,colors, markers,labels)
97         pmdist=fit(MarginalTDist, log.(x))
98         min=floor(minimum(x)-.15*mean(x))
99         max=ceil(maximum(x)+.20*mean(x))
100        X=collect(min:max)
101        y=pdf(pmdist,log.(X))
102        scaty=[mean(y) for x in 1:length(x)]
103        plt=scatter!(plt,x,scaty,color=color, marker=marker, label=label)

```

```

104     end
105
106     return plt
107 end
108
109 function plot_n_MTDist(xs, colors, markers, labels, xlabel, ylabel;
110     ← args ... )
111     plt=plot(;args ... )
112     for (x,color) in zip(xs,colors)
113         pmdist=fit(MarginalTDist, x)
114         min=quantile(pmdist, .01)
115         max=quantile(pmdist, .99)
116         X=collect(min:(max-min)/1000:max)
117         y=pdf(pmdist,X)
118
119         ← plot!(plt,x,y,ribbon=(y,zeros(length(y))),color=color,label=:none,xlabel=
120     end
121     for (x,color,marker,label) in zip(xs,colors, markers,labels)
122         pmdist=fit(MarginalTDist, x)
123         scaty=[pdf(pmdist,quantile(pmdist,.25)) for x in 1:length(x)]
124         plt=scatter!(plt,x,scaty,color=color, marker=marker,label=label)
125     end
126
127     return plt
128 end
129
130 function mean_mass_comparator(x,y;labels=[ "x" , "y" ])
131     xdist=fit(MarginalTDist,x)
132     ydist=fit(MarginalTDist,y)
133     if mean(xdist)>mean(ydist)
134         frac=ccdf(xdist,mean(ydist))
135         println("$(round(frac*100,digits=1))% of the marginal posterior
136             ← mean density of $(labels[1]) is above the mean of
137             ← $(labels[2]).")
138     else
139         frac=cdf(xdist,mean(ydist))
140         println("$(round(frac*100,digits=1))% of the marginal posterior
141             ← mean density of $(labels[1]) is below the mean of
142             ← $(labels[2]).")
143     end
144 end
145
146 function log_mean_mass_comparator(x,y;labels=[ "x" , "y" ])

```

```

141 xdist=fit(MarginalTDist,log.(filter(n→!iszero(n),x)))
142 ydist=fit(MarginalTDist,log.(filter(n→!iszero(n),y)))
143 if mean(xdist)>mean(ydist)
144     frac=ccdf(xdist,mean(ydist))
145     println("${(round(frac*100,digits=1))}% of the marginal posterior
146         ← mean density of ${labels[1]} is above the mean of
147         ← ${labels[2]}.")
148 else
149     frac=cdf(xdist,mean(ydist))
150     println("${(round(frac*100,digits=1))}% of the marginal posterior
151         ← mean density of ${labels[1]} is below the mean of
152         ← ${labels[2]}.")

end
end

```

---

### 17.2.3 /src/NGRef.jl

```

1 """
2     fit(NormalGamma, x)
3
4 Return the posterior `NormalGamma` distribution on the unknown mean and
4     ← precision of a Normal model of data vector `x`, assuming an
4     ← uninformative (reference) prior.
5
6 Example:
7
8     julia> fit(NormalGamma, rand(10))
9     NormalGamma{Float64}(mu=0.5052725306750604, nu=10.0, shape=4.5,
9     ← rate=0.5057485400844524)
10
11 Reference: Kevin P. Murphy, Conjugate Bayesian Analysis of the Gaussian
11     ← Distribution. 2007.
11     ← https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf
12 """
13     function Distributions.fit(::Type{NormalGamma}, x :: AbstractVector)
14         n=length(x)
15         μ=mean(x)
16         ssr=sum((x.-μ).^2)
17         α=(n-1)/2
18         β=ssr/2
19         return NormalGamma(μ,n,α,β)
20     end

```

```

21
22     function Distributions.params(d::NormalGamma)
23         return d.mu, d.nu, d.shape, d.rate
24     end
25
26     function marginals(d::NormalGamma)
27
28         ↳ m_T=MarginalTDist(d.mu,sqrt(d.rate/(d.shape*d.nu)),TDist(2*d.shape))
29         m_Ga=Gamma(d.shape,1/d.rate)
30         return m_T,m_Ga
31     end

```

---

### 17.2.4 /src/NGRefTools.jl

```

1 module NGRefTools
2     using ConjugatePriors, Distributions, Plots
3     #import Makie:surface
4
5     include("NGRef.jl")
6     export marginals
7     include("NIGRef.jl")
8     #export NGplot
9     include("MarginalTDist.jl")
10    export MarginalTDist, MTDist_MC_func, mean_mass_comparator,
11        ↳ log_mean_mass_comparator, plot_n_MTDist, plot_logn_MTDist
12    include("LogNormalUtils.jl")
13    export get_lognormal_params, get_lognormal_desc
14 end # module

```

---

### 17.2.5 /src/NIGRef.jl

```

1 """
2     fit(NormalInverseGamma, x)
3
4     Return the posterior `NormalInverseGamma` distribution on the unknown
5         ↳ mean and variance of a Normal model of data vector `x`, assuming an
6         ↳ uninformative (reference) prior.
7
8     Example:
9
10    julia> fit(NormalInverseGamma,rand(10))

```

```

9      NormalInverseGamma{Float64}(mu=0.5052725306750604, nu=10.0,
10     ↪ shape=4.5, rate=0.5057485400844524)
11
11 Reference: Kevin P. Murphy, Conjugate Bayesian Analysis of the Gaussian
12   ↪ Distribution. 2007.
12   ↪ https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf
12 """
13
13 function
14   ↪ Distributions.fit(::Type{NormalInverseGamma}, x::AbstractVector)
15   n=length(x)
16   Vn = 1/n
17   μ=mean(x)
18   an = -.5 + (n/2)
19   bn = .5 * (sum(x.^2)-(μ^2/Vn))
20   return NormalInverseGamma(μ,Vn,an,bn)
21
21 end
22
22 function Distributions.params(d::NormalInverseGamma)
23   return d.mu, d.v0, d.shape, d.scale
24 end
25
26 function marginals(d::NormalInverseGamma)
27
28   ↪ m_T=MarginalTDist(d.mu,sqrt(d.scale*d.v0/d.shape),TDist(2*d.shape))
29   m_Ga=InverseGamma(d.shape,d.scale)
30   return m_T,m_Ga
31 end
32
32 # function NGplot(d::NormalInverseGamma; grid=1000, upper=.95,
33   ↪ lower=.05)
34   #   marg_mean, marg_var = marginals(d)
35   #   xmax=quantile(marg_mean,upper)
36   #   xmin=quantile(marg_mean,lower)
37   #   ymax=quantile(marg_var,upper)
38   #   ymin=quantile(marg_var,lower)
39   #   xs=LinRange(xmin, xmax, grid)
40   #   ys=LinRange(ymin, ymax, grid)
41   #   zs=[pdf(d,x,y) for x in xs, y in ys]
42
42   #   return surface(xs,ys,zs)
43   # end

```

---

## 17.3 GMC\_NS

Github repository: [https://github.com/mmattocks/GMC\\_NS.jl](https://github.com/mmattocks/GMC_NS.jl)

### 17.3.1 /src/GMC\_NS.jl

---

```

1 module GMC_NS
2     using Distributions, Serialization, UnicodePlots
3     import ProgressMeter: AbstractProgress, Progress, @showprogress,
4         → next!, move_cursor_up_while_clearing_lines, printover,
4         → durationstring
5     import Printf: @sprintf
6     import Random: rand
7     import Statistics: unscaled_covzm, det, eigen
8     import Serialization: serialize, deserialize
9     import LinearAlgebra: dot, normalize
10    import BioBackgroundModels: lps
11    import StatsFuns: logaddexp, logsumexp, logsubexp
12    import Base: show
13    import Measurements: measurement
14    import ConjugatePriors: NormalGamma, NormalInverseGamma
15    import NGRefTools: MarginalTDist
16    import KernelDensity: kde, AbstractKDE
17
18    #GMC settings vector fields: [GMC_Nmin::Int64, GMC_τ_death::Float64,
19    → GMC_init_τ::Float64, GMC_tune_μ::Int64, GMC_tune_α::Float64,
20    → GMC_tune_PID::NTuple{3,Float64}, GMC_timestep_η::Float64,
21    → GMC_reflect_η::Float64, GMC_exhaust_σ::Float64,
22    → GMC_chain_κ::Int64]
23
24    #should GMC_reflect_η be smaller?
25    GMC_DEFAULTS=Vector{Any}([50,1e-5,1., 4,.25,(.3,0.,.0), .25, .005,
26    → 10., typemax(Int64)])
27    export GMC_DEFAULTS
28
29    include("ensemble/GMC_NS_E ensemble.jl")
30    export GMC_NS_E ensemble
31    include("model/GMC_NS_M model.jl")
32    export GMC_NS_Model, GMC_NS_Model_Record
33    include("model/model_utilities.jl")
34    include("GMC/galilean_trajectory.jl")
35    include("utilities/t_Tuner.jl")
36    include("utilities/ellipsoid.jl")

```

```

31   include("nested_sampler/search_patterns.jl")
32   include("nested_sampler/nested_step.jl")
33   include("nested_sampler/converge_ensemble.jl")
34   export converge_ensemble!
35   include("utilities/coordinate_utils.jl")
36   export box_bound!, to_prior, to_unit_ball, box_reflect!
37   include("ensemble/ensemble_utilities.jl")
38   export ensemble_history, clean_ensemble_dir, measure_evidence,
39     ↳ reset_ensemble!, move_ensemble!, copy_ensemble, rewind_ensemble,
40     ↳ show_models, show_models_e, get_model, rectify_ensemble!,
41     ↳ posterior_kde
42   include("utilities/ns_progressmeter.jl")
43   include("utilities/progress_displays.jl")
44   export
45     ↳ tuning_display, evidence_display, convergence_display, info_display, lh_display, l
46   include("utilities/stats.jl")
47   include("model/normal/Normal_Model.jl")
48   include("model/normal/Normal_Ensemble.jl")
49   export Normal_Ensemble
50   include("model/normal/LogNormal_Model.jl")
51   include("model/normal/LogNormal_Ensemble.jl")
52   export LogNormal_Ensemble
53   include("model/eggbox/Eggbox_Model.jl")
54   include("model/eggbox/Eggbox_Ensemble.jl")
55   export Eggbox_Ensemble
56 end # module

```

---

### 17.3.2 /src/GMC/galilean\_trajectory.jl

---

```

1 """
2     galilean_trajectory_sample!(m, e, tuner, cache)
3
4 Given 'm <: GMC_NS_Model', 'e <: GMC_NS_Ensemble', 'tuner <: GMC_Tuner',
5   ↳ and 'cache <: Union{Nothing,GMC_NS_Model}', return the next
6   ↳ model-sample from a Galilean trajectory through parameter space, as
7   ↳ well as any cached reflection model, obeying 'new_m.log_Li ≥
8   ↳ e.contour'.

```

```

6 Implements Skilling's 2019 update to the GMC algorithm. Briefly: proceed
→ "forward" along the trajectory's velocity vector unless the
→ likelihood contour is hit; in this case, attempt to reflect "east",
→ then "west", then "south". If a reflection is found, return the model
→ in the original position with the new velocity vector (maintaining
→ detailed balance), as well as the cached reflection model "forward"
→ along this new vector. If no model with likelihood greater than
→ contour can be found, return the original model (as long as the
→ trajectory is alive it will be retried at lower timestep  $\tau$ ).
7
8 Skilling, John. "Galilean and Hamiltonian Monte Carlo." In Proceedings,
→ 33:8. Garching, Germany: MDPI, 2019.
9 """
10 function galilean_trajectory_sample!(m, e, tuner, cache)
11     t=m.trajectory; next_i=m.i+1; cache_i=m.i+2
12
13     e.GMC_timestep_η > 0. ?
→     ( $\tau=\max(\text{tuner.}\tau+\text{rand}(\text{Normal}(0,\text{tuner.}\tau*\text{e.GMC_timestep_η})),\text{eps}())$ ) :
→     ( $\tau=\text{tuner.}\tau$ ) #perturb  $\tau$  if specified
14     d= $\tau*m.v$  #distance vector for given timestep
15     fwd_pos,adj_d,box_rflct=box_move(m.pos,d,e.box)
16
17     if !box_rflct #if we're not stopped by the sampling box
18         fwd_m=e.model_init!(t, next_i, to_prior.(fwd_pos,e.priors),
→         fwd_pos, m.v, e.obs, e.constants ...) #try to proceed along
→         distance vector
19         if m.θ==fwd_m.θ #if forward motion is no longer making a
→         difference to the parameter vector (ie timestep is too
→         small), kill the trajectory and bail out
20             tuner.τ=e.GMC_τ_death
21             return m, nothing
22         end
23     else
24         fwd_m=m #placeholder, no effect on box reflection
25     end
26
27     if box_rflct || (fwd_m.log_Li < m.log_Li) #if no model is available
→     from the position along the distance vector, search for
→     reflections and store in cache
28     process_report!(tuner, false) #report a reflection off a lh
→     contour
29

```

```

30 lhδ=lps(m.log_Li,-fwd_m.log_Li) #get the likelihood difference
   ↵ between the two points
31 n=boundary_norm(adj_d,lhδ) #get the gradient normal for the
   ↵ distance and lhδ
32 box_rflect ? (v[] =box_reflect(m.pos,e.box,m.v,e.GMC_reflect_η)) :
   ↵ v[] =reflect(m.v,n,e.GMC_reflect_η) #get the reflected velocity
   ↵ vector off the boundary "east", or off the sampling box
33 rd=τ*v[] #reflected distance given the timestep
34 east_pos,_=box_move(m.pos,rd,e.box)
35 cache=e.model_init(t, cache_i, to_prior.(east_pos,e.priors),
   ↵ east_pos, v[], e.obs, e.constants ...) #try going>
36
37 if cache.log_Li < m.log_Li && !box_rflect #no west reflection off
   ↵ the sampling box
38 process_report!(tuner, false)
39
40 west_pos,_=box_move(m.pos,-rd,e.box)
41 cache=e.model_init(t, cache_i, to_prior.(west_pos,e.priors),
   ↵ west_pos, -v[], e.obs, e.constants ...) #if east
   ↵ reflection fails, try west
42 end
43
44 if cache.log_Li < m.log_Li
45 process_report!(tuner, false)
46
47 e.GMC_reflect_η > 0. ? (v[] =r_perturb(-m.v, e.GMC_reflect_η))
   ↵ : (v[] =-m.v)
48 sd=τ*v[] #south distance along perturbed reflexn vec (inverted
   ↵ m.v)
49
50 south_pos,_=box_move(m.pos,sd,e.box)
51 cache=e.model_init(t, cache_i,
   ↵ to_prior.(south_pos,e.priors),south_pos,v[], e.obs,
   ↵ e.constants ...) #if west or box reflection fails, try
   ↵ south (reversed along the particle's vector)
52 end
53 cache.log_Li < m.log_Li && (process_report!(tuner, false)) #if
   ↵ that fails wait for smaller τ
54 end

```

```

56     if cache!=nothing && cache.log_Li ≥ m.log_Li #a reflection that
      ↵ results in a model above contour exists, so return a model in
      ↵ place with the appropriate velocity vector and the cached model
      ↵ down the reflection vector
57     r_m=construct_reflection(m, next_i, cache.v)
58     return (r_m, cache)
59 else #if no reflections occurred or no reflected positions above the
      ↵ contour were found - either forward search succeeded or the whole
      ↵ search failed
60   cache!=nothing && (cache=nothing) #reset cache if it's been set
      ↵ with a model below contour
61   fwd_m.i = next_i && fwd_m.log_Li ≥ m.log_Li ?
      ↵ (process_report!(tuner, true); return fwd_m, cache) : (return
      ↵ m, cache) #if the forward model is a new iterate and at or
      ↵ above the contour, return that, otherwise return the original
      ↵ model
62 end
63 end
64
65 function box_move(pos,d,box)
66   if !(all(box[:,1].<pos+d.<box[:,2])) #if not all
      ↵ coordinates after move would be in the box
67     boundary_d=box.-pos #get distances to box
      ↵ boundaries
68     b=isapprox.(pos,box) #find any boundaries that
      ↵ have been exceeded due to numerical creep
69     boundary_d[b]=0 #correct these distances
70     boundary_t=boundary_d./d #get time to hit
      ↵ boundaries, given particle displacement d
71
72     for idx in findall(x→x≥0.,boundary_t) #for any
      ↵ boundaries the particle is riding (distance
      ↵ 0), give a small positive time if the
      ↵ particle is headed into the boundary to allow
      ↵ that dimension to be selected for bounding
      ↵ (dim,bound)=(idx[1],idx[2])
73     boundary_t[idx]=0. && ((bound==1 && d[dim] <
      ↵ 0.) || (bound==2 && d[dim] > 0.)) &&
      ↵ (boundary_t[idx]=nextfloat(0.))
74
75   end
76

```

```

77
    ↳ first_b_idx=findfirst(isequal(minimum(boundary_t[findall(
    ↳ #find the index of the first boundary that
    ↳ would be hit
    first_b_d=boundary_d[first_b_idx]
    d=d.*([first_b_d/d[first_b_idx[1]]])
end

81
82     all(d.==0.) ? box_reflect=true : box_reflect=false
83
84     new_pos=pos+d
85     !(all(box[:,1].<new_pos.<box[:,2])) &&
    ↳ box_bound!(new_pos,box)
86
87     return new_pos, d, box_reflect
88 end

89
90 function box_reflect(pos,box,v,η)
91     b=isapprox.(pos,box)
92     low_idxs=[v[i]<0&&b[i,1] for i in 1:length(v)]
93     hi_idxs=[v[i]>0&&b[i,2] for i in 1:length(v)]
94
95     v[] = copy(v)
96     v[] [low_idxs]=-v[low_idxs]
97     v[] [hi_idxs]=-v[hi_idxs]
98
99     η > 0. ? (return r_perturb(v[],η)) : (return v[])
100 end

101
102 function boundary_norm(v, lhδ)
103     b=lhδ./v
104     b[b.==Inf]=prevfloat(Inf) #rectify overflow
105     b[b.==0.]=rand([nextfloat(0.),prevfloat(0.)])
    ↳ #rectify exact 0.
106     b[b.== -Inf]=nextfloat(-Inf) #rectify underflow
107     return normalize(b)
108 end

109
110 function reflect(v, n, η)
111     v[] = v-(2*n*dot(n,v))
112     η > 0. ? (return r_perturb(v[],η)) : (return v[])
113 end

114

```

```

115     function r_perturb(v[], η)
116         return v[]*cos(η) +
117             → rand(MvNormal(length(v[]), 1.))*sin(η)
118     end

```

---

### 17.3.3 /src/ensemble/GMC\_NS\_Engine.jl

```

1  """
2      GMC_NS_Engine
3
4 Abstract supertype for model ensembles intended to be subjected to
→   Galilean Monte Carlo Nested Sampling ('GMC_NS'). Subtyped ensemble
→   structs must implement the following fields as their interface with
→   'GMC_NS':
5
6     mutable struct Minimal_Engine <: GMC_NS_Engine
7         path::String #ensemble records serialized here
8
9         models::Vector{<:GMC_NS_Model_Record} #records of model paths &
→   lhs
10        model_initλ::Function #GMC_NS_Model constructor
11
12        contour<:AbstractFloat #current log_Li[end], init
→   minimum([record.log_Li for record in models])
13        log_Li::Vector{<:AbstractFloat} #likelihood of lowest-ranked
→   model at iterate i, init [-Inf]
14        log_Xi::Vector{<:AbstractFloat} #amt of prior mass included in
→   ensemble contour at Li, init [0]
15        log_wi::Vector{<:AbstractFloat} #width of prior mass covered in
→   this iterate, init [-Inf]
16        log_Liwi::Vector{<:AbstractFloat} #evidentiary weight of the
→   iterate, init [-Inf]
17        log_Zi::Vector{<:AbstractFloat} #ensemble evidence, init
→   [typemin(FloatType)]
18        Hi::Vector{<:AbstractFloat} #ensemble information, init [0]
19
20        obs::Any #observations formatted for GMC_NS_Model likelihood
→   Function
21        priors::Vector{<:Distribution} #prior distributions on
→   GMC_NS_Model parameters, formatted for model_initλ
22        constants::Vector #fixed parameters to be passed to GMC_NS_Model
→   constructor

```

```

23
24     sample_posterior::Bool #switch for posterior_samples collection
25     posterior_samples::Vector{<:GMC_NS_Model_Record}
26
27     #GMC tuning settings
28     GMC_tune_i<:Integer #maximum number of initial τ tuning steps
29     GMC_tune_p<:AbstractFloat #use ensemble size*p GMC samples per
→   initial tuning step
30     GMC_tune_μ<:Integer #tune τ on the basis of this number of the
→   most recent steps
31     GMC_tune_α<:AbstractFloat #tune τ such that the acceptance rate
→   is between this value and 1.
32     GMC_tune_β<:AbstractFloat #multiply τ by 1+β (sampling acceptance
→   rate=1.) or 1-β (rate <α)
33
34     #GMC Sampler settings
35     GMC_timestep_η<:AbstractFloat #>0. to apply an perturbation term
→   randomly drawn from Normal(0, η*τ) to τ at each step (can assist
→   equilibration)
36     GMC_reflect_η<:AbstractFloat #>0. to apply perturbation to
→   reflection vectors, keep value small (can assist equilibration)
37     GMC_exhaust_σ<:AbstractFloat #apply this scale parameter to the
→   ensemble size to obtain the maximum number of GMC iterates to perform
→   before terminating nested sampling
38     GMC_chain_k<:Integer #maximum chain length
39
40     t_counter<:Integer #counter for next instantiated trajectory
→   number
41   end
42
43 """
44 abstract type GMC_NS_Enginele end
45
46 """
47
48   show(e<:GMC_NS_Enginele)
49   show(io,e<:GMC_NS_Enginele)
50
51 Display the ensemble path, likelihood histogram, contour, maximum log
→   likelihood, and log evidence (marginal likelihood).
52 """
53 function Base.show(io::IO, e::GMC_NS_Enginele; progress=false)
54     livec=[model.log_Li for model in e.models]

```

```

55     maxLH=maximum(livec)
56     printstyled(io, "$(typeof(e)) @ $(e.path)\n", bold=true)
57     msg = @sprintf "Contour: %3.6e MaxLH:%3.3e log Evidence:%3.6e"
58     ↪ e.contour maxLH e.log_Zi[end]
59     println(io, msg)
60     if length(unique(livec)) > 1
61     try
62         hist=UnicodePlots.histogram(livec, title="Ensemble Likelihood
63             ↪ Distribution")
64         show(io, hist)
65         println()
66         progress && return(nrows(hist.graphics)+6)
67     catch
68         printstyled("HISTOGRAM UNAVAILABLE", color=:green)
69         println()
70         progress && return 3
71     end
72 else
73     printstyled("HISTOGRAM UNAVAILABLE", color=:green)
74     println()
75     progress && return 3
76 end
77 end

```

---

### 17.3.4 /src/ensemble/ensemble\_utilities.jl

```

1 function ensemble_history(e::GMC_NS_Engineering, bins=25)
2     !e.sample_posterior && throw(ArgumentError("This ensemble has no
3         ↪ posterior samples to show a history for!"))
4     livec=vcat([model.log_Li for model in e.models],[model.log_Li for
5         ↪ model in e.posterior_samples])
6     show(histogram(livec, nbins=bins))
7 end
8
9 function e_backup(e::GMC_NS_Engineering, tuner::Dict{Int64,<:GMC_Tuner})
10    cp(string(e.path,'/','ens'),string(e.path,'/','ens.bak'), force=true)
11    serialize(string(e.path,'/','ens'), e)
12    serialize(string(e.path,'/','tuner'), tuner)
13 end
14
15 function clean_ensemble_dir!(e::GMC_NS_Engineering)
16     if e.sample_posterior

```

```

15     ens_paths=vcat(
16         [basename(m.path) for m in e.models],
17         [basename(m.path) for m in e.posterior_samples],
18         "ens", "tuner"
19     )
20 else
21     ens_paths=vcat(
22         [basename(m.path) for m in e.models],
23         "ens", "tuner"
24     )
25 end
26
27 for file in readdir(e.path)
28     !(file in ens_paths) && rm(e.path*'*file)
29 end
30 end
31
32 function complete_evidence(e::GMC_NS_Ensemble)
33     log_Z=e.log_Zi[end]
34     H=e.Hi[end]
35     live_weight=lps(e.log_Xi[length(e.log_Li)], -log(length(e.models)))
36     lis=sort([model.log_Li for model in e.models])
37     liwis=sort(lps.(lis, live_weight))
38
39     for (li,liwi) in zip(lis,liwis)
40         last_Z=log_Z
41         log_Z=logaddexp(log_Z,liwi)
42         H=lps(
43             (exp(lps(liwi,-log_Z)) * li),
44             (exp(lps(last_Z,-log_Z)) * lps(H,last_Z)),
45             -log_Z)
46     end
47
48     return log_Z, H
49 end
50
51 function measure_evidence(e::GMC_NS_Ensemble)
52     log_Z, H = complete_evidence(e)
53     return measurement(log_Z,sqrt(abs(H)/length(e.models)))
54 end
55
56 function reset_ensemble!(e::GMC_NS_Ensemble)

```

```

57   ↵ Ni=Int64.(round.(collect(-1:-1:-length(e.log_Li)+1)./e.log_Xi[2:end]))
58 N=Ni[1]
59
60 new_models=Vector{typeof(e.models[1])}()
61
62 for traj in 1:N
63   if string(traj)*".1" in [basename(record.path) for record in
64     ↵ e.models]
65     ↵ push!(new_models,e.models[findfirst(isequal(string(traj)*".1"),
66     ↵ [basename(record.path) for record in e.models])])
67   else
68     ↵ push!(new_models,e.posterior_samples[findfirst(isequal(string(traj)*
69     ↵ [basename(record.path) for record in
70     ↵ e.posterior_samples])])
71   end
72 end
73
74 e.models=new_models
75 e.posterior_samples=Vector{typeof(e.models[1])}()
76
77 e.contour=minimum([record.log_Li for record in e.models])
78
79 e.log_Li=[e.log_Li[1]]
80 e.log_Xi=[e.log_Xi[1]]
81 e.log_wi=[e.log_wi[1]]
82 e.log_Liwi=[e.log_Liwi[1]]
83 e.log_Zi=[e.log_Zi[1]]
84 e.Hi=[e.Hi[1]]
85
86 e.t_counter=length(e.models)+1
87
88 clean_ensemble_dir!(e)
89 isfile(e.path*"/tuner") && rm(e.path*"/tuner")
90 serialize(e.path*"/ens", e)
91
92 return e
93 end
94
95 function move_ensemble!(e::GMC_NS_Ensemble,path::String)
96 !isdir(path) && mkpath(path)

```

```

93     rectype=typeof(e.models[1])
94
95     for file in readdir(e.path)
96         mv(e.path*'*file,path*'*file)
97     end
98
99     for (n,model) in enumerate(e.models)
100        e.models[n]=rectype(model.trajectory, model.i, model.pos,
101                           → path*'*basename(model.path), model.log_Li)
102    end
103    if e.sample_posterior
104        for (n,model) in enumerate(e.posterior_samples)
105            e.posterior_samples[n]=rectype(model.trajectory, model.i,
106                                           → model.pos, path*'*basename(model.path), model.log_Li)
107        end
108    end
109
110    rm(e.path)
111    e.path=path
112    serialize(e.path*"/ens",e)
113    return e
114 end
115
116 function copy_ensemble(e::GMC_NS_E ensemble, path::String)
117     new_e=deepcopy(e)
118     !isdir(path) && mkdir(path)
119     rectype=typeof(e.models[1])
120     for file in readdir(e.path)
121         cp(e.path*'*file,path*'*file)
122     end
123
124     for (n,model) in enumerate(e.models)
125         new_e.models[n]=rectype(model.trajectory, model.i, model.pos,
126                               → path*'*basename(model.path), model.log_Li)
127     end
128     if e.sample_posterior
129         for (n,model) in enumerate(e.posterior_samples)
130             new_e.posterior_samples[n]=rectype(model.trajectory, model.i,
131                                              → model.pos, path*'*basename(model.path), model.log_Li)
132         end
133     end
134
135     new_e.path=path

```



```

169     return e
170 end

171
172 function show_models(e::GMC_NS_Ensemble, idxs)
173     liperm=sortperm([model.log_Li for model in e.models], rev=true)
174     for idx in idxs
175         m=deserialize(e.models[liperm[idx]].path)
176         show(m)
177     end
178 end

179
180 function show_models_e(e::GMC_NS_Ensemble, idxs)
181     liperm=sortperm([model.log_Li for model in e.models], rev=true)
182     for idx in idxs
183         m=deserialize(e.models[liperm[idx]].path)
184         show(stdout,m,e)
185     end
186 end

187
188 function get_model(e::GMC_NS_Ensemble, no)
189     return deserialize(e.path*"/"*string(no))
190 end

191
192 function reestimate_ensemble!(e::GMC_NS_Ensemble, wi_mode="trapezoidal")
193
194     → Ni=Int64.(round.(collect(-1:-1:-length(e.log_Li)+1)./e.log_Xi[2:end]))
195     push!(Ni, length(e.models))

196     #fix any borked starting values
197     e.log_Li[1]=-Inf #L0 = 0
198     e.log_Xi[1]=0. #X0 = 1
199     e.log_wi[1]=-Inf #w0 = 0
200     e.log_Liwi[1]=-Inf #Liwi0 = 0
201     e.log_Zi[1]=-Inf #Z0 = 0
202     e.Hi[1]=0. #H0 = 0,
203
204     for i in 1:length(e.log_Li)-1
205         j=i+1
206
207         e.log_Li[j]=e.posterior_samples[i].log_Li
208
209         e.log_Xi[j]=-i/Ni[i]
210

```

```

211     if wi_mode=="trapezoidal"
212         e.log_wi[j]= logsubexp(e.log_Xi[i], -j/Ni[j]) - log(2) #log
213             ↪   width of prior mass spanned by the last step-trapezoidal
214             ↪   approx
215     elseif wi_mode=="simple"
216         e.log_wi[j]= logsubexp(e.log_Xi[i], e.log_Xi[j]) #log width
217             ↪   of prior mass spanned by the last step-simple approx
218     else
219         throw(ArgumentError("Unsupported wi_mode!"))
220     end
221     e.log_Liwi[j]=lps(e.log_Li[j],e.log_wi[j]) #log likelihood + log
222         ↪   width = increment of evidence spanned by iterate
223     e.log_Zi[j]=logaddexp(e.log_Zi[i],e.log_Liwi[j])      #log evidence
224     #information- dimensionless quantity
225     Hj=lps(
226         (exp(lps(e.log_Liwi[j],-e.log_Zi[j])) * e.log_Li[j]),
227         (exp(lps(e.log_Zi[i],-e.log_Zi[j])) *
228             ↪   lps(e.Hi[i],e.log_Zi[i])),
229             ↪   -e.log_Zi[j])
230     Hj ≡ -Inf ? (e.Hi[j]=0.) : (e.Hi[j]=Hj)
231     end
232 end
233
234 #get a posterior kernel density estimate from the ensemble
235 function posterior_kde(e::GMC_NS_Ensemble;
236     ↪   bivar=Vector{Pair{<:Integer}}()())
237     !e.sample_posterior && throw(ArgumentError("This ensemble is not set
238         ↪   to collect posterior samples!"))
239
240     ↪   θ_mat=zeros(length(e.models[1].pos),length(e.models)+length(e.posterior_samples))
241     weight_vec=zeros(length(e.models)+length(e.posterior_samples))
242
243     ↪   logz, _=complete_evidence(e)
244
245     for (n,mrec) in enumerate(e.posterior_samples)
246         m=deserialize(mrec.path)
247         θ_mat[:,n]=m.θ
248         weight_vec[n]=lps([m.log_Li,e.log_Xi[n+1],-logz])
249     end
250
251     for (n,mrec) in enumerate(e.models)
252         m=deserialize(mrec.path)

```

```

246     p=length(e.posterior_samples)
247     θ_mat[:,n+p]=m.θ
248     weight_vec[n+p]=lps([m.log_Li, e.log_Xi[end],
249                           ↳ -log(length(e.models)), -logz])
250   end
251
252   kdes=Vector{AbstractKDE}()
253   for t in 1:size(θ_mat,1)
254     push!(kdes,kde(θ_mat[t,:],weights=exp.(weight_vec)))
255   end
256
257   for bv in bivar
258     ↳ push!(kdes,kde((θ_mat[bv[1],:],θ_mat[bv[2],:]),weights=exp.(weight_vec)))
259   end
260
261   return kdes
262 end

```

---

### 17.3.5 /src/model/GMC\_NS\_Model.jl

```

1 """
2
3 GMC_NS_Model_Record
4
5 Abstract supertype for records of serialised 'GMC_NS_Model's. Subtyped
6   record structs must implement the following fields as their interface
7   with 'GMC_NS':
8
9   struct Minimal_Record
10     trajectory::Integer #id assigned at construction, identifies
11     ↳ trajectory
12     i::Integer #id integer assigned at construction, identifies
13     ↳ position on trajectory
14     pos::Vector{<:AbstractFloat} #hypersphere position of
15     ↳ model-particle
16     path::String #model is serialized to ensembledir/trajectory.i
17     log_Li::AbstractFloat #log likelihood calculated by
18     ↳ 'GMC_NS_Model' constructor
19   end
20 """
21
22 abstract type GMC_NS_Model_Record end
23
24
25

```

```

16 """
17     GMC_NS_Model_Record
18
19 Abstract supertype for models in 'GMC_NS_Ensemble's. Subtyped model
20   structs must NOT be mutable, and must not have inner constructors
21   that make field-by-field construction inaccessible. Subtyped model
22   structs must implement the following fields as their interface with
23   'GMC_NS', in order, and all additional fields must follow:
24
25     mutable struct Minimal_Model
26       trajectory::Integer #id assigned at construction, identifies
27       trajectory
28       i::Integer #id integer assigned at construction, identifies
29       position on trajectory
30
31       θ::Vector{<:AbstractFloat} #model's parameter Vector
32       log_Li<:AbstractFloat #model's likelihood, calculated by
33       constructor
34
35       pos::Vector{<:AbstractFloat}
36       v::Vector{<:AbstractFloat} #model's velocity in hypersphere
37       coords
38     end
39
40 """
41 abstract type GMC_NS_Model end

```

---

### 17.3.6 /src/model/model\_utilities.jl

```

1 function construct_reflection(m, i, r_v)
2     params=[getfield(m,pn) for pn in propertynames(m)]
3     params[2]=i
4     params[6]=r_v
5     return typeof(m)(params...)
6 end

```

---

### 17.3.7 /src/model/eggbox/Eggbox\_Ensemble.jl

```

1 mutable struct Eggbox_Ensemble <: GMC_NS_Ensemble
2     path::String
3
4     model_initλ::Function

```

```

5   models :: Vector{Eggbox_Record}
6
7   contour :: Float64
8   log_Li :: Vector{Float64}
9   log_Xi :: Vector{Float64}
10  log_wi :: Vector{Float64}
11  log_Liwi :: Vector{Float64}
12  log_Zi :: Vector{Float64}
13  Hi :: Vector{Float64}
14
15  obs :: Nothing
16  priors :: Vector{<:Distribution}
17  constants :: Vector{<:Real}
18  box :: Matrix{Float64}
19
20  sample_posterior :: Bool
21  posterior_samples :: Vector{Eggbox_Record}
22
23  GMC_Nmin :: Int64
24
25  GMC_τ_death :: Float64
26  GMC_init_τ :: Float64
27  GMC_tune_μ :: Int64
28  GMC_tune_α :: Float64
29  GMC_tune_PID :: NTuple{3, Float64}
30
31  GMC_timestep_η :: Float64
32  GMC_reflect_η :: Float64
33  GMC_exhaust_σ :: Float64
34  GMC_chain_κ :: Int64
35
36  t_counter :: Int64
37 end
38
39 Eggbox_Engine(path :: String, no_models :: Integer, prior, box,
40   ← GMC_settings ... ; sample_posterior :: Bool = true) =
41 Eggbox_Engine(
42   path,
43   construct_eggbox_model,
44   assemble_EMs(path, no_models, prior, box, Vector{Real}() ... ,
45   [-Inf], #L₀ = 0
46     [0.], #X₀ = 1
47     [-Inf], #w₀ = 0

```

```

47     [-Inf], #Liwi0 = 0
48     [-Inf], #Z0 = 0
49     [0.], #H0 = 0,
50     nothing,
51     length(prior)=1 ? ([GMC_NS.marginals(prior) ... ]) : (prior),
52     Vector{Real}(),
53     length(prior)=1 ? (to_unit_ball.(box,[GMC_NS.marginals(prior) ... ]))
54     → : (to_unit_ball.(box,prior)),
55     sample_posterior,
56     Vector{Eggbox_Record}(),
57     GMC_settings ... ,
58     no_models+1)
59
60 function Base.show(io::IO, e::Eggbox_Ensemble; progress=true)
61     x1o=[0.0, 0.4, 0.8, 0.2, 0.6, 1.0, 0.0, 0.4, 0.8, 0.2, 0.6, 1.0,
62     → 0.0,0.4,0.8,0.2,0.6,1.0]
63     x2o=[0.0, 0.0, 0.0, 0.2, 0.2, 0.2, 0.4, 0.4, 0.4, 0.4, 0.6, 0.6, 0.6, 0.8,
64     → 0.8, 0.8,1.0, 1.0, 1.0]
65
66     x1m=[m.pos[1] for m in e.models]
67     x2m=[m.pos[2] for m in e.models]
68     x1m.=+1.; x2m.=+1.
69     x1m./=2.; x2m./=2
70
71     plt=scatterplot(x1o,x2o,title="Eggbox Ensemble: Contour
72     → $(round(e.contour,digits=2))",color=:green,name="Optima")
73     scatterplot!(plt, x1m, x2m, color=:magenta, name="Model Particles")
74     show(io, plt)
75     println()
76
77     (progress && return 19)
78 end
79
80
81
82 function assemble_EMs(path::String, no_trajectories::Integer, prior, box,
83     → constants)
84     ensemble_records = Vector{Eggbox_Record}()
85     !isdir(path) && mkpath(path)
86     length(prior)=1 ? (marginals=[GMC_NS.marginals(prior) ... ]) :
87     → (marginals=prior)
88     box=to_unit_ball.(box,marginals)
89
90     @showprogress 1 "Assembling Normal Model ensemble..." for
91     → trajectory_no in 1:no_trajectories

```

```

83         model_path = string(path,'/trajectory_no,'.',1)
84     if !isfile(model_path)
85         proposal=[rand.(prior) ... ]
86         pos=to_unit_ball.(proposal,marginals)
87         box_bound!(pos,box)
88         θvec=to_prior.(pos,marginals)
89
90         model = construct_eggbox_model(trajectory_no, 1, θvec, pos,
91             → [0.], nothing, constants ... ; v_init=true)
92
93             serialize(model_path, model) #save the model to
94             → the ensemble directory
95             push!(ensemble_records,
96                 → Eggbox_Record(trajectory_no,1,pos,model_path,model.log_Li))
97         else #interrupted assembly pick up from where we left off
98             model = deserialize(model_path)
99             push!(ensemble_records,
100                 → Eggbox_Record(trajectory_no,1,model.pos,model_path,model.log_Li))
101         end
102     end
103
104     return ensemble_records, minimum([record.log_Li for record in
105         → ensemble_records])
106 end

```

---

### 17.3.8 /src/model/eggbox/Eggbox\_Model.jl

```

1 mutable struct Eggbox_Record <: GMC_NS_Model_Record
2     trajectory::Int64
3     i::Int64
4     pos::Vector{Float64}
5     path::String
6     log_Li::Float64
7 end
8
9 struct Eggbox_Model <: GMC_NS_Model
10    trajectory::Integer #id integer assigned at construction
11    i::Integer #id of parent model, if any
12
13    θ::Vector{Float64} #model's parameter Vector
14    log_Li::Float64 #model's likelihood, calculated by constructor
15

```

```

16     pos::Vector{Float64}
17     v::Vector{Float64} #model's velocity in parameter space
18 end
19
20 function construct_eggbox_model(trajectory::Integer, i::Integer,
21     ↳ θ::Vector{Float64}, pos::Vector{Float64}, v::Vector{Float64},
22     ↳ obs::Nothing; v_init=false)
23     x1,x2=θ
24     log_lh=(2 + cos(5π * x1) * cos(5π * x2))^5
25     v_init && (v=rand(MvNormal(length(θ),1.)))
26
27     Eggbox_Model(trajectory, i, θ, log_lh, pos, v)
28 end
29
30 function Base.show(io::IO, m::Eggbox_Model; xsteps=100)
31     println("Eggbox Model $(m.trajectory).$(m.i), log_Li $(m.log_Li)")
32     println("θ: $(m.θ)")
33     println("v: $(m.v)")
34 end

```

---

### 17.3.9 /src/model/normal/LogNormal\_Ensemble.jl

```

1 mutable struct LogNormal_Ensemble <: GMC_NS_Ensemble
2     path::String
3
4     model_initλ::Function
5     models::Vector{Normal_Record}
6
7     contour::Float64
8     log_Li::Vector{Float64}
9     log_Xi::Vector{Float64}
10    log_wi::Vector{Float64}
11    log_Liwi::Vector{Float64}
12    log_Zi::Vector{Float64}
13    Hi::Vector{Float64}
14
15    obs::Vector{Float64}
16    priors::Vector{<:Distribution}
17    constants::Vector{<:Real}
18    box::Matrix{Float64}
19
20    sample_posterior::Bool

```

```

21 posterior_samples::Vector{Normal_Record}
22
23 GMC_Nmin::Int64
24
25 GMC_τ_death::Float64
26 GMC_init_τ::Float64
27 GMC_tune_μ::Int64
28 GMC_tune_α::Float64
29 GMC_tune_PID::NTuple{3,Float64}
30
31 GMC_timestep_η::Float64
32 GMC_reflect_η::Float64
33 GMC_exhaust_σ::Float64
34 GMC_chain_K::Int64
35
36 t_counter::Int64
37 end
38
39 LogNormal_Ensemble(path::String, no_models::Integer,
40   → obs::AbstractVector{<:AbstractFloat}, prior, box, GMC_settings ... ;
41   → sample_posterior::Bool=true) =
42 LogNormal_Ensemble(
43   path,
44   construct_lognormal_model,
45   assemble_lNMs(path, no_models, obs, prior, box, Vector{Real}()) ... ,
46   [-Inf], #L₀ = 0
47   [0.], #X₀ = 1
48   [-Inf], #w₀ = 0
49   [-Inf], #Liwi₀ = 0
50   [-Inf], #Z₀ = 0
51   [0.], #H₀ = 0,
52   obs,
53   length(prior)==1 ? ([GMC_NS.marginals(prior) ... ]) : (prior),
54   Vector{Real}(),
55   length(prior)==1 ? (to_unit_ball.(box,[GMC_NS.marginals(prior) ... ])) :
56   → : (to_unit_ball.(box,prior)),
57   sample_posterior,
58   Vector{Normal_Record}(),
59   GMC_settings ... ,
60   no_models+1)
61
62 function Base.show(io::IO, m::LogNormal_Model, e::LogNormal_Ensemble;
63   → xsteps=100, progress=true)

```

```

60   μ,λ=m.θ
61   lμ=log(μ^2/sqrt(μ^2 + inv(λ)))
62   lσ=sqrt(log(1+(inv(λ)/μ^2)))
63   n=LogNormal(lμ,lσ)
64
65   → X=[quantile(n,.025):(quantile(n,.975)-quantile(n,.025))/xsteps:quantile(n,.975)]
66
67   scattermin=maximum(y)*.25
68   scattermax=maximum(y)*.75
69
70   → scatterrange=[scattermin:(scattermax-scattermin)/length(e.obs):scattermax ... ]
71   scattery=[rand(scatterrange) for i in 1:length(e.obs)]
72
73   xlims=(min(minimum(e.obs),minimum(X)),max(maximum(e.obs),maximum(X)))
74
75   plt=lineplot(X,y,xlabel="X",ylabel="p",title="LogNormal Model
    → $(m.trajectory).$(m.i), log_Li $(m.log_Li)", name="Model",
    → xlim=xlims)
76   scatterplot!(plt, e.obs, scattery, name="Obs")
77
78   show(io, plt)
79   println()
80   println("θ: $(m.θ)")
81   println("v: $(m.v)")
82
83   progress && return nrows(plt.graphics)+8
84 end
85
86 function assemble_lNMs(path::String, no_trajectories::Integer, obs,
87   → prior, box, constants)
88   ensemble_records = Vector{Normal_Record}()
89   !isdir(path) && mkpath(path)
90   length(prior)==1 ? (marginals=[GMC_NS.marginals(prior)...]) :
91   → (marginals=prior)
92   box=to_unit_ball.(box,marginals)
93
94   @showprogress 1 "Assembling log-Normal Model ensemble..." for
95   → trajectory_no in 1:no_trajectories
96     model_path = string(path,'/',trajectory_no,'.',1)
97     if !isfile(model_path)
98       proposal=[rand.(prior)...]
99       pos=to_unit_ball.(proposal,marginals)

```

```

96         box_bound!(pos, box)
97         θvec=to_prior.(pos,marginals)
98
99         model = construct_lognormal_model(trajectory_no, 1, θvec,
100           → pos, [0.], obs, constants ... ; v_init=true)
100
101           serialize(model_path, model) #save the model to
102             → the ensemble directory
103             push!(ensemble_records,
104               → Normal_Record(trajectory_no,1,pos,model_path,model.log_Li)
105             else #interrupted assembly pick up from where we left off
106               model = deserialize(model_path)
107               push!(ensemble_records,
108                 → Normal_Record(trajectory_no,1,model.pos,model_path,model.log_Li)
109               end
110             end
111
112             return ensemble_records, minimum([record.log_Li for record in
113               → ensemble_records])
114   end

```

---

### 17.3.10 /src/model/normal/LogNormal\_Model.jl

```

1 #NOTE: LogNormal_Models reuse Normal_Records
2
3 struct LogNormal_Model <: GMC_NS_Model
4   trajectory::Integer #id integer assigned at construction
5   i::Integer #id of parent model, if any
6
7   θ::Vector{Float64} #model's parameter Vector
8   log_Li::Float64 #model's likelihood, calculated by constructor
9
10  pos::Vector{Float64}
11  v::Vector{Float64} #model's velocity in parameter space
12 end
13
14 function construct_lognormal_model(trajectory::Integer, i::Integer,
15   → θ::Vector{Float64}, pos::Vector{Float64}, v::Vector{Float64},
16   → obs::Vector{Float64}; v_init=false)
17   μ, λ=θ
18   lμ=log(μ^2/sqrt(μ^2 + inv(λ)))
19   lσ=sqrt(log(1+(inv(λ)/μ^2)))

```

```

18     mod_lnormal=LogNormal(lμ,lσ)
19     log_lh=lps(logpdf.(mod_lnormal, obs))
20     v_init && (v=rand(MvNormal(length(θ),1.)))
21
22     LogNormal_Model(trajectory, i, θ, log_lh, pos, v)
23 end
24
25 function Base.show(io::IO, m::LogNormal_Model; xsteps=100)
26     μ,λ=m.θ
27     lμ=log(μ^2/sqrt(μ^2 + inv(λ)))
28     lσ=sqrt(log(1+(inv(λ)/μ^2)))
29     n=LogNormal(lμ,lσ)
30
31     → X=[quantile(n,.025):(quantile(n,.975)-quantile(n,.025))/xsteps:quantile(n,.975)]
32     y=pdf.(n,X)
33
34     show(io, lineplot(X,y,xlabel="X",ylabel="p",title=title="LogNormal
35     → Model $(m.trajectory).$(m.i), log_Li $(m.log_Li)"))
36     println()
37     println("θ: $(m.θ)")
38     println("v: $(m.v)")
39 end

```

---

### 17.3.11 /src/model/normal/Normal\_Ensemble.jl

```

1 mutable struct Normal_Ensemble <: GMC_NS_Ensemble
2     path::String
3
4     model_initλ::Function
5     models::Vector{Normal_Record}
6
7     contour::Float64
8     log_Li::Vector{Float64}
9     log_Xi::Vector{Float64}
10    log_wi::Vector{Float64}
11    log_Liwi::Vector{Float64}
12    log_Zi::Vector{Float64}
13    Hi::Vector{Float64}
14
15    obs::Vector{Float64}
16    priors::Vector{<:Distribution}
17    constants::Vector{<:Real}

```

```

18   box::Matrix{Float64}
19
20   sample_posterior::Bool
21   posterior_samples::Vector{Normal_Record}
22
23   GMC_Nmin::Int64
24
25   GMC_τ_death::Float64
26   GMC_init_τ::Float64
27   GMC_tune_μ::Int64
28   GMC_tune_α::Float64
29   GMC_tune_PID::NTuple{3,Float64}
30
31   GMC_timestep_η::Float64
32   GMC_reflect_η::Float64
33   GMC_exhaust_σ::Float64
34   GMC_chain_κ::Int64
35
36   t_counter::Int64
37 end
38
39 Normal_Engsemble(path::String, no_models::Integer,
40   ↳ obs::AbstractVector{<:AbstractFloat}, prior, box, GMC_settings ... ;
41   ↳ sample_posterior::Bool=true) =
42 Normal_Engsemble(
43     path,
44     construct_normal_model,
45     assemble_NM(path, no_models, obs, prior, box, Vector{Real}()) ... ,
46     [-Inf], #L₀ = 0
47       [0.], #X₀ = 1
48       [-Inf], #w₀ = 0
49       [-Inf], #Liwi₀ = 0
50       [-Inf], #Z₀ = 0
51       [0.], #H₀ = 0,
52     obs,
53     length(prior)=1 ? ([GMC_NS.marginals(prior) ... ]) : (prior),
54     Vector{Real}(),
55     length(prior)=1 ? (to_unit_ball.(box,[GMC_NS.marginals(prior) ... ])) :
56     ↳ : (to_unit_ball.(box,prior)),
57     sample_posterior,
58     Vector{Normal_Record}(),
59     GMC_settings ... ,
60     no_models+1)

```

```

58
59 function Base.show(io::IO, m::Normal_Model, e::Normal_Ensemble;
60   ↪ xsteps=100, progress=true)
61   μ, λ=m.θ
62   σ=sqrt(1/λ)
63   n=Normal(μ,σ)
64   X=[μ-4σ:(μ+4σ)-(μ-4σ))/xsteps:μ+4σ ... ]
65   y=pdf.(n,X)
66
67   scattermin=y[Int(floor(xsteps/2)-floor(xsteps/3))]
68   scattermax=y[Int(floor(xsteps/2)-floor(xsteps/6))]
69
70   ↪ scatterrange=[scattermin:(scattermax-scattermin)/length(e.obs):scattermax ... ]
71   scattery=[rand(scatterrange) for i in 1:length(e.obs)]
72
73   xlims=(min(minimum(e.obs),minimum(X)),max(maximum(e.obs),maximum(X)))
74
75   plt=lineplot(X,y,xlabel="X",ylabel="p",title="Normal Model
76   ↪ $(m.trajectory).$(m.i), log_Li $(m.log_Li)", name="Model",
77   ↪ xlim=xlims)
78   scatterplot!(plt, e.obs, scattery, name="Obs")
79
80
81   show(io, plt)
82   println()
83   println("θ: $(m.θ)")
84   println("v: $(m.v)")
85
86   progress && return nrows(plt.graphics)+8
87
88 end

89
90 @showprogress 1 "Assembling Normal Model ensemble..." for
91   ↪ trajectory_no in 1:no_trajectories
92       model_path = string(path,'/',trajectory_no,'.',1)
93       if !isfile(model_path)
94           proposal=[rand.(prior) ... ]

```

```

94         pos=to_unit_ball.(proposal,marginals)
95         box_bound!(pos,box)
96         θvec=to_prior.(pos,marginals)
97
98         model = construct_normal_model(trajectory_no, 1, θvec, pos,
99             → [0.], obs, constants ... ; v_init=true)
100
101             serialize(model_path, model) #save the model to
102                 → the ensemble directory
103             push!(ensemble_records,
104                 → Normal_Record(trajectory_no,1,pos,model_path,model.log_Li)
105             else #interrupted assembly pick up from where we left off
106                 model = deserialize(model_path)
107                 push!(ensemble_records,
108                     → Normal_Record(trajectory_no,1,model.pos,model_path,model.log_Li))
109             end
110         end
111
112     return ensemble_records, minimum([record.log_Li for record in
113         → ensemble_records])
114 end

```

---

### 17.3.12 /src/model/normal/Normal\_Model.jl

```

1 mutable struct Normal_Record <: GMC_NS_Model_Record
2     trajectory::Int64
3     i::Int64
4     pos::Vector{Float64}
5     path::String
6     log_Li::Float64
7 end
8
9 struct Normal_Model <: GMC_NS_Model
10    trajectory::Integer #id integer assigned at construction
11    i::Integer #id of parent model, if any
12
13    θ::Vector{Float64} #model's parameter Vector
14    log_Li::Float64 #model's likelihood, calculated by constructor
15
16    pos::Vector{Float64}
17    v::Vector{Float64} #model's velocity in parameter space
18 end

```

```

19
20 function construct_normal_model(trajectory :: Integer, i :: Integer,
21   ↪ θ :: Vector{Float64}, pos :: Vector{Float64}, v :: Vector{Float64},
22   ↪ obs :: Vector{Float64}; v_init=false)
23   μ, λ=θ
24   mod_normal=Normal(μ,sqrt(1/λ))
25   log_lh=lps(logpdf.(mod_normal, obs))
26   v_init && (v=rand(MvNormal(length(θ),1.)))
27
28   Normal_Model(trajectory, i, θ, log_lh, pos, v)
29 end
30
31 function Base.show(io::IO, m::Normal_Model; xsteps=100)
32   μ, λ=m.θ
33   σ=sqrt(1/λ)
34   n=Normal(μ,σ)
35   X=[μ-4σ:((μ+4σ)-(μ-4σ))/xsteps:μ+4σ ... ]
36   y=pdf.(n,X)
37
38   show(io, lineplot(X,y,xlabel="X",ylabel="p",title="Normal Model
39   ↪ $(m.trajectory).$(m.i), log_Li $(m.log_Li)"))
40   println()
41   println("θ: $(m.θ)")
42   println("v: $(m.v)")
43 end

```

---

### 17.3.13 /src/nested\_sampler/converge\_ensemble.jl

```

1 """
2     converge_ensemble!(e; max_iterates, backup, clean,
2     ↪ converge_criterion, converge_factor, mc_noise, progargs)
3
4 Given 'e <: GMC_NS_Ensemble', recursively execute nested sampling steps
4   ↪ ('nested_step!()') until ensemble convergence criterion is obtained,
4   ↪ or 'max_iterates' is reached.
5
6 Convergence criteria specified by 'converge_criterion <: String':
6   ↪ "standard" for evidence convergence, "compression" for ensemble
6   ↪ likelihood range compression (can be used to improve MAP parameter
6   ↪ estimation after evidence has converged). Factor to use specified by
6   ↪ 'converge_factor <: AbstractFloat'. "Standard" defined in Skilling
6   ↪ 2006.

```

```

7
8 For likelihood estimates generated by Monte Carlo model evaluation,
→   'mc_noise <: AbstractFloat' defines the expected standard deviation
→   of likelihood in the MAP region of parameter space; this will be used
→   as an alternative convergence criterion if the ensemble is converged
→   to within this value.
9
10 Keyword arguments to GMC_NS_Progress are passed from progargs.
11
12 Skilling, John. "Nested Sampling for Bayesian Computations." In Proc.
→   Valencia. Benidorm (Alicante, Spain): inference.org.uk, 2006.
→   https://www.inference.org.uk/bayesys/Valencia.pdf.
13 """
14 function converge_ensemble!(e::GMC_NS_Ensemble;
→   max_iterates=typemax(Int64), backup::Tuple{Bool, Integer}=(false, 0),
→   clean::Tuple{Bool, Integer, Integer}=(false, 0, 0),
→   converge_criterion::String="standard",
→   converge_factor::AbstractFloat=1e-3, mc_noise::AbstractFloat=0.,
→   progargs ... )
15     N = length(e.models); curr_it=length(e.log_Li)
16
17     if curr_it==1 || !isfile(e.path*"/tuner")
18         serialize(e.path*"/ens", e)
19         tuner_dict=get_tuner_dict(e)
20     else
21         tuner_dict=deserialize(e.path*"/tuner") #restore tuner from saved
→           if any
22     end
23
24     meter = GMC_NS_Progress(e, .1; start_it=curr_it, progargs ... )
25
26     converge_check = get_convfunc(converge_criterion)
27
28     reflect_cache = nothing #init cache for models precalculated in
→       galilean reflection
29
30     cln_switch = clean[1] && !e.sample_posterior #ignore clean arguments
→       if posterior samples are to be collected
31
32     while !converge_check(e, converge_factor, mc_noise) && (curr_it ≤
→       max_iterates)
33         warn, reflect_cache = nested_step!(e, tuner_dict, reflect_cache)

```

```

34     warn == 1 && (@error "Failed to find new models, aborting at
35         ↵ current iterate."; return e)
36     curr_it += 1
37
38     backup[1] && curr_it%backup[2] == 0 && e_backup(e,tuner_dict)
39         ↵ #every backup interval, serialise the ensemble and tuner
40     clen_switch && curr_it%clean[2] == 0 &&
41         ↵ clean_ensemble_dir(e,clean[3]) #every clean interval, remove
42         ↵ old discarded models
43
44     update!(meter, converge_check(e,converge_factor, vals=true) ... )
45 end
46
47 if converge_check(e,converge_factor, mc_noise)
48     final_logZ=measure_evidence(e)
49     @info "Job done, sampled to convergence. Final logZ
50         ↵ $(final_logZ.val) ± $(final_logZ.err)"
51
52     e_backup(e,tuner_dict) #final backup
53     clen_switch && clean_ensemble_dir(e,0) #final clean
54     return final_logZ
55 elseif curr_it==max_iterates
56     @info "Job done, sampled to maximum iterate $max_iterates.
57         ↵ Convergence criterion not obtained."
58
59     e_backup(e,tuner_dict) #final backup
60     clen_switch && clean_ensemble_dir(e,0) #final clean
61     return e.log_Zi[end]
62 end
63 end
64
65     function evidence_converge(e, evidence_fraction,
66         ↵ mc_noise=0.; vals=false)
67         mc_noise > 0. && noise_check(e,mc_noise) && return
68             ↵ true
69
70         val=lps(findmax([model.log_Li for model in
71             ↵ e.models])[1], e.log_Xi[end])
72         thresh=lps(log(evidence_fraction),e.log_Zi[end])
73
74         vals ? (return val, thresh) : (return val

```

```

67         function compress_converge(e, compression_ratio,
68             mc_noise=0.; vals=false)
69             mc_noise > 0. && noise_check(e,mc_noise) && return
70                 true
71
72             val=findmax([model.log_Li for model in
73                         e.models])[1]-e.contour
74             thresh=compression_ratio
75
76
77         function get_convfunc(criterion)
78             if criterion == "standard"
79                 return evidence_converge
80             elseif criterion == "compression"
81                 return compress_converge
82             else
83                 throw(ArgumentError("Convergence criterion
84                             → $criterion not supported! Try \"standard\" or
85                             → \"compression\"."))
86             end
87         end
88
89         #if the entire ensemble is within 99.7% of the noise
90         #    distribution of its mean likelihood, it is converged
91         #    to the extent possible given Monte Carlo noise
92         function noise_check(e, mc_noise)
93             mod_lhs=[m.log_Li for m in e.models]
94             std(mod_lhs) ≤ mc_noise ? (return true) : (return
95                 false)
96         end

```

---

### 17.3.14 /src/nested\_sampler/nested\_step.jl

```

1 """
2     nested_step!(e<:GMC_NS_Ensemble)
3
4 Step and update ensemble e by sampling the e.contour-bounded prior mass
5     via Galilean Monte Carlo.

```

```

6    """
7
8 function nested_step!(e::GMC_NS_Ensemble, tuners::Dict{Int64,τ_PID},
9   cache::Union{GMC_NS_Model,Nothing})
10  N = length(e.models); N?=N #number of sample models/particles on the
11    ↵ posterior surface
12  i = length(e.log_Li) #iterate number, index for last values
13  j = i+1 #index for newly pushed values
14
15  e.contour, least_likely_idx = findmin([model.log_Li for model in
16    ↵ e.models])
17  Li_model = e.models[least_likely_idx]
18  m = deserialize(Li_model.path)
19  t=tuners[Li_model.trajectory]
20
21  #SELECT NEW MODEL, SAVE TO ENSEMBLE DIRECTORY, CREATE RECORD AND PUSH
22  ↵ TO ENSEMBLE
23
24  cache==nothing ? (model_selected=false; cache_insert=false) :
25    ↵ (model_selected=true; cache_insert=true)
26  samples=1;
27  ↵ sample_limit=Int64(floor(length(e.models)*e.GMC_exhaust_σ))
28
29  while !model_selected && samples≤sample_limit
30    if t.τ > e.GMC_τ_death && m.i ≤ e.GMC_chain_κ #while the
      ↵ particle's timestep is greater than the minimum to keep a
      ↵ chain alive, and the chain is below the maximum allowed
      ↵ length, sample by advancing it along a galilean trajectory
      model_selected,cache =
        ↵ galilean_search!(m,e,t,least_likely_idx,samples,sample_limit,cache)
        ↵ #ensemble is modified by galilean_search! directly
      elseif (N-1)<e.GMC_Nmin #if removing the particle would put the
        ↵ sample under the minimum size, resample by diffusion from an
        ↵ existing particle
        ↵ model_selected=diffusion_search!(e,tuners,least_likely_idx,N,e.GMC_τ_
          ↵ #ensemble modified by diffusion_search!
          samples+=1
        else #if not, push to the posterior samples and let the
          ↵ trajectory chain die
          model_selected=true

```

```

31     e.sample_posterior && push!(e.posterior_samples,
32         → e.models[least_likely_idx])#if sampling posterior, push
33         → the model record to the ensemble's posterior samples
34         → vector
35     deleteat!(e.models, least_likely_idx) #remove least likely
36         → model record from the active particles in the ensemble
37     N?-=1 #update for trapezoidal log_wi calc
38 end
39 end

40 if cache_insert #in this case a cached precalculated reflection model
41     → can be used
42     insert_reflection!(e, cache) #insert the model into the ensemble
43     process_report!(t, true)
44     cache=nothing #reset the cache
45 end

46 samples=sample_limit && (return 1, cache) #failed to find a new
47     → sample after sampling up to the limit, return an error code

48 #UPDATE ENSEMBLE QUANTITIES
49 push!(e.log_Li, e.contour) #log likelihood of the least likely model
50     → - the current ensemble ll contour at Xi
51 push!(e.log_Xi, -i/N) #log Xi - crude estimate of the iterate's
52     → enclosed prior mass
53 push!(e.log_wi, logsubexp(e.log_Xi[i], -j/N) - log(2)) #log width of
54     → prior mass spanned by the last step-trapezoidal approx
55 push!(e.log_Liwi, lps(e.log_Li[j],e.log_wi[j])) #log likelihood + log
56     → width = increment of evidence spanned by iterate
57 push!(e.log_Zi, logaddexp(e.log_Zi[i],e.log_Liwi[j]))    #log
58     → evidence
59 #information- dimensionless quantity. cf. MultiNest @
60     → https://github.com/farhanferoz/MultiNest/blob/master/MultiNest_v3.12/nested.F
61     → MultiNest now gives info only at the final step
62 Hj=lps(
63     (exp(lps(e.log_Liwi[j],-e.log_Zi[j])) * e.log_Li[j]),
64         → #information contribution of this step
65     (exp(lps(e.log_Zi[i],-e.log_Zi[j])) * lps(e.Hi[i],e.log_Zi[i])),
66         → #rescale last information by evidence
67     -e.log_Zi[j])
68 Hj == -Inf ? push!(e.Hi,0.) : push!(e.Hi, Hj) #prevent problems from
69     → early strings of models with -Inf log likelihoods

```

```

58     return 0, cache
59 end

```

---

### 17.3.15 /src/nested\_sampler/search\_patterns.jl

```

1 function galilean_search!(m,e,t,llidx,samples,s_limit,cache)
2     new_m,cache=galilean_trajectory_sample!(m,e,t,cache)
3     m.i==1 && ((m,new_m,cache)=start_trim!(m,new_m,cache))
4
5
6     while new_m.i==m.i && samples≤s_limit && t.τ > e.GMC_τ_death#failure
7         ← to find any new step, retune tau and try again until sample limit
8         ← reached or trajectory's tuner reaches taudeath
9         new_m,cache=galilean_trajectory_sample!(m,e,t,cache)
10        samples+=1
11        m.i==1 && ((m,new_m,cache)=start_trim!(m,new_m,cache))
12    end
13
14    if new_m.i==m.i+1
15        e.sample_posterior && push!(e.posterior_samples,
16            ← e.models[llidx])#if sampling posterior, push the model record
17            ← to the ensemble's posterior samples vector
18
19        rectype=typeof(e.models[1])
20        deleteat!(e.models, llidx) #remove least likely model record
21
22        new_model_record = rectype(new_m.trajectory,new_m.i, new_m.pos,
23            ← string(e.path,'/',new_m.trajectory,'.',new_m.i),
24            ← new_m.log_Li)
25        push!(e.models, new_model_record) #insert new model record
26        serialize(new_model_record.path, new_m)
27
28        return true, cache
29    else
30        return false, cache
31    end
32 end
33
34 function start_trim!(m,new_m,cache)
35     if new_m.i!=m.i && isapprox(new_m.pos,m.pos)
36         params=[getfield(m,pn) for pn in propertynames(m)]
37         params[6]=new_m.v

```

```

32     m=typeof(m)(params ... )
33     params=[getfield(cache,pn) for pn in propertynames(m)]
34     params[2]=2;
35     new_m=typeof(m)(params ... )
36     cache=nothing
37   end
38   return m,new_m,cache
39 end
40
41 function insert_reflection!(e, cache)
42   t_idx=findfirst(t→t==cache.trajectory,[m.trajectory for m in
43   ↪ e.models])
44
45   e.sample_posterior && push!(e.posterior_samples, e.models[t_idx])#if
46   ↪ sampling posterior, push the model record to the ensemble's
47   ↪ posterior samples vector
48   rectype=typeof(e.models[1])
49   deleteat!(e.models, t_idx) #remove least likely model record
50
51   new_model_record = rectype(cache.trajectory,cache.i, cache.pos,
52   ↪ string(e.path,'/',cache.trajectory,'.',cache.i), cache.log_Li)
53   push!(e.models, new_model_record) #insert new model record
54   serialize(new_model_record.path, cache)
55 end
56
57 function diffusion_search!(e, tuners, llidx, sample_limit, τ_limit)
58   sample=0; model_selected=false; new_m=0; src_traj=0
59   while !(sample > sample_limit) && !model_selected
60     sample +=1
61     mrec=rand(e.models)
62     model=deserialize(mrec.path)
63     randdir=rand(MvNormal(length(model.θ),1.))
64     τ=tuners[model.trajectory].τ
65     new_pos=model.pos+randdir*τ
66     box_bound!(new_pos,e.box)
67     new_m=e.model_initλ(e.t_counter, 1, to_prior.(new_pos,e.priors),
68     ↪ new_pos, randdir, e.obs, e.constants ... )
69
70     while τ*.7>τ_limit && new_m.log_Li<e.contour
71       τ*=.7
72       randdir=rand(MvNormal(length(model.θ),1.))
73       new_pos=model.pos+randdir*τ
74       box_bound!(new_pos,e.box)

```



```

105     while new_m.log_Li < e.contour && samples < s_limit
106         new_pos=sample_ellipsoid(e_ellip)
107         box_bound!(new_pos,e.box)
108         new_m=e.model_initλ(e.t_counter, 1, to_prior.(new_pos,e.priors),
109             ↳ new_pos, [0.], e.obs, e.box, e.constants..., v_init=true)
110         samples+=1
111     end
112
113     if new_m.log_Li > e.contour
114         e.sample_posterior && push!(e.posterior_samples,
115             ↳ e.models[llidx])#if sampling posterior, push the model record
116             ↳ to the ensemble's posterior samples vector
117         rectype=typeof(e.models[1])
118         deleteat!(e.models, llidx) #remove least likely model record
119
120         new_model_record = rectype(new_m.trajectory,new_m.i, new_m.pos,
121             ↳ string(e.path,'/',new_m.trajectory,'.',new_m.i),
122             ↳ new_m.log_Li)
123         push!(e.models, new_model_record) #insert new model record
124         serialize(new_model_record.path, new_m)
125         tdict[e.t_counter]=t_PID(e)
126         e.t_counter +=1
127         return true
128     else
129         return false
130     end
131 end

```

---

### 17.3.16 /src/utilities/coordinate\_utils.jl

```

1 function to_prior(pos,prior)
2     return quantile(prior, .5+.5*pos)
3 end
4
5 function to_unit_ball(pos,prior)
6     return (cdf(prior,pos)-.5)/.5
7 end
8
9
10 function box_bound!(pos,box)
11     if !(all(box[:,1].<pos.<box[:,2]))
12         pos[pos.<box[:,1]]*=box[:,1][pos.<box[:,1]]

```

```

13     pos[pos.>box[:,2]]*=box[:,2][pos.>box[:,2]]
14   end
15 end

```

---

### 17.3.17 /src/utilities/ellipsoid.jl

```

1 # Represents an N-dimensional ellipsoid
2 struct Ellipsoid
3     ctr::Vector{Float64} # center coordinates
4     cov::Array{Float64, 2}
5     icov::Array{Float64, 2} # inverse of cov
6     vol::Float64
7 end
8
9 # Draw a random point from within a unit N-ball
10 function randnball(ndim)
11     z = randn(ndim)
12     r2 = 0.
13     for i=1:ndim
14         r2 += z[i]*z[i]
15     end
16     factor = rand()^(1. /ndim) / sqrt(r2)
17     for i=1:ndim
18         z[i] *= factor
19     end
20     return z
21 end
22
23 # proportionality constant depending on dimension
24 # for n even:      (2pi)^(n    /2) / (2 * 4 * ... * n)
25 # for n odd : 2 * (2pi)^((n-1)/2) / (1 * 3 * ... * n)
26 function nball_vol_factor(ndim::Int)
27     if ndim % 2 == 0
28         c = 1.
29         for i=2:2:ndim
30             c *= 2pi / i
31         end
32         return c
33     else
34         c = 2.
35         for i = 3:2:ndim
36             c *= 2pi / i

```

```

37         end
38     return c
39   end
40 end
41
42 function ellipsoid_volume(scaled_cov::Matrix{Float64})
43     ndim = size(scaled_cov, 1)
44     return nbball_vol_factor(ndim) * sqrt(det(scaled_cov))
45 end
46
47 # find the bounding ellipsoid of points x where
48 function bounding_ellipsoid(x::Matrix{Float64}, enlarge=1.0)
49
50     ndim, npoints = size(x)
51
52     ctr = mean(x, dims=2)[:, 1]
53     delta = x .- ctr
54     cov = unscaled_covzm(delta, 2)
55     icov = inv(cov)
56
57     # Calculate expansion factor necessary to bound each point.
58     # This finds the maximum of (delta_i' * icov * delta_i)
59     fmax = -Inf
60     for k in 1:npoints
61         f = 0.0
62         for j=1:ndim
63             for i=1:ndim
64                 f += icov[i, j] * delta[i, k] * delta[j, k]
65             end
66         end
67         fmax = max(fmax, f)
68     end
69
70     fmax *= enlarge
71     cov .=* fmax
72     icov .=* 1. /fmax
73     vol = ellipsoid_volume(cov)
74
75     return Ellipsoid(ctr, cov, icov, vol)
76 end
77
78 function sample_ellipsoid(ell::Ellipsoid)
79     ndim = length(ell.ctr)

```

```

80
81     # Get scaled eigenvectors (in columns): vs[:,i] is the i-th
82     # → eigenvector.
83     f = eigen(ell.cov)
84     v, w = f.vectors, f.values
85     for j=1:ndim
86         tmp = sqrt(abs(w[j]))
87         for i=1:ndim
88             v[i, j] *= tmp
89         end
90     end
91
92     return v*randnball(ndim) + ell.ctr
93 end

```

---

### 17.3.18 /src/utilities/ns\_progressmeter.jl

```

1 const CONVERGENCE_MEMORY=500
2
3 mutable struct GMC_NS_Progress{T<:Real} <: AbstractProgress
4     interval::T
5     dt::AbstractFloat
6     start_it::Integer
7     counter::Integer
8     triggered::Bool
9     tfirst::AbstractFloat
10    tlast::AbstractFloat
11    tstop::AbstractFloat
12    printed::Bool      # true if we have issued at least one status
13    # → update
14    desc::AbstractString # prefix to the percentage, e.g. "Computing ... "
15    color::Symbol        # default to green
16    output::IO           # output stream into which the progress is
17    # → written
18    numprintedvalues::Integer # num values printed below progress in
19    # → last iteration
20    offset::Integer       # position offset of progress bar
21    # → (default is 0)
22
23    e::GMC_NS_Ensemble
24    #tuner::GMC_Tuner
25    top_m::GMC_NS_Model

```

```
22
23     mean_stp_time::AbstractFloat
24
25     upper_displays::Vector{Vector{Function}}
26     lower_displays::Vector{Vector{Function}}
27     display_rotate_iterates::Int64
28     upperidx::Integer
29     loweridx::Integer
30
31     convergence_history::Vector{Float64}
32     time_history::Vector{Float64}
33
34     function GMC_NS_Progress{T}(e::GMC_NS_Ensemble,
35                                 top_m::GMC_NS_Model,
36                                 #tuner::GMC_Tuner,
37                                 interval::T;
38                                 dt::Real=0.1,
39                                 desc::AbstractString="Nested Sampling ::",
40                                 color::Symbol=:green,
41                                 output::IO=stdout,
42                                 offset::Int=0,
43                                 start_it::Int=1,
44                                 upper_displays=[[evidence_display]],
45                                 lower_displays=[[ensemble_display]],
46                                 disp_rot_its=0,
47                                 ) where T
48         tfirst = tlast = time()
49         printed = false
50         new{T}(interval,
51                dt,
52                start_it,
53                start_it,
54                false,
55                tfirst,
56                tlast,
57                0.,
58                printed,
59                desc,
60                color,
61                output,
62                0,
63                offset,
64                e,
```

```

65         #tuner,
66         top_m,
67         0.,
68         upper_displays,
69         lower_displays,
70         disp_rot_its,
71         1,
72         1,
73         zeros(CONVERGENCE_MEMORY),
74         zeros(CONVERGENCE_MEMORY))
75     end
76 end
77 """
78     GMC_NS_Progress(e, interval; kwargs ... )
79
80
81 Return a GMC_NS.jl ProgressMeter for the given ensemble, to be updated at
82     ↪ 'interval'. Important keyword arguments:
83
84 'upper_displays <: AbstractVector{<:AbstractVector{Function}}': define a
85     ↪ vector of above-status-line display functions to rotate through
86 'lower_displays <: AbstractVector{<:AbstractVector{Function}}': define a
87     ↪ vector of below-status-line display functions to rotate through
88 'disp_rot_its <: Integer': define the number of iterates to rotate to the
89     ↪ next upper and lower display function vectors
90
91 """
92
93 function GMC_NS_Progress(e::GMC_NS_Engine,
94     #tuner::GMC_Tuner,
95     interval::Real; dt::Real=0.1, desc::AbstractString="GMC-NS::",
96     ↪ color::Symbol=:green, output::IO=stderr, offset::Integer=0,
97     ↪ start_it::Integer=1,
98     ↪ upper_displays::AbstractVector{<:AbstractVector{Function}}=Vector{Vector{Fun
99     ↪ lower_displays::AbstractVector{<:AbstractVector{Function}}=Vector{Vector{Fun
100    ↪ disp_rot_its::Integer=0})
101    top_m = deserialize(e.models[findmax([model.log_Li for model in
102        ↪ e.models])[2]].path)
103
104    return GMC_NS_Progress{typeof(interval)}(e, top_m,
105        # tuner,
106        interval, dt=dt, desc=desc, color=color, output=output,
107        ↪ offset=offset, start_it=start_it, upper_displays=upper_displays,
108        ↪ lower_displays=lower_displays, disp_rot_its=disp_rot_its)

```

```

96 end

97

98

99 function update!(p::GMC_NS_Progress, val, thresh; options...)
100     p.counter += 1

101

102     p.tstp=time()-p.tlast
103     popfirst!(p.time_history)
104     push!(p.time_history, p.tstp)

105

106     p.interval = val - thresh
107     isinf(p.interval) && (p.interval = 0.)
108     popfirst!(p.convergence_history)
109     push!(p.convergence_history, p.interval)

110

111     string(p.top_m.trajectory,'.',p.top_m.i) !=
112         → basename(p.e.models[findmax([model.log_Li for model in
113             → p.e.models])[2]].path) &&
114             → (p.top_m=deserialize(p.e.models[findmax([model.log_Li for model
115                 → in p.e.models])[2]].path))

116     updateProgress!(p; options...)
117
118 end

119 function updateProgress!(p::GMC_NS_Progress; offset::Integer = p.offset,
120     → keep = (offset == 0))
121     p.offset = offset
122     t = time()
123     p.display_rotate_iterates > 0 && p.counter%p.display_rotate_iterates
124         → = 0 && rotate_displays(p)

125

126     if p.interval < 0 && !p.triggered
127         p.triggered = true
128         if p.printed
129             p.triggered = true
130             dur = durationstring(t-p.tfirst)
131             msg = @sprintf "%s Converged. Time: %s (%d iterations). logZ:
132                 → %s\n" p.desc dur p.counter p.e.log_Zi[end]

133

134     print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
135     move_cursor_up_while_clearing_lines(p.output,
136         → p.numprintedvalues)

```

```

130         length(p.upper_displays)>0 ? (upper_lines=display_dash(p,
131             → p.upper_displays[p.upperidx])) : (upper_lines=0)
132         printover(p.output, msg, :magenta)
133         length(p.lower_displays)>0 ? (lower_lines=display_dash(p,
134             → p.lower_displays[p.loweridx])) : (lower_lines=0)

135
136     if keep
137         println(p.output)
138     else
139         print(p.output, "\r\u1b[A" ^ (p.offset +
140             → p.numprintedvalues))
141     end
142 end
143 return
144
145 if t > p.tlast+p.dt && !p.triggered
146     p.counter < CONVERGENCE_MEMORY ?
147         mean_step_time=mean(p.time_history[end-(p.counter-1):end]) :
148             → mean_step_time=mean(p.time_history)
149     msg = @sprintf "%s Iterate: %s Recent step time μ: %s Convergence
150             → Interval: %g\n" p.desc p.counter hmss(mean_step_time)
151             → p.interval
152
153     print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
154     move_cursor_up_while_clearing_lines(p.output, p.numprintedvalues)
155     length(p.upper_displays)>0 ? (upper_lines=display_dash(p,
156             → p.upper_displays[p.upperidx])) : (upper_lines=0)
157     printover(p.output, msg, p.color)
158     length(p.lower_displays)>0 ? (lower_lines=display_dash(p,
159             → p.lower_displays[p.loweridx])) : (lower_lines=0)

160     p.numprintedvalues=upper_lines + lower_lines + 1
161     print(p.output, "\r\u1b[A" ^ (p.offset + p.numprintedvalues))

162     # Compensate for any overhead of printing. This can be
163     # especially important if you're running over a slow network
164     # connection.
165     p.tlast = t + 2*(time()-t)
166     p.printed = true

```

```

164     end
165 end
166 function rotate_displays(p)
167     p.upperidx < length(p.upper_displays) ? p.upperidx+=1
168     ↪ : p.upperidx=1
169     p.loweridx < length(p.lower_displays) ? p.loweridx+=1
170     ↪ : p.loweridx=1
171 end
172
173 function hmss(dt)
174     dt<0 ? (dt=-dt; prfx="-") : (prfx="")
175     isnan(dt) && return "NaN"
176     (h,r) = divrem(dt,60*60)
177     (m,r) = divrem(r, 60)
178     (isnan(h)||isnan(m)||isnan(r)) && return "NaN"
179     string(prfx,Int(h),":",Int(m),":",Int(ceil(r)))
180 end
181
182 function display_dash(p::GMC_NS_Progress,
183     ↪ funcvec::Vector{Function})
184     lines=0
185     for func in funcvec
186         lines+=func(p)
187     end
188     return lines
189 end

```

---

### 17.3.19 /src/utilities/progress\_displays.jl

```

1 function tuning_display(p)
2     lines=show(p.output, p.tuner, progress=true)
3     println();
4     return lines
5 end
6
7 function convergence_display(p)
8     try
9
10         ↪ ciplot=lineplot([p.counter-(length(p.convergence_history)-1):p.counter ...
11         ↪ p.convergence_history, title="Convergence Interval Recent ...
12         ↪ History", xlabel="Iterate", ylabel="CI", color=:yellow)
13         lines=nrows(ciplot.graphics)+5
14     catch
15     end
16 end

```

```
11     show(p.output, ciplot); println()
12     return lines
13   catch
14     printstyled(p.output, "CONVERGENCE PLOT UNAVAILABLE. STANDBY\n",
15                 ↳ bold=true, color=:yellow); println()
16     return 2
17   end
18 end
19
20 function evidence_display(p)
21   try
22     log_Zis=p.e.log_Zi[2:end]
23     posidx=findfirst(i→i > -Inf, log_Zis)
24     log_Zis[1:posidx-1] .= log_Zis[posidx]
25     evplot=lineplot(log_Zis, title="Evidence History",
26                      ↳ xlabel="Iterate", color=:red, name="Ensemble logZ")
27     lines=nrows(evplot.graphics)+5
28     show(p.output, evplot); println()
29     return lines
30   catch
31     printstyled(p.output, "EVIDENCE PLOT UNAVAILABLE. STANDBY\n",
32                 ↳ bold=true, color=:red); println()
33     return 2
34   end
35 end
36
37 function info_display(p)
38   try
39     infoplot=lineplot(p.e.Hi[2:end], title="Information History",
40                        ↳ xlabel="Iterate", color=:green, name="Ensemble H")
41     lines=nrows(infoplot.graphics)+5
42     show(p.output, infoplot); println()
43     return lines
44   catch
45     printstyled(p.output, "INFORMATION PLOT UNAVAILABLE. STANDBY\n",
46                 ↳ bold=true, color=:green); println()
47     return 2
48   end
49 end
50
51 function lh_display(p)
52   try
```

```

48     lhplot=lineplot(p.e.log_Li[2:end], title="Contour History",
49         ← xlabel="Iterate", color=:magenta, name="Ensemble logLH")
50     lines=nrows(lhplot.graphics)+5
51     show(p.output, lhplot); println()
52     return lines
53   catch
54     printstyled(p.output, "CONTOUR HISTORY UNAVAILABLE. STANDBY\n",
55         ← bold=true, color=:magenta); println()
56     return 2
57   end
58 end
59
60 function liwi_display(p)
61   try
62     ← liwiplot=lineplot([max(2,p.counter-(CONVERGENCE_MEMORY-1)):p.counter ... ],
63     ← title="Recent iterate evidentiary weight", xlabel="Iterate",
64     ← name="Ensemble log Liwi", color=:cyan)
65     lines=nrows(liwiplot.graphics)+5
66     show(p.output, liwiplot); println()
67     return lines
68   catch
69     printstyled(p.output, "EVIDENTIARY HISTORY UNAVAILABLE.
70         ← STANDBY\n", bold=true, color=:cyan); println()
71     return 2
72   end
73 end
74
75 function ensemble_display(p)
76   return lines=show(p.output, p.e, progress=true)
77 end
78
79 function model_display(p)
80   try
81     println("Current MAP model:")
82     return lines=show(p.output, p.top_m, progress=true)
83   catch
84     printstyled(p.output, "MODEL DISPLAY UNAVAILABLE\n", bold=true,
85         ← color=:blue); println()
86     return 3
87   end
88 end
89
90 end

```

---

```

84 function model_obs_display(p)
85     try
86         println("Current MAP model")
87         return lines=show(p.output, p.top_m, p.e, progress=true)
88     catch
89         printstyled(p.output, "MODEL DISPLAY UNAVAILABLE\n", bold=true,
90                     ← color=:blue); println()
91         return 3
92     end
93 end

```

---

### 17.3.20 /src/utilities/stats.jl

---

```

1 function get_lognormal_params(desired_μ, desired_σ)
2     μ²=desired_μ^2
3     σ²=desired_σ^2
4     ln_μ=log(μ²/sqrt(μ²+σ²))
5     ln_σ=sqrt(log(1+(σ²/μ²)))
6     return ln_μ, ln_σ
7 end
8
9 function marginals(d::NormalInverseGamma)
10    → m_T=MarginalTDist(d.mu,sqrt((d.shape*inv(d.vθ))/d.scale),TDist(2*d.shape))
11    m_Ga=InverseGamma(d.shape,d.scale)
12    return m_T,m_Ga
13 end
14
15 function marginals(d::NormalGamma)
16    m_T=MarginalTDist(d.mu,sqrt(d.rate/(d.shape*d.nu)),TDist(2*d.shape))
17    m_Ga=Gamma(d.shape,1/d.rate)
18    return m_T,m_Ga
19 end

```

---

### 17.3.21 /src/utilities/t\_Tuner.jl

---

```

1 abstract type GMC_Tuner end
2
3 mutable struct τ_Tuner <: GMC_Tuner
4     τ::Float64
5     τ_history::Vector{Float64}

```

```
6      α::Float64
7      β::Float64
8      a::BitVector
9      memory::Integer
10
11     function τ_Tuner(τ1, e::GMC_NS_Ensemble)
12         new(τ1,
13             zeros(),
14             e.GMC_tune_α,
15             e.GMC_tune_β,
16             falses(),
17             e.GMC_tune_μ)
18     end
19 end
20
21 mutable struct τ_PID <: GMC_Tuner
22     τ::Float64
23     τ_history::Vector{Float64}
24     α::Float64
25     β::Float64
26
27     a::BitVector
28     a_history::BitVector
29     []::Vector{Float64}
30     memory::Integer
31     Kp::Float64
32     Ki::Float64
33     Kd::Float64
34
35     ε::Float64
36     pterm::Float64
37     itemr::Float64
38     dterm::Float64
39
40     function τ_PID(e::GMC_NS_Ensemble)
41         new(e.GMC_init_τ,
42             zeros(),
43             e.GMC_tune_α,
44             1.,
45             falses(),
46             falses(),
47             Vector{Float64}(),
48             e.GMC_tune_μ,
```

```

49         e.GMC_tune_PID ... ,
50         0.,0.,0.,0.)
51     end
52 end
53
54 function process_report!(t::τ_PID, report::Bool)
55     push!(t.a_history, report)
56     length(t.a_history)>t.memory ? (t.a=t.a_history[end-t.memory:end]) :
57         (t.a=t.a_history)
58     push!(t.[],mean(t.a))
59     τ_update!(t)
60 end
61
62 function τ_update!(t::τ_PID)
63     push!(t.τ_history,t.τ)
64
65     t.ε=t.[][]
66     t.pterm=t.Kp * t.ε
67     !(t.τ==eps()) && (t.iterm+=(t.Ki * t.ε))
68
69     length(t.[]) > Int64(round(.1*t.memory)) ? (d =
70         t.[][]
71         - Int64(round(.1*t.memory))) - t.[][]
72         : #d is positive
73         if acceptance is declining- in this case want smaller beta and
74         tau
75         d=0.
76     t.dterm=t.Kd * d
77
78     t.β=max(eps(), 1. + t.pterm + t.iterm - t.dterm)
79     t.τ=max(eps(), t.τ * t.β)
80 end
81
82
83
84 function init_tune(e::GMC_NS_E ensemble)
85     @info "Performing initial GMC timestep tuning on ensemble ... "
86     τ=1.; tuned=false; τd=1.1; accept=1.; it=1
87     while !tuned && it<e.GMC_tune_ι
88         test_report=false(0)
89         println(stdout, "τ: $τ, It: $it, Accept: $accept")
90
91         while
92             length(test_report)<Int64(floor(length(e.models)*e.GMC_tune_ρ))
93             m=deserialize(rand([m.path for m in e.models]))
94             candidate=galilean_trajectory_sample!(m,e,τ)

```

```

87
88         candidate.id==m.id ? push!(test_report,0) :
89             ↪ push!(test_report,1)
90         candidate=0
91         GC.gc()
92     end
93
94     last_accept=accept
95     accept=sum(test_report)/length(test_report)
96
97     if accept == 1.
98         τ*=τd
99         last_accept == 1. && (τd*=1+e.GMC_tune_β)
100        last_accept < e.GMC_tune_α &&
101            ↪ (τd=min(1.000001,1-e.GMC_tune_β*τd))
102    elseif accept < e.GMC_tune_α
103        τ/=τd
104        last_accept == 1. && (τd=min(1.000001,1-e.GMC_tune_β*τd))
105        last_accept < e.GMC_tune_α && (τd*=1+e.GMC_tune_β)
106    else
107        return τ
108    end
109
110    it+=1
111    move_cursor_up_while_clearing_lines(stdout, 1)
112 end
113
114 function tune_τ!(t::τ_Tuner, step_report::BitVector)
115     push!(t.τ_history,t.τ)
116     t.a=vcat(t.a,step_report)
117     ls=length(t.a)
118     ls>t.memory && (t.a=t.a[ls-t.memory:end])
119
120     accept=sum(t.a)/ls
121     if accept==1.
122         t.τ=t.τ*(1+t.β)
123     elseif accept<t.α
124         t.τ=t.τ*(1-t.β)
125     end
126 end
127 end

```

```

128
129 function Base.show(io::IO, t::τ_PID; progress=false)
130     try
131         plot=lineplot(t.[], title="Recent Unobstructed Moves",
132                         xlabel="Proposals", ylabel="Rate", color=:white, name="Free
133                         moves")
134
135         lbls=plot.labels_left
136         (arrow="")
137         all(t.α .> [v for v in parse.(Float64,values(lbls))]) &&
138             (arrow="↑")
139         all(t.α .< [v for v in parse.(Float64,values(lbls))]) &&
140             (arrow="↓")
141
142         lineplot!(plot, [t.α for i in 1:length(t.[])], name="α setpoint
143                         *arrow, color=:red")
144         show(io, plot)
145         println()
146         printstyled("τ:$(t.τ), α:$(t.α), β:$(t.β)", bold=true)
147         println()
148         printstyled("Kp:$(t.Kp), Ki:$(t.Ki), Kd:$(t.Kd)", color=:green)
149         println()
150         printstyled("ptrm:$(t.pterm), itrm:$(t.iterm),
151                         dterm:$(t.dterm)", color=:magenta)
152         println()
153
154     end
155
156     progress && return nrows(plot.graphics)+9
157     catch
158         printstyled("τ_PID NOT AVAILABLE. STANDBY", bold=true)
159         progress && return 1
160     end
161
162     function get_tuner_dict(e)
163         d=Dict{Int64,τ_PID}()
164         for rec in e.models
165             d[rec.trajectory]=τ_PID(e)
166         end
167         return d
168     end

```

---

### 17.3.22 /test/eggbox\_test.jl

---

```

1 using GMC_NS, Distributions
2
3 prior=Uniform(0,1)
4 priors=[prior,prior]
5
6 box=[0. 1.
7      0. 1.]
8
9 gmc=GMC_DEFAULTS
10 gmc[1]=180
11 gmc[2]=eps()
12
13 e=Eggbox_Ensemble("Eggbox_test", 1000, priors, box, gmc ... )
14
15 uds=Vector{Vector{Function}}{[[evidence_display],[liwi_display],[convergence_display]]
16 lds=Vector{Vector{Function}}{[[ensemble_display]]}
17
18 converge_ensemble!(e,backup=(true,100),upper_displays=uds,lower_displays=lds,
    ↵ disp_rot_its=1000, converge_factor=1e-6)

```

---

### 17.3.23 /test/ensemble\_tests.jl

---

```

1 @testset "GMC_NS_Ensemble tests ..." begin
2     sample_dist=Normal(100.,5.)
3     samples=rand(sample_dist, 10000)
4
5     n_chains=10
6
7     max_mu=1000
8     min_mu=eps()
9     max_lambda=100
10    min_lambda=1e-4
11
12    ln_max_mu=log(max_mu^2/sqrt(max_mu^2 + inv(max_lambda)))
13    ln_min_mu=log(min_mu^2/sqrt(min_mu^2 + inv(min_lambda)))
14    ln_max_lambda=1/log(1+(inv(max_lambda)/max_mu^2))
15    ln_min_lambda=1/log(1+(inv(min_lambda)/min_mu^2))
16
17    n_priors=[Uniform(min_mu,max_mu),Uniform(min_lambda,max_lambda)]
18    n_box=[min_mu max_mu;min_lambda max_lambda]

```

---

```

19     ln_priors=[Uniform(ln_min_μ,ln_max_μ),Uniform(ln_min_λ,ln_max_λ)]
20     ln_box=[ln_min_μ ln_max_μ;ln_min_λ ln_max_λ]
21
22     gmc=GMC_DEFAULTS
23     gmc[1]=3
24     gmc[2]=1e-5
25     gmc[end]=1e4
26
27     ne=Normal_E ensemble("NE_test", n_chains, samples, n_priors, n_box,
28     →   gmc ... )
28     lne=LogNormal_E ensemble("LNE_test", n_chains, samples, ln_priors,
29     →   ln_box, gmc ... )
29
30     uds=Vector{Vector{Function}}()
31
32     lds=Vector{Vector{Function}}()
33
34     ne_ev = converge_ensemble!(ne, backup=(true,100), upper_displays=uds,
35     →   lower_displays=lds)
35     lne_ev = converge_ensemble!(lne, backup=(true,100), upper_displays=uds,
36     →   lower_displays=lds)
36
37     @test ne_ev > lne_ev
38
39     for e in [ne, lne]
40         @info "Testing detailed balance at $(e.path) ... "
41
42         @test issorted(e.log_Li)
43         @test all(e.log_Li.<0.)
44         @test issorted(e.log_Xi, rev=true)
45         @test all(e.log_Xi.≤0.)
46         @test all(e.log_wi.<0.)
47         @test all(e.log_Liwi.<0.)
48         @test all(e.log_Zi.<0.)
49
50         Nvec=Vector{Int64}()
51         for i in 2:length(e.log_Li)
52             Xi=e.log_Xi[i]
53             N=round(-(i-1)/Xi)
54             push!(Nvec, N)
55         end
56
57         @test all(e.GMC_Nmin.≤ Nvec.≤ n_chains)

```

```

58
59     for chain in 1:e.t_counter-1
60         ch_pfx=e.path*'*string(chain)*'.'
61         total_spls=0
62         while isfile(ch_pfx*string(total_spls+1))
63             total_spls+=1
64         end
65
66         lastm=deserialize(ch_pfx*"1")
67
68         for spl in 2:total_spls
69             splm=deserialize(ch_pfx*string(spl))
70
71             @test !(splm.θ!=lastm.θ && splm.v!=lastm.v)
72             @test !(splm.θ==lastm.θ && splm.v==lastm.v)
73             if splm.θ==lastm.θ
74                 @test splm.v!=lastm.v
75                 @test splm.log_Li==lastm.log_Li
76             elseif splm.v==lastm.v
77                 @test splm.θ!=lastm.θ
78                 @test splm.log_Li ≥ lastm.log_Li
79             else
80                 @error "Galilean trajectory generation is broken!"
81             end
82             lastm=splm
83         end
84     end
85
86
87     rm("NE_test", recursive=true)
88     rm("LNE_test", recursive=true)
89 end
90

```

---

### 17.3.24 /test/galilean\_unit\_tests.jl

---

```

1 @testset "Galilean trajectory unit tests ... " begin
2     #test box movement
3     pos=[.5,.5]
4     box=[0 1.
5         0 1.]
6     allow_d=[.1,.1]

```

```

7   box_d=[1.,1.]
8
9   allowed_pos, allowed_d=box_move(pos,allow_d,box)
10  @test allowed_pos==pos+allow_d
11  @test allowed_d == allow_d
12
13  boxed_pos, boxed_d=box_move(pos, box_d, box)
14  @test boxed_pos==[1.,1.]
15  @test boxed_d==[.5,.5]
16
17  box_reflect_v=box_reflect([.5,.99999999],box,[.1,.1],0.)
18  @test box_reflect_v==[.1,-.1]
19
20  #test boundary norm finding
21  norm=1000
22  v=[.5,.5]
23  bn=boundary_norm(v, norm)
24  @test bn == normalize(norm./v)
25
26  #test reflection given norm
27  rv=reflect(v,bn,0.)
28  @test isapprox(rv, -v)
29
30  #test reflection perturbation
31  pv=r_perturb(v,.01)
32  @test v!=pv
33 end

```

---

### 17.3.25 /test/normal\_demo.jl

```

1 using GMC_NS, Distributions, ConjugatePriors
2
3 sample_dist=Normal(100.,5.)
4 samples=rand(sample_dist, 10000)
5
6 prior=NormalGamma(300.,2e-3,1.,10.) #a lousy prior
7
8 box=[0. 1000.
9      1/1000. 1/eps()]
10
11 gmc=GMC_DEFAULTS
12 gmc[1]=5

```

---

```

13 gmc[2]=eps()
14
15 e=Normal_Ensemble("NE_test", 10, samples, prior, box, gmc ... )
16
17 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]]
18
19 lds=Vector{Vector{Function}}([[model_obs_display]])
20
21 converge_ensemble!(e,backup=(true,100),upper_displays=uds,
→ lower_displays=lds, disp_rot_its=1000)

```

---

### 17.3.26 /test/runtests.jl

---

```

1 @info "Loading test packages ... "
2 using GMC_NS, Test, Distributions, Serialization
3 import GMC_NS:box_move, box_reflect, boundary_norm, reflect, r_perturb
4 import LinearAlgebra:normalize
5
6 @info "Beginning tests ... "
7 using Random
8 Random.seed!()
9
10 include("galilean_unit_tests.jl")
11 include("ensemble_tests.jl")
12
13 @info "Tests complete!"

```

---

## 17.4 CMZNicheSims

Github repository: <https://github.com/mmattocks/CMZNicheSims.jl>

### 17.4.1 /src/CMZNicheSims.jl

---

```

1 module CMZNicheSims
2     using Distributions, GMC_NS, UnicodePlots, StatsPlots
3     import ProgressMeter:
→     @showprogress,move_cursor_up_while_clearing_lines
4     import BioBackgroundModels:lps
5     import StatsFuns:logsumexp
6     import Serialization:serialize,deserialize
7     import KernelDensity:kde,AbstractKDE

```

```

8
9  const MAXVAL=prevfloat(Inf) #don't permit infinite CMZ populations or
   ↳ retinal volumes
10
11 include("thymidine_sim/thymidine_cell.jl")
12 include("thymidine_sim/thymidine_model.jl")
13 include("thymidine_sim/thymidine_ensemble.jl")
14 export Thymidine_Eensemle
15 include("thymidine_sim/thymidine_sim.jl")
16 include("thymidine_sim/thymidine_lh.jl")
17 include("CMZ_sim/CMZ_lh.jl")
18 include("CMZ_sim/CMZ_model.jl")
19 include("CMZ_sim/CMZ_ensemble.jl")
20 export CMZ_Eensemle
21 include("slice_sim/lens_model.jl")
22 export Lens_Model
23 include("slice_sim/slice_lh.jl")
24 include("slice_sim/slice_model.jl")
25 include("slice_sim/slice_ensemble.jl")
26 export Slice_Eensemle
27 include("cycle_decay_sim/decay_lh.jl")
28 include("cycle_decay_sim/decay_constructor.jl")
29 include("cycle_decay_sim/decay_ensemble.jl")
30 export Decay_Eensemle
31 include("multislice_sim/multislice_model.jl")
32 include("multislice_sim/multislice_ensemble.jl")
33 export MultiSlice_Eensemle, MultiSlice_Decay_Eensemle
34 end # module

```

---

### 17.4.2 /src/CMZ\_sim/CMZ\_ensemble.jl

---

```

1 mutable struct CMZ_Eensemle <: GMC_NS_Eensemle
2   path::String
3
4   model_initλ::Function
5   models::Vector{CMZ_Record}
6
7   contour::Float64
8   log_Li::Vector{Float64}
9   log_Xi::Vector{Float64}
10  log_wi::Vector{Float64}
11  log_Liwi::Vector{Float64}

```

```

12 log_Zi::Vector{Float64}
13 Hi::Vector{Float64}

14

15     → obs::AbstractVector{<:Tuple{<:AbstractVector{<:Float64},<:AbstractVector{<:Fl
16 priors::Vector{<:Distribution}
17 constants::Vector{<:Any}
18 #T, popdist, voldist, mc_its, phs
19 box::Matrix{Float64}

20

21 sample_posterior::Bool
22 posterior_samples::Vector{CMZ_Record}

23

24 GMC_Nmin::Int64

25

26 GMC_τ_death::Float64
27 GMC_init_τ::Float64
28 GMC_tune_μ::Int64
29 GMC_tune_α::Float64
30 GMC_tune_PID::NTuple{3,Float64}

31

32 GMC_timestep_η::Float64
33 GMC_reflect_η::Float64
34 GMC_exhaust_σ::Float64
35 GMC_chain_κ::Int64

36

37 t_counter::Int64

38 end

39

40 CMZ_Ensemble(path::String, no_models::Integer,
    → obs::AbstractVector{<:Tuple{<:AbstractVector{<:Float64},<:AbstractVector{<:Float6
    → priors::AbstractVector{<:Distribution}, constants, box, GMC_settings;
    → sample_posterior::Bool=true) =
41 CMZ_Ensemble(
    path,
    construct_CMZ,
    assemble_CMs(path, no_models, obs, priors, constants, box)...,
42 [-Inf], #L0 = 0
    [0], #X0 = 1
43 [-Inf], #w0 = 0
    [-Inf], #Liwi0 = 0
44 [-Inf], #Z0 = 0
    [0], #H0 = 0,
45 [-Inf], #V0 = 0
    [0], #C0 = 0
46 [-Inf], #S0 = 0
    [0], #D0 = 0
47 [-Inf], #I0 = 0
    [0], #R0 = 0
48 [-Inf], #N0 = 0
    [0], #P0 = 0
49 [-Inf], #Q0 = 0
    [0], #M0 = 0
50 [-Inf], #L0 = 0
    [0], #X0 = 1

```

```

51     obs,
52     priors,
53     constants, #T, popdist, voldist, mc_its, phs
54     box,
55     sample_posterior,
56     Vector{CMZ_Record}(),
57     GMC_settings ... ,
58     no_models+1)
59
60 function assemble_CMs(path::String, no_trajectories::Integer, obs,
61   ↳ priors, constants, box)
62   ensemble_records = Vector{CMZ_Record}()
63   !isdir(path) && mkpath(path)
64   phs=constants[6]
65   @showprogress 1 "Assembling CMZ_Model ensemble..." for trajectory_no
66   ↳ in 1:no_trajectories
67     model_path = string(path,'/',trajectory_no,'.',1)
68     if !isfile(model_path)
69       proposal=rand.(priors)
70
71       ↳ proposal[2*phs+1:(2*phs+1)+(phs-2)]=sort(proposal[2*phs+1:(2*phs+1)+(
72
73       pos=to_unit_ball.(proposal,priors)
74       box_bound!(pos,box)
75       θvec=to_prior.(pos,priors)
76
77       model = construct_CMZ(trajectory_no, 1, θvec, pos, [0.], obs,
78       ↳ constants ... ; v_init=true)
79
80       serialize(model_path, model) #save the model to the ensemble
81       ↳ directory
82       push!(ensemble_records, CMZ_Record(trajectory_no, 1,
83         ↳ pos,model_path,model.log_Li))
84     else #interrupted assembly pick up from where we left off
85       model = deserialize(model_path)
86       push!(ensemble_records, CMZ_Record(trajectory_no, 1,
87         ↳ model.pos,model_path,model.log_Li))
88     end
89   end
90
91   return ensemble_records, minimum([record.log_Li for record in
92     ↳ ensemble_records])
93 end

```

```

86
87
88 function Base.show(io::IO, m::CMZ_Model, e::CMZ_Ensemble; progress=false)
89     T=e.constants[1]
90
91     catpobs=vcat([e.obs[t][1] for t in 1:length(T)]...)
92     ymax=max(maximum(m.disp_mat[:, :, 1]), maximum(catpobs))
93     ymin=min(minimum(m.disp_mat[:, :, 1]), minimum(catpobs))
94
95     plt=lineplot(T,m.disp_mat[:, 2, 1],title="CMZ_Model"
96                 ↳ $(m.trajectory).$(m.i), log_Li $(m.log_Li)",color=:green,name="μ
97                 ↳ pop", ylim=[ymin,ymax])
98     lineplot!(plt,T,m.disp_mat[:, 1, 1],color=:magenta,name="95% CI")
99     lineplot!(plt,T,m.disp_mat[:, 3, 1],color=:magenta)
100
101    Ts=vcat([[T[n] for i in 1:length(e.obs[n][1])] for n in
102              ↳ 1:length(T)]...)
103    scatterplot!(plt,Ts, catpobs, color=:yellow, name="Obs")
104
105    catvobs=vcat([e.obs[t][2] for t in 1:length(T)]...)
106    ymax=max(maximum(m.disp_mat[:, :, 1]), maximum(catvobs))
107    ymin=min(minimum(m.disp_mat[:, :, 1]), minimum(catvobs))
108
109    plt=lineplot(T,m.disp_mat[:, 2, 2],color=:blue,name="μ vol",
110                 ↳ ylim=[ymin,ymax])
111    lineplot!(plt,T,m.disp_mat[:, 1, 2],color=:yellow,name="95% CI")
112    lineplot!(plt,T,m.disp_mat[:, 3, 2],color=:yellow)
113
114    Ts=vcat([[T[n] for i in 1:length(e.obs[n][2])] for n in
115              ↳ 1:length(T)]...)
116    scatterplot!(plt,Ts, catvobs, color=:yellow, name="Obs")
117
118    show(io, plt)
119
120    println()
121    println("θ: $(m.θ)")
122    println("v: $(m.v)")
123
124    (progress && return nrows(plt.graphics)+26);
125 end

```

#### 17.4.3 /src/CMZ\_sim/CMZ\_lh.jl

```

1 function CMZ_mc_llh(popdist::Distribution, voldist::Distribution,
2   volconst::Float64, phase_ends::Vector{Float64},
3   pparams::Vector{Float64}, mc_its::Int64, T::Vector{<:AbstractFloat},
4   obs::Vector{<:Tuple{<:AbstractVector{<:Float64},<:AbstractVector{<:Float64}}})
5   times=length(T)
6   pends=floor.(phase_ends)
7   events=Int64.(sort(unique(vcat(T,pends))))
8   results=zeros(mc_its,length(events),2)
9
10  nt=Threads.nthreads()
11  Threads.@threads for t in 1:nt
12    t==nt ? (idxs=1+(t-1)*Int(floor(mc_its/nt)):mc_its) :
13      (idxs=(1+(t-1)*Int(floor(mc_its/nt))):t*Int(floor(mc_its/nt)))
14
15    phase=1
16    cycle_time,exit_rate=pparams[1+((phase-1)*2):2+((phase-1)*2)]
17
18    results[idxs,1,1]=rand.(popdist)
19    results[idxs,1,2]=rand.(voldist)
20
21    last_ph_ch=1;
22    next_event=2;
23    while next_event≤length(events)
24      n=events[next_event]-events[last_ph_ch]
25
26      pop_factor=max(eps(),(2^(24/cycle_time)-exit_rate))
27      pop_factor=1. && (pop_factor=pop_factor+eps())
28
29      vol_factor=volconst*exit_rate*(1-pop_factor^(n-1))/(1-pop_factor)
30
31      lastpops=view(results,idxs,last_ph_ch,1)
32      lastvols=view(results,idxs,last_ph_ch,2)
33
34      results[idxs,next_event,1]=max.(min.(lastpops.*pop_factor^n,MAXVAL),
35      results[idxs,next_event,2]=min.(lastvols.+(lastpops.*vol_factor),MAXVAL)

```

```

32         if events[next_event] in pends #after updating pop and vol to
33             → t, update phase parameters for next event, make t, vol,
34             → and pop. the phase change vals
35             phase+=1; last_ph_ch=next_event
36             phase <= length(pparams)/2 &&
37             → ((cycle_time,exit_rate)=pparams[1+((phase-1)*2):2+((phase-1)*2)])
38         end #advance phase if necessary
39
40
41         next_event+=1
42     end
43   end
44
45   tidxs=[findfirst(t→t=time,events) for time in T]
46
47   pop_lns=Vector{LogNormal}(undef,times)
48   vol_lns=Vector{LogNormal}(undef,times)
49   pop_lns[1]=popdist
50   vol_lns[1]=voldist
51
52   Threads.@threads for t in 2:times
53       pop_lns[t]=fit(LogNormal,results[:,tidxs[t],1])
54       vol_lns[t]=fit(LogNormal,results[:,tidxs[t],2])
55   end
56
57   pop_lhs=Vector{Float64}(undef,times-1)
58   vol_lhs=Vector{Float64}(undef,times-1)
59
60   Threads.@threads for t in 1:times-1
61       pop_lhs[t]=lps(logpdf(pop_lns[t+1],obs[t+1][1]))
62       vol_lhs[t]=lps(logpdf(vol_lns[t+1],obs[t+1][2]))
63   end
64
65   log_lh=lps(lps(pop_lhs),lps(vol_lhs))
66
67   disp_mat=zeros(times,3,2)
68   Threads.@threads for t in 1:times
69
70       → disp_mat[t,:,:]=quantile(pop_lns[t],.025),mean(pop_lns[t]),quantile(pop_
71
72       → disp_mat[t,:,:]=quantile(vol_lns[t],.025),mean(vol_lns[t]),quantile(vol_
73   end
74
75   return log_lh, disp_mat

```

---

70 end

---

#### 17.4.4 /src/CMZ\_sim/CMZ\_model.jl

---

```

1 mutable struct CMZ_Record <: GMC_NS_Model_Record
2     trajectory::Int64
3     i::Int64
4     pos::Vector{Float64}
5     path::String
6     log_Li::Float64
7 end
8
9 struct CMZ_Model <: GMC_NS_Model
10    trajectory::Int64
11    i::Int64
12
13    θ::Vector{Float64}
14    log_Li::Float64
15
16    pos::Vector{Float64}
17    v::Vector{Float64}
18
19    disp_mat::Array{Float64} #matrix for mean & 95%CI plot of model
→      output
20 end
21
22 function construct_CMZ(trajectory::Int64, i::Int64, θ::Vector{Float64},
→   pos::Vector{Float64}, v::Vector{Float64},
→   obs::Vector{Tuple{Vector{Float64},Vector{Float64}}}),
→   T::Vector{Float64}, popdist::Distribution, voldist::Distribution,
→   volconst::Float64, mc_its::Int64, phs::Int64; v_init=false)
23   pparams=θ[1:2*phs];phase_ends=θ[2*phs+1:(2*phs+1)+(phs-2)]
24
25   log_lh,disp_mat=CMZ_mc_llh(popdist, voldist, volconst, phase_ends,
→     pparams, mc_its, T, obs)
26
27   v_init && (v=rand(MvNormal(length(θ),1.)))
28
29   CMZ_Model(trajectory, i, θ, log_lh, pos, v, disp_mat)
30 end

```

---

### 17.4.5 /src/cycle\_decay\_sim/decay\_constructor.jl

---

```

1 function construct_decay_slice(trajectory::Int64, i::Int64,
2   ↳ θ::Vector{Float64}, pos::Vector{Float64}, v::Vector{Float64},
2   ↳ obs::Vector{Vector{Float64}}, T::Vector{Float64},
2   ↳ popdist::Distribution, lens_model::Lens_Model, mc_its::Int64;
2   ↳ v_init=false)
2   log_lh,disp_mat=decay_slice_mc_llh(popdist, lens_model, θ, mc_its, T,
2   ↳ obs)
3
4   v_init && (v=rand(MvNormal(length(θ),1.)))
5
6   return Slice_Model(trajectory, i, θ, log_lh, pos, v, disp_mat)
7 end

```

---

### 17.4.6 /src/cycle\_decay\_sim/decay\_ensemble.jl

---

```

1 Decay_Engine(path::String, no_models::Integer,
2   ↳ obs::AbstractVector{<:AbstractVector{<:Float64}},
2   ↳ priors::AbstractVector{<:Distribution}, constants, box, GMC_settings;
2   ↳ sample_posterior::Bool=true) =
2 Slice_Engine(
3   path,
4   construct_decay_slice,
5   assemble_DMs(path, no_models, obs, priors, constants, box) ... ,
6   [-Inf], #L0 = 0
7   [0], #X0 = 1
8   [-Inf], #w0 = 0
9   [-Inf], #Liwi0 = 0
10  [-Inf], #Z0 = 0
11  [0], #H0 = 0,
12  obs,
13  priors,
14  constants, #T, popdist, lens_model, mc_its
15  box,
16  sample_posterior,
17  Vector{Slice_Record}(),
18  GMC_settings ... ,
19  no_models+1)
20
21 function assemble_DMs(path::String, no_trajectories::Integer, obs,
22   ↳ priors, constants, box)

```

```

22     ensemble_records = Vector{Slice_Record}()
23     !isdir(path) && mkpath(path)
24     @showprogress 1 "Assembling Slice_Model ensemble..." for
25         → trajectory_no in 1:no_trajectories
26             model_path = string(path,'/',trajectory_no,'.',1)
27             if !isfile(model_path)
28                 proposal=rand.(priors)
29                 pos=to_unit_ball.(proposal,priors)
30                 box_bound!(pos,box)
31                 θvec=to_prior.(pos,priors)
32
33             model = construct_decay_slice(trajectory_no, 1, θvec, pos,
34                 → [0.], obs, constants...; v_init=true)
35
36             serialize(model_path, model) #save the model to the ensemble
37             → directory
38             push!(ensemble_records, Slice_Record(trajectory_no, 1,
39                 → pos,model_path,model.log_Li))
40
41         else #interrupted assembly pick up from where we left off
42             model = deserialize(model_path)
43             push!(ensemble_records, Slice_Record(trajectory_no, 1,
44                 → model.pos,model_path,model.log_Li))
45
46     end
47
48     return ensemble_records, minimum([record.log_Li for record in
49         → ensemble_records])
50 end

```

---

### 17.4.7 /src/cycle\_decay\_sim/decay\_llh.jl

```

1 function decay_slice_mc_llh(popdist::Distribution, lm::Lens_Model,
2     → dparams::Vector{Float64}, mc_its::Int64, T::Vector{Float64},
3     → obs::Vector{Vector{Float64}})
4     times=length(T)
5     events=Int64.(sort(T))
6     results=zeros(mc_its,length(events))
7
8     nt=Threads.nthreads()
9
10    init_day=events[1]
11    init_ct, decay_const, exit_rate=dparams

```

```

10     cts=[init_ct*exp(decay_const*elapsed) for elapsed in
11         → collect(events[1]:1:events[end])-init_day]
12
13     Threads.@threads for t in 1:nt
14         t==nt ? (idxs=1+(t-1)*Int(floor(mc_its/nt)):mc_its) :
15             → (idxs=(1+(t-1)*Int(floor(mc_its/nt))):t*Int(floor(mc_its/nt)))
16
17         results[idxs,1] *= rand.(popdist)
18
19         day=init_day; next_event=2
20         curr_pops=copy(results[idxs,1,1])
21         while day ≤ events[end]
22             days_elapsed = day - init_day
23             cycle_time=cts[day-init_day+1]
24             pop_factor=max(eps(),(2^(24/cycle_time)-exit_rate))
25             circexit=circumferential_exit(lm,day-1,day,curr_pops)
26
27             → curr_pops=max.(min.(curr_pops.*pop_factor,MAXVAL)--circexit,eps())
28
29             day in events[2:end] && (
30                 results[idxs,next_event] *= curr_pops;
31                 next_event+=1)
32             day+=1
33         end
34     end
35
36     tidxs=[findfirst(t→t==time,events) for time in T]
37
38     pop_lns=Vector{LogNormal}(undef,times)
39     pop_lns[1]=popdist
40     Threads.@threads for t in 2:times
41         pop_lns[t]=fit(LogNormal,results[:,tidxs[t]])
42     end
43
44     pop_lhs=Vector{Float64}(undef,times-1)
45
46     Threads.@threads for t in 1:times-1
47         pop_lhs[t]=lps(logpdf(pop_lns[t+1],obs[t+1]))
48     end
49
50     log_lh=lps(pop_lhs)

```

```

50     disp_mat=zeros(times,3)
51     Threads.@threads for t in 1:times
52
53         → disp_mat[t,:]=[quantile(pop_lns[t],.025),mean(pop_lns[t]),quantile(pop_lns[t],.975)]
54     end
55
56     return log_lh, disp_mat
57 end

```

---

### 17.4.8 /src/multislice\_sim/multislice\_ensemble.jl

```

1 mutable struct MultiSlice_Ensemble <: GMC_NS_Ensemble
2     path::String
3
4     model_initλ::Function
5     models::Vector{Slice_Record}
6
7     contour::Float64
8     log_Li::Vector{Float64}
9     log_Xi::Vector{Float64}
10    log_wi::Vector{Float64}
11    log_Liwi::Vector{Float64}
12    log_Zi::Vector{Float64}
13    Hi::Vector{Float64}
14
15    obs::AbstractVector{<:AbstractVector{<:AbstractVector{<:Float64}}}
16    priors::Vector{<:Distribution}
17    constants::Vector{<:Any}
18    #T, popdist, lens_model, mc_its, phs
19    box::Matrix{Float64}
20
21    sample_posterior::Bool
22    posterior_samples::Vector{Slice_Record}
23
24    GMC_Nmin::Int64
25
26    GMC_τ_death::Float64
27    GMC_init_τ::Float64
28    GMC_tune_μ::Int64
29    GMC_tune_α::Float64
30    GMC_tune_PID::NTuple{3,Float64}
31

```

```

32     GMC_timestep_η::Float64
33     GMC_reflect_η::Float64
34     GMC_exhaust_σ::Float64
35     GMC_chain_κ::Int64
36
37     t_counter::Int64
38 end
39
40 MultiSlice_Ensemble(path::String, no_models::Integer,
41   ↵ obs::AbstractVector{<:AbstractVector{<:AbstractVector{<:Float64}}}},
42   ↵ priors::AbstractVector{<:Distribution}, constants, box, GMC_settings;
43   ↵ sample_posterior::Bool=true) =
44 MultiSlice_Ensemble(
45   path,
46   construct_multislice,
47   assemble_MSMS(path, no_models, obs, priors, constants, box) ... ,
48   [-Inf], #L₀ = 0
49   [0], #X₀ = 1
50   [-Inf], #w₀ = 0
51   [-Inf], #Liwi₀ = 0
52   [-Inf], #Z₀ = 0
53   [0], #H₀ = 0,
54   obs,
55   priors,
56   constants, #T, popdist, lens_model, mc_its, phs
57   box,
58   sample_posterior,
59   Vector{Slice_Record}(),
60   GMC_settings ... ,
61   no_models+1)
62
63 MultiSlice_Decay_Ensemble(path::String, no_models::Integer,
64   ↵ obs::AbstractVector{<:AbstractVector{<:AbstractVector{<:Float64}}}},
65   ↵ priors::AbstractVector{<:Distribution}, constants, box, GMC_settings;
66   ↵ sample_posterior::Bool=true) =
67 MultiSlice_Ensemble(
68   path,
69   construct_decay_multislice,
70   assemble_MSDMs(path, no_models, obs, priors, constants, box) ... ,
71   [-Inf], #L₀ = 0
72   [0], #X₀ = 1
73   [-Inf], #w₀ = 0
74   [-Inf], #Liwi₀ = 0

```

```

69 [-Inf], #Z0 = 0
70 [0], #H0 = 0,
71 obs,
72 priors,
73 constants, #T, popdist, lens_model, mc_its
74 box,
75 sample_posterior,
76 Vector{Slice_Record}(),
77 GMC_settings ... ,
78 no_models+1)

79
80 function assemble_MSMS(path::String, no_trajectories::Integer, obs,
81   priors, constants, box)
82   ensemble_records = Vector{Slice_Record}()
83   !isdir(path) && mkpath(path)
84   phs=constants[5]
85   @showprogress 1 "Assembling Slice_Model ensemble..." for
86     trajectory_no in 1:no_trajectories
87       model_path = string(path,'/',trajectory_no,'.',1)
88       if !isfile(model_path)
89         proposal=rand.(priors)
90
91         proposal[2*phs+1:(2*phs+1)+(phs-2)]=sort(proposal[2*phs+1:(2*phs+1)+(
92
93           pos=to_unit_ball.(proposal,priors)
94           box_bound!(pos,box)
95           θvec=to_prior.(pos,priors)
96
97           model = construct_multislice(trajectory_no, 1, θvec, pos,
98             [0.], obs, constants...; v_init=true)
99
100          serialize(model_path, model) #save the model to the ensemble
101          directory
102          push!(ensemble_records, Slice_Record(trajectory_no, 1,
103            pos,model_path,model.log_Li))
104        else #interrupted assembly pick up from where we left off
105          model = deserialize(model_path)
106          push!(ensemble_records, Slice_Record(trajectory_no, 1,
107            model.pos,model_path,model.log_Li))
108        end
109      end
110    end
111  end

```

```

104     return ensemble_records, minimum([record.log_Li for record in
105         ↪ ensemble_records])
106 end
107
108 function assemble_MSDMs(path::String, no_trajectories::Integer, obs,
109     ↪ priors, constants, box)
110     ensemble_records = Vector{Slice_Record}()
111     !isdir(path) && mkpath(path)
112     @showprogress 1 "Assembling Slice_Model ensemble..." for
113         ↪ trajectory_no in 1:no_trajectories
114         model_path = string(path,'/',trajectory_no,'.',1)
115         if !isfile(model_path)
116             proposal=rand.(priors)
117             pos=to_unit_ball.(proposal,priors)
118             box_bound!(pos,box)
119             θvec=to_prior.(pos,priors)
120
121             model = construct_decay_multislice(trajectory_no, 1, θvec,
122                 ↪ pos, [0.], obs, constants ... ; v_init=true)
123
124             serialize(model_path, model) #save the model to the ensemble
125                 ↪ directory
126             push!(ensemble_records, Slice_Record(trajectory_no, 1,
127                 ↪ pos,model_path,model.log_Li))
128         else #interrupted assembly pick up from where we left off
129             model = deserialize(model_path)
130             push!(ensemble_records, Slice_Record(trajectory_no, 1,
131                 ↪ model.pos,model_path,model.log_Li))
132         end
133     end
134
135     return ensemble_records, minimum([record.log_Li for record in
136         ↪ ensemble_records])
137 end
138
139 function Base.show(io::IO, m::MultiSlice_Model, e::MultiSlice_Ensemble;
140     ↪ progress=false)
141     T=e.constants[1]
142
143     println("MultiSlice_Model $(m.trajectory).$(m.i)")
144
145     rows=0
146     for (n,(obs, slice)) in enumerate(zip(e.obs,m.slices))

```

```

138     catpobs=vcat([obs[t] for t in 1:length(T)] ... )
139     ymax=max(maximum(slice.disp_mat[:, :]),maximum(catpobs))
140     ymin=min(minimum(slice.disp_mat[:, :]),minimum(catpobs))
141
142     plt=lineplot(T,slice.disp_mat[:, 2],title="Slice
143     ↪ $n",color=:green,name="μ pop", ylim=[ymin,ymax])
144     lineplot!(plt,T,slice.disp_mat[:, 1],color=:magenta,name="95% CI")
145     lineplot!(plt,T,slice.disp_mat[:, 3],color=:magenta)
146
147     Ts=vcat([[T[n] for i in 1:length(obs[n])] for n in
148     ↪ 1:length(T)] ...)
149     scatterplot!(plt,Ts, catpobs, color=:yellow, name="Obs")
150
151     rows+=nrows(plt.graphics)
152     show(io, plt)
153     println()
154
155     println()
156     println("log_Li: $(m.log_Li)")
157     println("θ: $(m.θ)")
158     println("v: $(m.v)")
159
160     progress ? (return rows+14) : (return)
161 end

```

---

### 17.4.9 /src/multislice\_sim/multislice\_model.jl

```

1 struct MultiSlice_Model <: GMC_NS_Model
2     trajectory::Int64
3     i::Int64
4
5     θ::Vector{Float64}
6     log_Li::Float64
7
8     pos::Vector{Float64}
9     v::Vector{Float64}
10
11    slices::Vector{Slice_Model}
12 end
13

```

```

14 function construct_multislice(trajecotry::Int64, i::Int64,
15   ↳ θ::Vector{Float64}, pos::Vector{Float64}, v::Vector{Float64},
16   ↳ obs::Vector{Vector{Vector{Float64}}}, T::Vector{Float64},
17   ↳ popdists::Vector{<:Distribution}, lens_model::Lens_Model,
18   ↳ mc_its::Int64, phs::Int64; v_init=false)
19   pparams=θ[1:2*phs];phase_ends=θ[2*phs+1:(2*phs+1)+(phs-2)]
20
21
22 log_lh=0.
23
24
25 slices=Vector{Slice_Model}()
26 for (popdist,pobs) in zip(popdists, obs)
27   llh,disp_mat=slice_mc_llh(popdist, lens_model, phase_ends,
28     ↳ pparams, mc_its, T, pobs)
29   log_lh+=llh
30
31
32 push!(slices,Slice_Model(trajecotry, i, θ, log_lh, pos, v,
33   ↳ disp_mat))
34 end
35
36 v_init && (v=rand(MvNormal(length(θ),1.)))
37
38
39 return MultiSlice_Model(trajecotry, i, θ, log_lh, pos, v, slices)
40 end
41
42
43 function construct_decay_multislice(trajecotry::Int64, i::Int64,
44   ↳ θ::Vector{Float64}, pos::Vector{Float64}, v::Vector{Float64},
45   ↳ obs::Vector{Vector{Vector{Float64}}}, T::Vector{Float64},
46   ↳ popdists::Vector{<:Distribution}, lens_model::Lens_Model,
47   ↳ mc_its::Int64; v_init=false)
48   log_lh=0.
49
50
51 slices=Vector{Slice_Model}()
52 for (popdist,pobs) in zip(popdists, obs)
53   llh,disp_mat=decay_slice_mc_llh(popdist, lens_model, θ, mc_its,
54     ↳ T, pobs)
55   log_lh+=llh
56
57
58 push!(slices,Slice_Model(trajecotry, i, θ, log_lh, pos, v,
59   ↳ disp_mat))
60 end
61
62 v_init && (v=rand(MvNormal(length(θ),1.)))
63
64

```

---

```

45     return MultiSlice_Model(trajectory, i, θ, log_lh, pos, v, slices)
46 end

```

---

### 17.4.10 /src/slice\_sim/lens\_model.jl

```

1 struct Lens_Model
2     factor::Float64
3     power::Float64
4     section::Float64
5 end
6
7 function circumferential_exit(lm,t1,t2,pops)
8     t1c=lm.factor*(t1^lm.power) #lens circumference at time 1, according
9         ↪ to lens model
10    t2c=lm.factor*(t2^lm.power) #lens circumference at time 2
11    nc=t2c-t1c #new circumference grown btw t1 and t2
12
13    sliceno=t1c/lm.section #number of slices implied by lm's section size
14        ↪ for t1 lens diameter
15    psg=nc/sliceno #amount of circumferential growth each CMZ slice is
16        ↪ responsible for (per slice growth)
17    return (psg/lm.section).*pops #per slice growth divided by slice
18        ↪ section thickness supplies the implied number of RPCs leaving CMZ
19        ↪ to supply growing retina
20 end
21
22 function Base.length(lm::Lens_Model)
23     return 1
24 end

```

---

### 17.4.11 /src/slice\_sim/slice\_ensemble.jl

---

```

1 mutable struct Slice_Ensemble <: GMC_NS_Ensemble
2     path::String
3
4     model_initλ::Function
5     models::Vector{Slice_Record}
6
7     contour::Float64
8     log_Li::Vector{Float64}

```

```

 9      log_Xi::Vector{Float64}
10      log_wi::Vector{Float64}
11      log_Liwi::Vector{Float64}
12      log_Zi::Vector{Float64}
13      Hi::Vector{Float64}

14
15      obs::AbstractVector{<:AbstractVector{<:Float64}}
16      priors::Vector{<:Distribution}
17      constants::Vector{<:Any}
18      #T, popdist, lens_model, mc_its, phs
19      box::Matrix{Float64}

20
21      sample_posterior::Bool
22      posterior_samples::Vector{Slice_Record}

23
24      GMC_Nmin::Int64

25
26      GMC_tau_death::Float64
27      GMC_init_tau::Float64
28      GMC_tune_mu::Int64
29      GMC_tune_alpha::Float64
30      GMC_tune_PID::NTuple{3,Float64}

31
32      GMC_timestep_eta::Float64
33      GMC_reflect_eta::Float64
34      GMC_exhaust_sigma::Float64
35      GMC_chain_K::Int64

36
37      t_counter::Int64
38 end
39
40 Slice_Engine(path::String, no_models::Integer,
41   ← obs::AbstractVector{<:AbstractVector{<:Float64}},
42   ← priors::AbstractVector{<:Distribution}, constants, box, GMC_settings;
43   ← sample_posterior::Bool=true) =
44 Slice_Engine(
45   path,
46   construct_slice,
47   assemble_SMS(path, no_models, obs, priors, constants, box) ... ,
48   [-Inf], #L0 = 0
49     [0], #X0 = 1
50     [-Inf], #w0 = 0
51     [-Inf], #Liwi0 = 0

```

```

49 [-Inf], #Z0 = 0
50 [0], #H0 = 0,
51 obs,
52 priors,
53 constants, #T, popdist, lens_model, mc_its, phs
54 box,
55 sample_posterior,
56 Vector{Slice_Record}(),
57 GMC_settings ... ,
58 no_models+1)

59
60 function assemble_SMs(path::String, no_trajectories::Integer, obs,
61   ↪ priors, constants, box)
62   ensemble_records = Vector{Slice_Record}()
63   !isdir(path) && mkpath(path)
64   phs=constants[5]
65   @showprogress 1 "Assembling Slice_Model ensemble..." for
66     ↪ trajectory_no in 1:no_trajectories
67     model_path = string(path,'/',trajectory_no,'.',1)
68     if !isfile(model_path)
69       proposal=rand.(priors)
70
71       ↪ proposal[2*phs+1:(2*phs+1)+(phs-2)]=sort(proposal[2*phs+1:(2*phs+1)+(
72
73         pos=to_unit_ball.(proposal,priors)
74         box_bound!(pos,box)
75         θvec=to_prior.(pos,priors)
76
77         model = construct_slice(trajectory_no, 1, θvec, pos, [0.],
78           ↪ obs, constants ... ; v_init=true)
79
80         serialize(model_path, model) #save the model to the ensemble
81         ↪ directory
82         push!(ensemble_records, Slice_Record(trajectory_no, 1,
83           ↪ pos,model_path,model.log_Li))
84       else #interrupted assembly pick up from where we left off
85         model = deserialize(model_path)
86         push!(ensemble_records, Slice_Record(trajectory_no, 1,
87           ↪ model.pos,model_path,model.log_Li))
88       end
89     end
90   end

```

```

84     return ensemble_records, minimum([record.log_Li for record in
85         ensemble_records])
86
87
88 function Base.show(io::IO, m::Slice_Model, e::Slice_Ensemble;
89     progress=false)
90     T=e.constants[1]
91
92     catpobs=vcat([e.obs[t] for t in 1:length(T)]...)
93     ymax=max(maximum(m.disp_mat[:, :]), maximum(catpobs))
94     ymin=min(minimum(m.disp_mat[:, :]), minimum(catpobs))
95
96     plt=lineplot(T,m.disp_mat[:, 2], title="Slice_Model
97     $(m.trajectory).$(m.i), log_Li $(m.log_Li)", color=:green, name="μ
98     pop", ylim=[ymin,ymax])
99     lineplot!(plt,T,m.disp_mat[:, 1], color=:magenta, name="95% CI")
100    lineplot!(plt,T,m.disp_mat[:, 3], color=:magenta)
101
102    Ts=vcat([[T[n] for i in 1:length(e.obs[n])] for n in 1:length(T)]...)
103    scatterplot!(plt,Ts, catpobs, color=:yellow, name="Obs")
104
105    show(io, plt)
106    println()
107    println("θ: $(m.θ)")
108    println("v: $(m.v)")

109    (progress && return nrows(plt.graphics)+16);
110 end

```

---

### 17.4.12 /src/slice\_sim/slice\_lh.jl

---

```

1 function slice_mc_llh(popdist::Distribution, lm::Lens_Model,
2     phase_ends::Vector{Float64}, pparams::Vector{Float64}, mc_its::Int64,
3     T::Vector{Float64}, obs::Vector{Vector{Float64}})
4     times=length(T)
5     pends=floor.(phase_ends)
6     events=Int64.(sort(unique(vcat(T,pends))))
7     results=zeros(mc_its,length(events))

8     nt=Threads.nthreads()
9     Threads.@threads for t in 1:nt

```

```

9      t=nt ? (idxs=1+(t-1)*Int(floor(mc_its/nt)):mc_its) :
10     ↵ (idxs=(1+(t-1)*Int(floor(mc_its/nt))):t*Int(floor(mc_its/nt)))

11    phase=1
12    cycle_time,exit_rate=pparams[1+((phase-1)*2):2+((phase-1)*2)]
13
14    results[idxs,1]=rand.(popdist)
15
16    next_event=2;
17    while next_event ≤ length(events)
18        n=events[next_event]-events[next_event-1]
19
20        pop_factor=max(eps(),(2^(24/cycle_time)-exit_rate))
21
22        lastpops=view(results,idxs,next_event-1)
23
24        results[idxs,next_event]=min.(lastpops.*pop_factor^n,MAXVAL)
25
26
27        ↵ results[idxs,next_event]=max.(view(results,idxs,next_event))-circumf
28
29        if events[next_event] in pends #after updating pop and vol to
30            ↵ t, update phase parameters for next event, make t。 vol,
31            ↵ and pop。 the phase change vals
32            phase+=1;
33            phase ≤ length(pparams)/2 &&
34            ↵ ((cycle_time,exit_rate)=pparams[1+((phase-1)*2):2+((phase-1)*2)])
35        end #advance phase if necessary
36
37        next_event+=1
38    end
39
40    end
41
42    tidxs=[findfirst(t→t=time,events) for time in T]
43
44    pop_lns=Vector{LogNormal}(undef,times)
45    pop_lns[1]=popdist
46    Threads.@threads for t in 2:times
47        pop_lns[t]=fit(LogNormal,results[:,tidxs[t]])
48    end
49
50    pop_lhs=Vector{Float64}(undef,times-1)

```

```

47     Threads.@threads for t in 1:times-1
48         pop_lhs[t]=lps(logpdf(pop_lns[t+1],obs[t+1]))
49     end
50
51     log_lh=lps(pop_lhs)
52
53     disp_mat=zeros(times,3)
54     Threads.@threads for t in 1:times
55
56         → disp_mat[t,:]=[quantile(pop_lns[t],.025),mean(pop_lns[t]),quantile(pop_lns[t],.975)]
57     end
58
59     return log_lh, disp_mat
60 end

```

---

### 17.4.13 /src/slice\_sim/slice\_model.jl

```

1 struct Slice_Record <: GMC_NS_Model_Record
2     trajectory :: Int64
3     i :: Int64
4     pos :: Vector{Float64}
5     path :: String
6     log_Li :: Float64
7 end
8
9 struct Slice_Model <: GMC_NS_Model
10    trajectory :: Int64
11    i :: Int64
12
13    θ :: Vector{Float64}
14    log_Li :: Float64
15
16    pos :: Vector{Float64}
17    v :: Vector{Float64}
18
19    disp_mat :: Array{Float64} #matrix for mean & 95%CI plot of model
20    → output
21 end

```

```

22 function construct_slice(trajecotry::Int64, i::Int64, θ::Vector{Float64},
23   pos::Vector{Float64}, v::Vector{Float64},
24   obs::Vector{Vector{Float64}}, T::Vector{Float64},
25   popdist::Distribution, lens_model::Lens_Model, mc_its::Int64,
26   phs::Int64; v_init=false)
27   pparams=θ[1:2*phs];phase_ends=θ[2*phs+1:(2*phs+1)+(phs-2)]
28
29   log_lh,disp_mat=slice_mc_llh(popdist, lens_model, phase_ends,
30     pparams, mc_its, T, obs)
31
32   v_init && (v=rand(MvNormal(length(θ),1.)))
33
34   return Slice_Model(trajecotry, i, θ, log_lh, pos, v, disp_mat)
35 end

```

---

#### 17.4.14 /src/thymidine\_sim/thymidine\_cell.jl

```

1 abstract type AbstractCell end
2
3 mutable struct LabelCell <: AbstractCell
4   time::Float64
5   tδ::Float64
6   g1::Float64
7   s::Float64
8   label::Float64
9 end

```

---

#### 17.4.15 /src/thymidine\_sim/thymidine\_ensemble.jl

```

1 mutable struct Thymidine_Ellement <: GMC_NS_Ellement
2   path::String
3
4   model_initλ::Function
5   models::Vector{Thymidine_Record}
6
7   contour::Float64
8   log_Li::Vector{Float64}
9   log_Xi::Vector{Float64}
10  log_wi::Vector{Float64}
11  log_Liwi::Vector{Float64}
12  log_Zi::Vector{Float64}

```



```

53     constants, #popdist, T, pulse, mc_its, end_time
54     box,
55     sample_posterior,
56     Vector{Thymidine_Record}(),
57     GMC_settings ... ,
58     no_models+1)
59
60 function assemble_TMs(path::String, no_trajectories::Integer, obs,
61   ↪ priors, constants, box)
62   ensemble_records = Vector{Thymidine_Record}()
63   !isdir(path) && mkpath(path)
64   @showprogress 1 "Assembling Thymidine_Model ensemble..." for
65   ↪ trajectory_no in 1:no_trajectories
66   model_path = string(path,'/',trajectory_no,'.',1)
67   if !isfile(model_path)
68     proposal=rand.(priors)
69
70     pos=to_unit_ball.(proposal,priors)
71     box_bound!(pos,box)
72     θvec=to_prior.(pos,priors)
73
74     model = thymidine_constructor(trajectory_no, 1, θvec, pos,
75       ↪ [0.], obs, constants... ; v_init=true)
76
77     serialize(model_path, model) #save the model to
78     ↪ the ensemble directory
79     push!(ensemble_records,
80       ↪ Thymidine_Record(trajectory_no, 1,
81       ↪ pos,model_path,model.log_Li))
82   else #interrupted assembly pick up from where we left off
83     model = deserialize(model_path)
84     push!(ensemble_records,
85       ↪ Thymidine_Record(trajectory_no, 1,
86       ↪ model.pos,model_path,model.log_Li))
87   end
88 end
89
90 return ensemble_records, minimum([record.log_Li for record in
91   ↪ ensemble_records])
92 end
93
94 function Base.show(io::IO, m::Thymidine_Model, e::Thymidine_Engine;
95   ↪ progress=false)

```

```

86   T=e.constants[2]
87
88   catobs=vcat(e.obs ... )
89   ymax=max(maximum(m.disp_mat),maximum(catobs))
90   ymin=min(minimum(m.disp_mat),minimum(catobs))
91
92   plt=lineplot(T,m.disp_mat[:,2],title="Thymidine_Model
93   ← $(m.trajectory).$(m.i), log_Li $(m.log_Li)",color=:green,name="μ
94   ← count", ylim=[ymin,ymax])
95   lineplot!(plt,T,m.disp_mat[:,1],color=:magenta,name="95% CI")
96   lineplot!(plt,T,m.disp_mat[:,3],color=:magenta)
97
98   Ts=vcat([[T[n] for i in 1:length(e.obs[n])] for n in 1:length(T)] ... )
99   scatterplot!(plt,Ts, catobs, color=:yellow, name="Obs")
100
101   show(io, plt)
102   println()
103   println("θ: $(m.θ)")
104 end
105
106 function print_MAP_output(e::Thymidine_Engsemble, path::String=e.path;
107   ← its::Int64=Int64(1e6), size=(900,600), clims=(0.,.05))
108   MLEmod=deserialize(e.models[findmax([m.log_Li for m in
109   ← e.models])[2]].path)
110   θ=MLEmod.θ
111   lt=length(e.priors)
112
113   n_pops=1
114   lt>5 && (n_pops+=(lt-5)/6)
115
116   pparams=Vector{Tuple{LogNormal, Float64, Float64, Float64}}(){}
117   for pop in 1:n_pops
118     tcμ, tcσ², g1_frac, s_frac, sis_frac=
119     ← θ[Int64(1+((pop-1)*5)):Int64(5*pop)]
120     tcσ=sqrt(tcσ²);
121     push!(pparams,(LogNormal(tcμ,tcσ),g1_frac,s_frac,sis_frac))
122   end
123
124   n_pops > 1 ? (pop_fracs=θ[Int64(5*n_pops)+1:end]) : (pop_fracs[])
125
126   popdist, T, pulse, mc_its, end_time = e.constants

```

```

124
125     log_llh, disp_mat, joint_DNPs=thymidine_mc_llh(popdist, pop_fracs,
→       pparams, its, end_time, pulse, T, e.obs, true)
126
127     max_ct=0
128     for DNP in joint_DNPs
129         max_ct<maximum(DNP.support) && (max_ct=maximum(DNP.support))
130     end
131     cts=LinRange(0,max_ct,max_ct+1)
132     probs=zeros(length(T),length(cts))
133     for (t,DNP) in enumerate(joint_DNPs)
134         for (ct, p) in zip(DNP.support,DNP.p)
135             ct_idx=findfirst(isequal(ct),cts)
136             ct_idx ≡ nothing && (probs[t,ct_idx]=p)
137         end
138     end
139     catobs=vcat(e.obs ... )
140     ymax=Int64(round(maximum(catobs)*1.1))
141     ys=[0,ymax]
142     Ts=vcat([[T[n] for i in 1:length(e.obs[n])] for n in 1:length(T)] ... )
143
144
145     → MAPout=Plots.heatmap(T,cts,transpose(probs),ylims=ys,color=:viridis,size=size
→       xlabel="Post-pulse chase time (hr)", ylabel="Labelled cells (#)",
→       colorbar_title="Probability mass", legend=:topleft,
→       foreground_color_legend=nothing, background_color_legend=nothing,
→       legendfontcolor=:white, xticks=T)
146     Plots.scatter!(MAPout,Ts,catobs,marker=:cross,markercolor=:magenta,
→       label="Observations")
147     Plots.plot!(MAPout, T, disp_mat[:,2], color=:white, width=2,
→       label="Simulated count mean")
148     Plots.plot!(MAPout, T, disp_mat[:,1], color=:white, style=:dash,
→       label="Simulated count 95% probability mass")
149     Plots.plot!(MAPout, T, disp_mat[:,3], color=:white, style=:dash,
→       label=:none)
150
151     savefig(MAPout, path)
152   end
153
154   function print_marginals(e::Thymidine_Ensemble, path::String;
→     param_names=["LogNormal Tc μ", "LogNormal Tc σ²", "G1 Fraction", "S
→     Fraction", "Sister Shift Fraction"])
→     wtvec=zeros(0)

```

```

155     lt=length(e.priors)
156     θvecs=zeros(0) for param in 1:lt]
157     for (n,rec) in enumerate(e.posterior_samples)
158         m=deserialize(rec.path)
159         for i in 1:length(θvecs)
160             push!(θvecs[i],m.θ[i])
161         end
162         push!(wtvec,e.log_Liwi[n+1])
163     end
164
165     for rec in e.models
166         m=deserialize(rec.path)
167         for i in 1:length(θvecs)
168             push!(θvecs[i],m.θ[i])
169         end
170         → push!(wtvec,(lps(m.log_Li,lps(e.log_Xi[end],-log(length(e.models)))))))
171     end
172
173     n_pops=1
174     lt>5 && (n_pops+=Int64((lt-5)/6))
175     n_pops>1 ? (plotrows=6; og_pn=copy(param_names)) : (plotrows=5)
176
177     for pop in 2:n_pops
178         param_names=vcat(param_names,vcat(og_pn,"Population $pop
179             → Fraction"))
180     end
181
182     mapm=deserialize(e.models[findmax([m.log_Li for m in
183         → e.models])[2]].path)
184
185     plots=Vector{Plots.Plot}()
186     for (n,θvec) in enumerate(θvecs)
187         p_label=param_names[n]
188         occursin("Fraction",p_label) ? (xls=[0,1]) :
189             → (xls=[quantile(e.priors[n],.01),quantile(e.priors[n],.99)])
190
191         θkde=kde(θvec,weights=exp.(wtvec))
192         n==1 ? (lblpos=:right) : (lblpos=:none)
193         plt=StatsPlots.plot(e.priors[n], color=:darkmagenta, fill=true,
194             → fillalpha=.5, label="Prior", xlabel=p_label,
195             → ylabel="Density", xlims=xls)

```

```

192     plot!(θkde.x,θkde.density, color=:green, fill=true,
193           ← fillalpha=.75, label="Posterior")
194
195     ← plot!([mapm.θ[n],mapm.θ[n]],[0,maximum(θkde.density)],color=:black,
196           ← label="MAP", legend=lblpos)
197     push!(plots,plt)
198     n_pops>1 && n==5 && push!(plots,plot())
199
200   end
201
202   combined=plot(plots ... ,
203   ← layout=grid(plotrows,n_pops),size=(600*n_pops,800))
204
205   savefig(combined, path)
206 end

```

---

#### 17.4.16 /src/thymidine\_sim/thymidine\_lh.jl

```

1 function thymidine_mc_llh(pop_dist, pop_fracs, pparams, mc_its, end_time,
2   ← pulse, T::Vector{<:AbstractFloat},
3   ← obs::Vector{<:AbstractVector{<:Integer}}, display=false)
4   n_pops=length(pparams); times=length(T)
5   popset_labelled=zeros(Int64,mc_its,n_pops,times)
6   max_labelled=maximum.(obs)
7
8   Threads.@threads for it in 1:mc_its
9     pop_lins=zeros(Int64,n_pops)
10    pop_lins[1]=Int64(max(1,round(rand(pop_dist))))
11    for p in 2:n_pops
12      pop_lins[p]=Int64(round(pop_fracs[p-1]*pop_lins[p-1]))
13      pop_lins[p-1]--=pop_lins[p]
14    end
15
16    for p in 1:n_pops
17      tc_dist,g1_frac,s_frac,sis_frac = pparams[p]
18      plv=view(popset_labelled,it,p,:)
19
20      for l in 1:Int64(floor(pop_lins[p]/2))
21        !all(max_labelled .<
22          ← sum(popset_labelled[it,:,:],dims=1)[1,:]) &&
23          ← sim_lin_pair!(plv, T, end_time, pulse, tc_dist,
24          ← g1_frac, s_frac, sis_frac)
25    end

```

```

21     for l in 1:pop_lins[p]%
22         !all(max_labelled .<
23             → sum(popset_labelled[it,:,:,:],dims=1)[1,:]) &&
24             → sim_lineage!(plv, T, end_time, pulse, tc_dist,
25             → g1_frac, s_frac, sis_frac)
26     end
27 end
28
29 pop_DNPs=Matrix{DiscreteNonParametric}(undef,n_pops,times)
30 for p in 1:n_pops
31     Threads.@threads for t in 1:times
32
33         → pop_DNPs[p,t]=fit(DiscreteNonParametric,popset_labelled[:,p,t])
34     end
35 end
36
37 if n_pops > 1
38     joints=Vector{DiscreteNonParametric}(undef,times)
39     Threads.@threads for t in 1:times
40         joints[t]=joint_DNP_sum(pop_DNPs[:,t])
41     end
42 else
43     joints=pop_DNPs[1,:]
44 end
45
46 log_lhs=Vector{Float64}(undef,times)
47 Threads.@threads for t in 1:times
48     log_lhs[t]=lps(logpdf(joints[t],obs[t]))
49 end
50
51 log_lh=lps(log_lhs)
52
53 disp_mat=zeros(times,3)
54 Threads.@threads for t in 1:times
55
56     → disp_mat[t,:]=[quantile(joints[t],.025),mean(joints[t]),quantile(joints[t],.975)]
57 end
58 display ? (return log_lh, disp_mat, joints) : (return log_lh,
59     → disp_mat)
60 end

```

```

58 function joint_DNP_sum(cs)
59     npops=length(cs)
60     ct_supports=Vector{Vector{Int64}} }()
61     for c in cs
62         push!(ct_supports,c.support[[c.p ... ].>0.])
63     end
64
65     possible_sums=collect(Iterators.product(ct_supports ... ))
66     possible_totals=unique(sum.(possible_sums))
67     total_mat=sum.(possible_sums)
68
69     probs=zeros(length(possible_totals))
70
71     Threads.@threads for (n,pt) in collect(enumerate(possible_totals))
72         psums=possible_sums[findall(t→t==pt,total_mat)]
73         probs[n]=logsumexp([lps([logpdf(cs[p],ps[p]) for p in 1:npops])
74             ↳ for ps in psums])
75     end
76
77     return DiscreteNonParametric(possible_totals, exp.(probs))
78 end
79
80

```

---

#### 17.4.17 /src/thymidine\_sim/thymidine\_model.jl

```

1 mutable struct Thymidine_Record <: GMC_NS_Model_Record
2     trajectory :: Int64
3     i :: Int64
4     pos :: Vector{Float64}
5     path :: String
6     log_Li :: Float64
7 end
8
9 struct Thymidine_Model <: GMC_NS_Model
10    trajectory :: Int64
11    i :: Int64
12
13    θ :: Vector{Float64}
14    log_Li :: Float64
15

```

```

16     pos::Vector{Float64}
17     v::Vector{Float64}
18
19     disp_mat::Matrix{Float64} #matrix for mean & 95%CI plot of model
20     ↪   output
21 end
22
23 function thymidine_constructor(trajectory, i, θ, pos, v, obs, popdist, T,
24   ↪   pulse, mc_its, end_time; v_init=false)
25   lt=length(θ)
26
27   lt!=5 && mod(lt-5,6)≠0 && throw(ArgumentError("θ must contain tcμ,
28   ↪   tcσ2, g1_frac, s_frac, sis_frac values per lineage population,
29   ↪   and one popfrac per population beyond the first!"))
30   pulse<0 && throw(ArgumentError("pulse length must be ≥0!"))
31
32   n_pops=1
33   lt>5 && (n_pops+=(lt-5)/6)
34
35   pparams=Vector{Tuple{LogNormal, Float64, Float64, Float64}}(){}
36   for pop in 1:n_pops
37     tcμ, tcσ2, g1_frac, s_frac, sis_frac=
38     ↪   θ[Int64(1+((pop-1)*5)):Int64(5*pop)]
39     tcσ=sqrt(tcσ2);
40     push!(pparams,(LogNormal(tcμ,tcσ),g1_frac,s_frac,sis_frac))
41   end
42
43   n_pops > 1 ? (pop_fracs=θ[Int64(5*n_pops)+1:end]) : (pop_fracs[])
44
45   log_lh,disp_mat=thymidine_mc_llh(popdist, pop_fracs, pparams, mc_its,
46   ↪   end_time, pulse, T, obs)
47
48   v_init && (v=rand(MvNormal(length(θ),1.)))
49
50   Thymidine_Model(trajectory, i, θ, log_lh, pos, v, disp_mat)
51 end

```

---

### 17.4.18 /src/thymidine\_sim/thymidine\_sim.jl

---

```

1 DETECTION_THRESHOLD=.15
2 MIN_CT=4.
3

```

```

4 function sim_lin_pair!(plv, T, end_time, pulse, Tc, g1_frac, s_frac,
5   ↪ sis_frac)
6   tδ1, g1_1, s1 = cycle_model(Tc, g1_frac, s_frac)
7   tδ2, g1_2, s2 = (1+rand([-1,1])*sis_frac).*(tδ1, g1_1, s1)
8
9   mint=-min(tδ1,tδ2)
10  mint==0. ? (birth_time=mint) : (birth_time=rand(Uniform(mint,0.)))
11
12  cells1=[LabelCell(birth_time,tδ1,g1_1,s1,0.)]
13  cells2=[LabelCell(birth_time,tδ2,g1_2,s2,0.)]
14
15  for cellvec in [cells1,cells2]
16    ci=1
17    while ci≤length(cellvec)
18      cycle_sim!(plv, ci, cellvec, T, end_time, pulse, Tc,
19        ↪ g1_frac,s_frac, sis_frac)
20      ci+=1
21    end
22  end
23
24 function sim_lineage!(plv, T, end_time, pulse, Tc, g1_frac, s_frac,
25   ↪ sis_frac)
26   tδ1, g1, s1 = cycle_model(Tc, g1_frac, s_frac)
27   birth_time=rand(Uniform(-tδ1,0.))
28   cells=[LabelCell(birth_time,tδ1,g1,s1,0.)]
29
30   ci=1
31   while ci≤ length(cells)
32     cycle_sim!(plv, ci, cells, T, end_time, pulse, Tc, g1_frac,
33       ↪ s_frac, sis_frac)
34     ci+=1
35   end
36 end
37
38 function cycle_sim!(plv, ci::Int64, cells::Vector{LabelCell},
39   ↪ T::Vector{Float64}, end_time::Float64, pulse::Float64, Tc::LogNormal,
40   ↪ g1_frac::Float64, s_frac::Float64, sis_frac::Float64)
41   n_times=length(T);
42
43   cell=cells[ci]
44   t=cell.time; tδ₀=cell.tδ₀; g1₀=cell.g1₀; s₀=cell.s₀; l=cell.label

```

```

41
42 first_cycle=true
43
44 while t<end_time
45   tδ,s=0.,0.
46   first_cycle ? (tδ=tδ;g1=g1;s=s;first_cycle=false) : ((tδ, g1,
47   ↵ s)=cycle_model(Tc, g1_frac, s_frac))
48
49   s_start=t+g1
50   s_end=s_start+s
51   cycle_mitosis=t+tδ
52   cycle_events=[s_start,s_end,cycle_mitosis]
53
54   oldl=l; cycle_label=false
55   if (s_start ≤ pulse && s_end ≥ 0) #some s-phase overlaps with
56   ↵ the pulse in this case
57     cycle_label=true
58     l=update_label(l, s, s_start, s_end, pulse)
59   end
60
61   tidxs=t.<T.<cycle_mitosis
62   n_idxes=sum(tidxs)
63   (length(tidxs)>0 && (oldl > DETECTION_THRESHOLD || l >
64   ↵ DETECTION_THRESHOLD))
65
66   if n_idxes>0 && (oldl > DETECTION_THRESHOLD || l >
67   ↵ DETECTION_THRESHOLD)
68     cycle_label ? (label_outcomes=[oldl,-Inf,l]) :
69     ↵ (label_outcomes=[oldl,oldl,oldl])
70     cubuf=falses(n_idxes)
71     for (n,timept) in enumerate(T[tidxs])
72       count_labelled_at_T!(n,cubuf,timept, cycle_events,
73       ↵ label_outcomes, oldl, s_start, s, pulse)
74     end
75     plv[tidxs]+=cubuf
76   end
77
78   l/=2 #mitosis halves label for the daughters
79
80   tδ2, g1_2, s2 = (1+rand([-1,1])*sis_frac).*(tδ, g1, s)
81   tδ2 < MIN_CT && (tδ2=tδ; g1_2=g1; s2 = s;) #prevent sister shift
82   ↵ from resulting in unrealistically fast proliferation

```

```

77     cycle_mitosis < end_time &&
    ↳ push!(cells,LabelCell(cycle_mitosis,tδ2,g1_2,s2,l))
78     t+=tδ
79   end
80 end
81
82 function cycle_model(Tc, g1_frac, s_frac)
83     tδ=max(MIN_CT,rand(Tc)) #floor of 1 hr for any distribution of cycle
    ↳ times
84     s=tδ * s_frac
85     g1=g1_frac * (tδ - s)
86     return tδ, g1, s
87 end
88
89 function update_label(label, s, s_start, s_end, pulse)
90     s_overlap=min(pulse,s_start+s_end)-max(0,s_start)
91     nuc_label_frac=s_overlap/s
92     return min(nuc_label_frac+label,1.) #label before mitosis time will
    ↳ be what's been accumulated from past + this cycle's s phase
93 end
94
95 function count_labelled_at_T!(n, cubuf, timept, cycle_events,
    ↳ label_outcomes, oldl, s_start, s, pulse)
96     outcome_idx=timept.<cycle_events
97     timept_label=label_outcomes[outcome_idx][1]
98     timept_label== -Inf &&
    ↳ (timept_label=partial_label(timept,oldl,s_start,s,pulse))
99     timept_label ≥ DETECTION_THRESHOLD && (cubuf[n]=true)
100 end
101
102 function partial_label(timept, label, s_start, s, pulse)
103     s_overlap=min(pulse,timept)-max(0.,s_start)
104     nuc_label_frac=s_overlap/s
105     return min(nuc_label_frac+label,1.)
106 end

```

---

## 17.5 BioBackgroundModels

Github repository: <https://github.com/mmattocks/BioBackgroundModels.jl>

### 17.5.1 /src/BioBackgroundModels.jl

---

```

1 module BioBackgroundModels
2
3 import Base:copy,size
4 import Distances: euclidean
5 import Distributions:Univariate,Dirichlet,Categorical,logpdf,isprobvec
6 import Distributed: RemoteChannel, myid, remote_do, rmprocs
7 import HMMBase: AbstractHMM, assert_hmm, istransmat
8 import MCMCChains: Chains, ChainDataFrame, heideldiag
9 import Printf: @sprintf
10 import Random: rand, shuffle
11 import Serialization: serialize
12 import StatsFuns: logsumexp, logaddexp
13 import Statistics: mean
14 import UnicodePlots: lineplot,lineplot!, scatterplot,scatterplot!
15 using BioSequences, DataFrames, FASTX, GFF3, ProgressMeter
16
17 include("BHMM/BHMM.jl")
18 export BHMM
19 include("EM/chain.jl")
20 export Chain_ID, EM_step
21 include("API/genome_sampling.jl")
22 export setup_sample_jobs, execute_sample_jobs
23 include("API/EM_master.jl")
24 export setup_EM_jobs!, execute_EM_jobs!
25
26 include("reports/chain_report.jl")
27 include("reports/partition_report.jl")
28 include("reports/replicate_convergence.jl")
29 include("API/reports.jl")
30 export generate_reports
31
32 include("EM/baum-welch.jl")
33 include("EM/churbanov.jl")
34 include("utilities/load_balancer.jl")
35 export LoadConfig
36 include("EM/EM_converge.jl")
37 include("genome_sampling/partition_masker.jl")
38 include("genome_sampling/sequence_sampler.jl")
39 include("likelihood_funcs/bg_lh_matrix.jl")
40 export BGHMM_likelihood_calc
41 include("likelihood_funcs/hmm.jl")

```

```

42 export obs_lh_given_hmm
43 include("utilities/observation_coding.jl")
44 include("utilities/BBG_analysis.jl")
45 include("utilities/BBG_progressmeter.jl")
46 include("utilities/HMM_init.jl")
47 include("utilities/model_display.jl")
48 include("utilities/utilities.jl")
49 export split_obs_sets
50 include("utilities/log_prob_sum.jl")
51 export lps
52 end # module

```

---

### 17.5.2 /src/API/EM\_master.jl

```

1 """
2     setup_EM_jobs!(job_ids, obs_sets; delta_thresh, chains,
3     ↪ init_function)
4
5 Given a vector of 'Chain_IDs', and the appropriate obs_sets dictionary,
6     ↪ return the appropriate tuple of channels and chains for the
7     ↪ 'execute_EM_jobs!' function. Given an existing chains dict, resumes
8     ↪ any existing non-converged chains (as assessed by the provided
9     ↪ 'delta_thresh' value), otherwise initialises hmms for new chains with
10    ↪ optionally user-specified 'init_function' (default
11    ↪ 'autotransition_init').
12
13 """
14
15 function setup_EM_jobs!(job_ids::Vector{Chain_ID},
16     ↪ obs_sets::Dict{String,Vector{LongSequence{DNAAlphabet{2}}}};
17     ↪ delta_thresh::Float64=1e-3,
18     ↪ chains::Dict{Chain_ID,Vector{EM_step}}=Dict{Chain_ID,Vector{EM_step}}(),
19     ↪ init_function::Function=autotransition_init)
20     #argument checking
21     length(job_ids) < 1 && throw(ArgumentError("Empty job id vector!"))
22     length(obs_sets) < 1 && throw(ArgumentError("Empty observation
23     ↪ sets!"))
24
25     no_input_hmms = length(job_ids)
26     input_channel= RemoteChannel(()→Channel{Tuple}(no_input_hmms*3))
27     ↪ #channel to hold BHMM learning jobs
28     output_channel= RemoteChannel(()→Channel{Tuple}(Inf)) #channel to
29     ↪ take EM iterates off of

```

```

15
16     code_dict = code_job_obs(job_ids, obs_sets)
17
18     @showprogress 1 "Setting up HMMs ... " for id in job_ids #for each
19         ↳ jobid, add an initial BHMM to input_channel for EM_workers
20         if haskey(chains, id) && length(chains[id]) > 0 #true if resuming
21             ↳ from incomplete chain
22             chain_end=chains[id][end]
23             if !chain_end.converged || (chain_end.converged &&
24                 ↳ chain_end.delta > delta_thresh)#push the last hmm iterate
25                 ↳ for nonconverged chains to the input channel with coded
26                 ↳ observations and values for chain resumption
27                 put!(input_channel, (id, chain_end.iterate,
28                     ↳ chain_end.hmm, chain_end.log_p, code_dict[(id.obs_id,
29                         ↳ id.order)]))
30             else #skip any jobs that have converged from previous runs
31                 no_input_hmms -= 1
32         end
33
34     else
35         hmm = init_function(id.K, id.order, id.obs_id) #initialise
36             ↳ first BHMM in chain
37         chains[id] = Vector{EM_step}() #initialise the relevant chain
38         obs=code_dict[(id.obs_id,id.order)]
39         lh=obs_lh_given_hmm(obs,hmm,linear=false)
40         put!(input_channel, (id, 1, hmm, lh, obs))
41     end
42
43     return no_input_hmms, chains, input_channel, output_channel
44 end
45
46 """
47     execute_EM_jobs!(worker_pool, no_input_hmms, chains, input_channel,
48         ↳ output_channel, chains_path; load_dict, EM_func, delta_thresh,
49         ↳ max_iterates, verbose)
50

```

```

41 Given: a vector of worker IDs ([1] to use the master), the splatted
42   output of 'setup_EM_jobs' (no_input_hmms, chains, input_channel,
43   output_channel), and a valid path to save chains ('chains_path');
44   assign chains to workers using 'load_dict' to iteratively execute
45   'EM_func' (default linear_step) until 'delta_thresh' convergence
46   criterion is obtained or 'max_iterates' have been calculated.
47   Specifying 'verbose=true' gives some additional debug output from
48   'EM_converge!'.

49 """
50
51 function execute_EM_jobs!(worker_pool::Vector{Int64},
52   no_input_hmms::Integer, chains::Dict{Chain_ID,Vector{EM_step}},
53   input_channel::RemoteChannel, output_channel::RemoteChannel,
54   chains_path::String; load_dict=Dict{Int64,LoadConfig}(),
55   EM_func::Function=linear_step, delta_thresh=1e-3, max_iterates=5000,
56   verbose=false)
57   #argument checking
58   length(worker_pool) < 1 && throw(ArgumentError("Worker pool must
59   contain one or more worker IDs!"))
60   no_input_hmms < 1 && throw(ArgumentError("Zero input HMMs reported,
61   likely continuing from chains already converged beyond default
62   delta_thresh for setup_EM_jobs"))
63   length(chains) < 1 && throw(ArgumentError("No chains supplied, likely
64   job set from setup_EM_jobs passed incorrectly"))
65   !isready(input_channel) && throw(ArgumentError("BHMM input channel
66   has no contents, likely job set from setup_EM_jobs already
67   executed"))

68
69 #SEND BHMM FIT JOBS TO WORKERS
70 if isready(input_channel) > 0
71   @info "Fitting HMMs.."
72   #WORKERS FIT HMMS
73   for worker in worker_pool
74     if worker in keys(load_dict)
75       remote_do(EM_converge!, worker, input_channel,
76         output_channel, no_input_hmms,
77         load_config=load_dict[worker], EM_func=EM_func,
78         delta_thresh=delta_thresh, max_iterates=max_iterates,
79         verbose=verbose)
80     else
81       remote_do(EM_converge!, worker, input_channel,
82         output_channel, no_input_hmms, EM_func=EM_func,
83         delta_thresh=delta_thresh, max_iterates=max_iterates,
84         verbose=verbose)
85     end
86   end
87 end

```

```

59         end
60     end
61 else
62     @warn "No input HMMs (all already converged?), skipping
63         fitting.."
64 end
65
66 #GET LEARNT HMMS OFF REMOTECHANNEL, SERIALISE AT EVERY ITERATION,
67     → UPDATE PROGRESS METERS
68 job_counter=no_input_hmms
69 learning_meters=Dict{Chain_ID, ProgressHMM}()
70 overall_meter=Progress(no_input_hmms,"Overall batch progress:")
71
72 while job_counter > 0
73     wait(output_channel)
74     workerid, jobid, iterate, hmm, log_p, delta, converged, steptime
75         → = take!(output_channel)
76     #either update an existing ProgressHMM meter or create a new one
77         → for the job
78     if haskey(learning_meters, jobid) && iterate > 2
79         update!(learning_meters[jobid], delta, steptime)
80     else
81         offset = workerid - 1
82         if iterate ≤ 2
83             learning_meters[jobid] = ProgressHMM(delta_thresh,
84                 → "$jobid on Wk $workerid:", offset, 2)
85             update!(learning_meters[jobid], delta, steptime)
86         else
87             learning_meters[jobid] = ProgressHMM(delta_thresh,
88                 → "$jobid on Wk $workerid:", offset, iterate)
89             update!(learning_meters[jobid], delta, steptime)
90         end
91     end
92     #push the hmm and related params to the results_dict
93     push!(chains[jobid], EM_step(iterate, hmm, log_p, delta,
94         → converged))
95     #try to serialize the results; catch interrupts and other errors
96         → to prevent corruption
97     try
98         serialize(chains_path, chains)
99     catch e
100         @warn "Serializing failed!"
101         println(e)

```

```

94     end
95
96     #decrement the job counter, update overall progress meter, and
97     #→ save the current results dict on convergence or max iterate
98     if converged || iterate == max_iterates
99         job_counter -= 1
100        ProgressMeter.update!(overall_meter,
101            → (no_input_hmms-job_counter))
102        if !isready(input_channel) #if there are no more jobs to be
103            → learnt, retire the worker
104            workerid!=1 && rmprocs(workerid)
105        end
106    end
107
108    #count converged & unconverged jobs, report results
109    converged_counter = 0
110    unconverged_counter = 0
111    for (id, chain) in chains
112        chain[end].converged = true ? (converged_counter += 1) :
113            → (unconverged_counter += 1)
114    end
115
116    @info "Background HMM batch learning task complete,
117        → $converged_counter converged jobs, $unconverged_counter jobs
118        → failed to converge in $max_iterates iterates since job start."
119 end

```

---

### 17.5.3 /src/API/genome\_sampling.jl

---

```

1 """
2     setup_sample_jobs(genome_path, genome_index_path, gff3_path,
2         → sample_set_length, sample_window_min, sample_window_max,
2         → perigenic_pad; deterministic)
3

```

```

4 Return sample job channels for 'execute_sample_jobs', given 'genome_path'
→ to a properly formatted FASTA genome, 'genome_index_path' to the
→ associated .FAI, 'gff3_path' to the genome's GFF3 feature database, a
→ total number of bases to sample, 'sample_set_length', as well as
→ 'sample_window_min' and 'sample_window_max' lengths for individual
→ samples. 'perigenic_pad' specifies the number of bases up- and
→ down-stream of exonic sequences to be considered 'perigenic'; if 0,
→ the perigenic partition will be entirely intronic. Optional boolean
→ 'deterministic' may be set to 'true' to always return '+' strand
→ sequences from unstranded samples; otherwise unstranded samples may
→ be returned with either orientation.

5 """
6
7 #function to partition genome and set up Distributed RemoteChannels so
→ partitions can be sampled simultaneously
8 function setup_sample_jobs(genome_path::String,
→ genome_index_path::String, gff3_path::String,
→ sample_set_length::Integer, sample_window_min::Integer,
→ sample_window_max::Integer, perigenic_pad::Integer;
→ deterministic::Bool=false)
9     #argument checking
10    !ispather(genome_path) && throw(ArgumentError("Bad genome path!"))
11    !ispather(genome_index_path) && throw(ArgumentError("Bad genome index
→ path!"))
12    !ispather(gff3_path) && throw(ArgumentError("Bad gff3 path!"))
13    sample_set_length < 1 && throw(ArgumentError("Sample set length must
→ be a positive integer!"))
14    sample_window_min < 1 || sample_window_max < 1 &&
→ throw(ArgumentError("Sample window minimum and maximum bounds
→ must be positive integers!"))
15    sample_window_min ≥ sample_window_max && throw(ArgumentError("Sample
→ window minimum size must be smaller than maximum size"))
16    perigenic_pad < 0 && throw(ArgumentError("Perigenic pad must be 0 or
→ positive!"))

17    coordinate_partitions = partition_genome_coordinates(gff3_path,
→ perigenic_pad)
18    sample_sets = DataFrame[]
19    input_sample_jobs =
→ RemoteChannel(()→Channel{Tuple}(length(coordinate_partitions)))
→ #channel to hold sampling jobs

```



```

49         if n ≤ partitions #no more workers than partitions to be
50             ↪ used
51                 remote_do(get_sample_set, worker, input_sample_channel,
52                             ↪ completed_sample_channel, progress_channel)
53             end
54         else
55             @error "No sampling jobs!"
56         end
57
58     #progress meters for sampling
59     sampling_meters=Dict{String, Progress}()
60     overall_sampling_meter=Progress(partitions,"Overall sampling
61         ↪ progress:")
62     completed_counter = 0
63     ProgressMeter.update!(overall_sampling_meter, completed_counter)
64     sampling_offset = ones(Bool, partitions)
65
66     #collect progress updates while waiting on completion of sampling
67         ↪ jobs
68     while completed_counter < partitions
69         wait(progress_channel)
70         partition_id, progress = take!(progress_channel)
71         if haskey(sampling_meters, partition_id)
72             ProgressMeter.update!(sampling_meters[partition_id],
73                 ↪ progress)
74         else
75             offset = findfirst(sampling_offset)[1]
76             sampling_meters[partition_id] = Progress(sample_set_length,
77                 ↪ "Sampling partition $partition_id:", offset)
78             ProgressMeter.update!(sampling_meters[partition_id],
79                 ↪ progress)
80             sampling_offset[offset] = false
81         end
82         if progress == sample_set_length
83             completed_counter += 1
84             ProgressMeter.update!(overall_sampling_meter,
85                 ↪ completed_counter)
86         end
87     end
88
89     #collect sample dfs by partition id when ready
90     sample_recorddfs = Dict{String,DataFrame}()

```

```

84     collected_counter = 0
85     while collected_counter < partitions
86         wait(completed_sample_channel)
87         partition_id, sample_df = take!(completed_sample_channel)
88         sample_record_dfs[partition_id] = sample_df
89         collected_counter += 1
90         @info "Partition $partition_id completed sampling ... "
91     end
92
93     return sample_record_dfs
94 end

```

---

#### 17.5.4 /src/API/reports.jl

```

1 struct Report_Folder
2     partition_id::String
3     partition_report::Partition_Report
4     replicate_report::Replicate_Report
5     chain_reports::Dict{Chain_ID,Chain_Report}
6 end
7
8 function generate_reports(chains::Dict{Chain_ID,Vector{EM_step}},
9     ↳ test_sets::Dict{String,Vector{LongSequence{DNAAlphabet{2}}}})
9     #check chains dict for problems
10
11    length(chains)==0 && throw(ArgumentError("Empty chains dict!"))
12    any(chain→length(chain)<2, values(chains)) &&
13        ↳ throw(ArgumentError("Some chains are too short (<2 EM steps)!")
14        ↳ "Probably not all chains have been operated on by EM workers yet.
15        ↳ Try EM_converge first!"))
16    unconv=sum(!chain[end].converged for chain in values(chains))
17    unconv > 0 && @warn "Not all chains are converged to the selected
18        ↳ step delta."
19    length(test_sets)==0 && throw(ArgumentError("Empty test_sets dict!"))
20
21    chain_reports = report_chains(chains, test_sets)
22    partition_reports = report_partitions(chain_reports)
23
24    report_folders=Dict{String, Report_Folder}()
25
26    for part_report in partition_reports
27        rep_report=report_replicates(part_report.best_repset, chains)
28    end
29
30    return report_folders
31 end

```

```

24     chain_subset=Dict{Chain_ID,Chain_Report}()
25     for id in keys(chain_reports)
26         id.obs_id==part_report.partition_id &&
27             → (chain_subset[id]=chain_reports[id])
28     end
29
30         → report_folders[part_report.partition_id]=Report_Folder(part_report.partit
31         → part_report, rep_report, chain_subset)
32     end
33
34     return report_folders
35 end

```

### 17.5.5 /src/BHMM/BHMM.jl

```

1 """
2     BHMM([a::AbstractVector{T}, ]A::AbstractMatrix{T},
3         B::AbstractVector{<:Categorical}) where F where T
4 Build an BHMM with transition matrix `A` and observations distributions
5         `B`.
6 If the initial state distribution `a` is not specified, a uniform
7         distribution is assumed.
8 Observations distributions can be of different types (for example
9         `Normal` and `Exponential`).
10 However they must be of the same dimension (all scalars or all
11         multivariates).
12 # Example
13 """
14 """
15 """
16 """
17 struct BHMM{T} <: AbstractHMM{Univariate}
18     a :: AbstractVector{T}
19     A :: AbstractMatrix{T}
20     B :: AbstractVector{<:Categorical}
21     partition :: String
22     BHMM{T}(a, A, B, partition="") where {T} = assert_BHMM(a, A, B) &&
23         new(a, A, B, partition)
24 end
25 """

```

```

20 BHMM(a::AbstractVector{T}, A::AbstractMatrix{T},
21     ↳ B::AbstractVector{<:Categorical}, partition ...) where {T} =
22     ↳ BHMM{T}(a, A, B, partition ...)
23 BHMM(A::AbstractMatrix{T}, B::AbstractVector{<:Categorical},
24     ↳ partition ...) where {T} = BHMM{T}(ones(size(A)[1])/size(A)[1], A, B,
25     ↳ partition ...)
26
27 copy(hmm::BHMM) = BHMM(copy(hmm.a), copy(hmm.A), copy(hmm.B),
28     ↳ hmm.partition)
29 size(hmm::BHMM) = (length(hmm.B), length(hmm.B[1].p))
30
31 """
32     assert_BHMM(a, A, B)
33 Throw an `ArgumentError` if the initial state distribution and the
34     ↳ transition matrix rows does not sum to 1,
35 and if the observations distributions does not have the same dimensions.
36 """
37
38 function assert_BHMM(a::AbstractVector,
39                     A::AbstractMatrix,
40                     B::AbstractVector{<:Categorical})
41     !isprobvec(a) && throw(ArgumentError("Initial state vector a is not a
42         ↳ valid probability vector!"))
43     !istransmat(A) && throw(ArgumentError("Transition matrix A is not
44         ↳ valid!"))
45
46     !all([length(d.p) for d in B] .== length(B[1].p)) &&
47         ↳ throw(ArgumentError("All distributions must have the same
48             ↳ dimensions"))
49     !(length(a) == size(A,1) == length(B)) && throw(ArgumentError("Length
50         ↳ of initial state vector a, dimension of transition matrix A, and
51             ↳ number of distributions B are not the same!"))
52     return true
53 end
54
55
56 function Base.show(io::IO, hmm::BHMM)
57     println(io, "Background BHMM")
58     println(io, "State Initial and Transition Probabilities")
59     print(io, "a: ")
60     show(io, hmm.a)
61     println(io)
62     print(io, "A: ")
63     display(hmm.A)

```

```

51     println(io)
52     println(io, "INFORMATIVE SYMBOLS BY STATE")
53     for (n,d) in enumerate(hmm.B)
54         print(io, "K$n ")
55         print_emitters(d)
56     end
57 end

```

---

### 17.5.6 /src/EM/EM\_converge.jl

```

1 function EM_converge!(hmm_jobs :: RemoteChannel,
2   ↳ output_hmms :: RemoteChannel, no_models :: Integer;
2   ↳ load_config :: LoadConfig=LoadConfig(1:typemax(Int64)-1,
2   ↳ 0:typemax(Int64)-1), EM_func :: Function=linear_step,
2   ↳ delta_thresh=1e-3, max_iterates=5000, verbose=false)
2   while isready(hmm_jobs)
3       workerid = myid()
4       jobid, start_iterate, hmm, job_norm, observations =
5       ↳ load_balancer(no_models, hmm_jobs, load_config)
5       jobid == 0 && break #no valid job for this worker according to
6       ↳ load_table entry
6
7       start_iterate > max_iterates - 1 && throw(ArgumentError("BHMM
8           ↳ chain $jobid is already longer ($start_iterate iterates) than
9           ↳ specified max_iterates!"))
10      curr_iterate = start_iterate
11
12      #mask calculations here rather than ms_mle_step to prevent
13      ↳ recalculation every iterate
14      #build array of observation lengths
15
16      obs_lengths = [findfirst(iszero, observations[o,:])-1 for o in
17      ↳ 1:size(observations,1)] #mask calculations here rather than
18      ↳ mle_step to prevent recalculation every iterate
19      EM_func==bw_step && (observations=transpose(observations))
20
21      start_iterate = 1 && put!(output_hmms, (workerid, jobid,
22          ↳ curr_iterate, hmm, job_norm, 0.0, false, 0.0)); #on the first
23          ↳ iterate return the initial BHMM immediately
24      verbose && @info "Fitting BHMM on Wk $workerid, start iterate
25          ↳ $start_iterate, $jobid with $(size(hmm)[1]) states and
26          ↳ $(size(hmm)[2]) symbols ... "

```

```

18
19     curr_iterate += 1
20
21     start=time()
22     new_hmm, last_norm = EM_func(hmm, observations, obs_lengths)
23     delta = abs(lps(job_norm, -last_norm))
24     put!(output_hmms, (workerid, jobid, curr_iterate, new_hmm,
25                         → last_norm, delta, false, time()-start))
26
27     for i in curr_iterate:max_iterates
28         start=time()
29         new_hmm, norm = EM_func(new_hmm, observations, obs_lengths)
30         curr_iterate += 1
31         delta = abs(lps(norm, -last_norm))
32         if delta < delta_thresh
33             put!(output_hmms, (workerid, jobid, curr_iterate,
34                         → new_hmm, norm, delta, true, time()-start))
35             verbose && @info "$jobid converged after"
36             → $(curr_iterate-1) EM steps"
37             break
38         else
39             put!(output_hmms, (workerid, jobid, curr_iterate,
40                         → new_hmm, norm, delta, false, time()-start))
41             verbose && @info "$jobid EM step $(curr_iterate-1) delta"
42             → $delta"
43             last_norm = norm
44         end
45     end
46 end
47 end

```

---

### 17.5.7 /src/EM/baum-welch.jl

---

```

1 """
2     ms_mle_step(hmm::AbstractHMM{F}, observations) where F
3
4 Perform one step of the EM (Baum-Welch) algorithm.
5
6 # Example
7 ````julia
8 hmm, log_likelihood = ms_mle_step(hmm, observations)
9 ````
```

```

10 """
11
12 function bw_step(hmm::BHMM, observations, obs_lengths)
13     lls = bw_llhs(hmm, observations)
14     log_α = messages_forwards_log(hmm.a, hmm.A, lls, obs_lengths)
15     log_β = messages_backwards_log(hmm.A, lls, obs_lengths)
16     log_A = log.(hmm.A)
17
18     K,Tmaxplus1,O = size(lls) #the last T value is the 0 end marker of
19     → the longest T
20
21     #transforms to cut down log_ξ, log_γ assignment times
22     lls = permutedims(lls, [2,3,1]) # from (K,T,O) to (T,O,K)
23     log_α = permutedims(log_α, [2,3,1])
24     log_β = permutedims(log_β, [2,3,1])
25
26     # E-step
27     log_ξ = fill(-Inf, Tmaxplus1,O,K,K)
28     log_γ = fill(-Inf, Tmaxplus1,O,K)
29     log_pobs = zeros(O)
30
31     @inbounds for o = 1:O
32         log_pobs[o] = logsumexp(lps.(log_α[1,o,:], log_β[1,o,:]))
33     end
34
35     Threads.@threads for idx in [(i,j,o) for i=1:K, j=1:K, o=1:O]
36         i,j,o = idx[1],idx[2],idx[3]
37         obsl = obs_lengths[o]
38         @inbounds for t = 1:obsl-1 #log_ξ & log_γ calculated to T-1 for
39             → each o
40             log_ξ[t,o,i,j] = lps(log_α[t,o,i], log_A[i,j], log_β[t+1,o,j],
41             → lls[t+1,o,j], -log_pobs[o])
42             log_γ[t,o,i] = lps(log_α[t,o,i], log_β[t,o,i], -log_pobs[o])
43         end
44         t=obsl #log_ξ @ T = 0
45         log_ξ[t,o,i,j] = 0
46         log_γ[t,o,i] = lps(log_α[t,o,i], log_β[t,o,i], -log_pobs[o])
47     end
48
49     ξ = similar(log_ξ)
50     ξ .= exp.(log_ξ)
51     Σk_ξ = sum(ξ, dims=[3,4])
52     nan_mask = Σk_ξ .= 0

```

```

50     Σk_ξ[nan_mask] .= Inf #prevent NaNs in dummy renorm
51     ξ .= Σk_ξ #dummy renorm across K to keep numerical creep from
      ↳ causing isprobvec to fail on new new_A during hmm creation
52
53     γ = similar(log_γ)
54     γ .= exp.(log_γ)
55     Σk_γ = sum(γ, dims=3)
56     Σk_γ[nan_mask[:, :, :]] .= Inf #prevent NaNs in dummy renorm
57     γ .= Σk_γ #dummy renorm
58
59     # M-step
60     new_A = zeros(K, K)
61     for i=1:K, j=1:K
62         Σotξ_vec = zeros(0)
63         Σotγ_vec = zeros(0)
64         Threads.@threads for o in 1:O
65             Σotξ_vec[o] = sum(ξ[1:obs_lengths[o]-1,o,i,j])
66             Σotγ_vec[o] = sum(γ[1:obs_lengths[o]-1,o,i])
67         end
68         new_A[i,j] = sum(Σotξ_vec) / sum(Σotγ_vec)
69     end
70     new_A .= sum(new_A, dims=2) #dummy renorm
71     new_a = (sum(γ[1,:,:], dims=1)./sum(sum(γ[1,:,:], dims=1)))[1,:]
72     new_a .= sum(new_a) #dummy renorm
73
74     obs_mask = .!nan_mask
75     obs_collection = observations[obs_mask[:, :]]
76
77     B = Categorical[]
78     @inbounds for (i, d) in enumerate(hmm.B)
79         γ_d = γ[:, :, i]
80         push!(B, t_categorical_mle(Categorical, d.support[end],
81           ↳ obs_collection, γ_d[obs_mask[:, :]]))
82     end
83
84     return typeof(hmm)(new_a, new_A, B), lps(log_pobs)
85 end
86
87     function bw_llhs(hmm, observations)
88         lls = zeros(length(hmm.B), size(observations)...)
89         Threads.@threads for d in 1:length(hmm.B)
90             lls[d, :, :] = logpdf.(hmm.B[d], observations)
91         end
92         return lls

```

```

91         end
92 #Multisequence competent log implementations of forward
93     ↳ and backwards algos
94     function messages_forwards_log(init_distn, trans_matrix,
95         ↳ log_likelihoods, obs_lengths)
96         log_alphas = zeros(size(log_likelihoods))
97         log_trans_matrix = log.(trans_matrix)
98         log_alphas[:,1,:] = log.(init_distn) .+
99             ↳ log_likelihoods[:,1,:]
100        Threads.@threads for o in 1:size(log_likelihoods)[3]
101            @inbounds for t in 2:obs_lengths[o], j in
102                ↳ 1:size(log_likelihoods)[1]
103                    ↳ log_alphas[j,t,o]=logsumexp(log_alphas[:,t-1,o]
104                        ↳ .+ log_trans_matrix[:,j]) +
105                        ↳ log_likelihoods[j,t,o]
106                end
107            end
108            return log_alphas
109        end
110
111        function messages_backwards_log(trans_matrix,
112            ↳ log_likelihoods, obs_lengths)
113            log_betas = zeros(size(log_likelihoods))
114            log_trans_matrix = log.(trans_matrix)
115            Threads.@threads for o in 1:size(log_likelihoods)[3]
116                @inbounds for t in obs_lengths[o]-1:-1:1
117                    tmp = view(log_betas, :, t+1, o) .+
118                        ↳ view(log_likelihoods, :, t+1, o)
119                    @inbounds for i in 1:size(log_likelihoods)[1]
120                        log_betas[i,t,o] =
121                            ↳ logsumexp(view(log_trans_matrix, i,:)
122                                ↳ .+ tmp)
123                    end
124                end
125            end
126            return log_betas
127        end
128 #subfunc derived from Distributions.jl categorical.jl
129     ↳ fit_mle, threaded
130     function t_categorical_mle(::Type{<:Categorical},
131         ↳ k::Integer, x::AbstractArray{T}, w::AbstractArray{F})
132         ↳ where T<:Integer where F<:AbstractFloat

```

```

120
    ↵ Categorical(t_pnormalize!(t_add_categorical_counts!(zeros(k),
    ↵ x, w)))
121 end
122
123 t_pnormalize!(v::Vector) = (v ./= sum(v); v)
124
125 function t_add_categorical_counts!(h::Vector{F},
    ↵ x::AbstractArray{T}, w::AbstractArray{F}) where
    ↵ T<:Integer where F<:AbstractFloat
        n = length(x)
        if n ≠ length(w)
            throw(DimensionMismatch("Inconsistent array
                ↵ lengths."))
        end
        hlock = ReentrantLock()
        Threads.@threads for i=1:n
            @inbounds xi = x[i]
            @inbounds wi = w[i]
            lock(hlock)
            h[xi] += wi    # cannot use @inbounds, as no
                ↵ guarantee that x[i] is in bound
            unlock(hlock)
        end
        return h
126
127
128
129
130
131
132
133
134
135
136
137
138
139

```

## 17.5.8 /src/EM/chain.jl

```
1 struct Chain_ID
2     obs_id::AbstractString
3     K::Integer
4     order::Integer
5     replicate::Integer
6     Chain_ID(obs_id,K,order,replicate) =
7         ↳ assert_chain_id(K,order,replicate) &&
8             ↳ new(obs_id,K,order,replicate)
9 end
10
11 function assert_chain_id(K, order, replicate)
12     K < 1 && throw(ArgumentError("Chain_ID K (# of hmm states) must be a
13         ↳ positive integer!"))
14 end
```

```

11     order < 0 && throw(ArgumentError("Chain_ID symbol order must be zero
12         ↪ or a positive integer!"))
13     replicate < 1 && throw(ArgumentError("Chain_ID replicate must be a
14         ↪ positive integer!"))
15     return true
16 end
17
18 struct EM_step
19     iterate::Integer
20     hmm::BHMM
21     log_p::AbstractFloat
22     delta::AbstractFloat
23     converged::Bool
24
25     ↪ EM_step(iterate,hmm,log_p,delta,converged)=assert_step(iterate,hmm,log_p)
26     ↪ && new(iterate, hmm, log_p, delta, converged)
27 end
28
29 function assert_step(iterate, hmm, log_p)
30     iterate < 1 && throw(ArgumentError("EM_step iterate number must be a
31         ↪ positive integer!"))
32     assert_hmm(hmm.a, hmm.A, hmm.B)
33     log_p > 0.0 && throw(ArgumentError("EM_step log probability value
34         ↪ must be 0 or negative!"))
35     return true
36 end

```

---

### 17.5.9 /src/EM/churbanov.jl

---

```

1 function linear_step(hmm, observations, obs_lengths)
2     O,T = size(observations);
3     a = transpose(log.(hmm.a)); A = log.(hmm.A) #less expensive to
4         ↪ transpose transmatrix in the backwards_sweep than to transpose
5         ↪ here and take a ranged view ie view(A,m:m,:) is more expensive
6         ↪ than transpose(view(A,m,:))
7     N = length(hmm.B); Γ = length(hmm.B[1].support);
8     mask=observations.≠0
9     #INITIALIZATION
10    βoi_T = zeros(O,N); βoi_t = zeros(O,N) #log betas at T initialised as
11        ↪ zeros
12    Eoyim_T = fill(-Inf,O,Γ,N,N); Eoyim_t = fill(-Inf,O,Γ,N,N)
13    for m in 1:N, i in 1:N, y in 1:Γ, o in 1:O

```

```

10     observations[o, obs_lengths[o]] = y && m == i && (Eoyim_T[o, y,
11         ↵ i, m] == 0)
12 end
13 Tijm_T = fill(-Inf, 0, N, N, N); Tijm_t = fill(-Inf, 0, N, N, N) #Ti,j(T,m) =
14     ↵ 0 for all m; in logspace
15
16 #RECURRENCE
17
18     ↵ βoi_T, Tijm_T, Eoyim_T=backwards_sweep!(hmm, A, N, Γ, βoi_T, βoi_t, Tijm_T, Tijm_t, Eoy
19
20 #TERMINATION
21 lls = c_llhs(hmm, observations[:, 1])
22 α1om = lps.lls, a) #first position forward msgs
23 log_pobs = [c_lse(lps.(α1om[o, :], βoi_T[o, :])) for o in 1:N]
24
25 Toij = [c_lse([lps(Tijm_T[o, i, j, m], α1om[o, m], -log_pobs[o]) for m in
26     ↵ 1:N]) for o in 1:N, i in 1:N, j in 1:N] #terminate Tijs with
27     ↵ forward messages
28 Eoiy = [c_lse([lps(Eoyim_T[o, y, i, m], α1om[o, m], -log_pobs[o]) for m
29     ↵ in 1:N]) for o in 1:N, i in 1:N, y in 1:Γ] #terminate Eids with
30     ↵ forward messages
31
32 #INTEGRATE ACROSS OBSERVATIONS AND SOLVE FOR NEW BHMM PARAMS
33 #INITIAL STATE DIST
34 a_o=α1om.+βoi_T.-c_lse.(eachrow(α1om.+βoi_T)) #estimate a for each o
35 obs_penalty=log(0) #broadcast subtraction to normalise log prob vals
36     ↵ by obs number
37 new_a=c_lse.(eachcol(a_o))-obs_penalty #sum over obs and normalise
38     ↵ by number of obs
39 #TRANSITION MATRIX
40 new_A = [c_lse(view(Toij, :, i, j)) for i in 1:N, j in
41     ↵ 1:N]-c_lse.(eachcol(c_lse.([view(Toij, o, i, :) for o in 1:N, i in
42     ↵ 1:N])))
43 #EMISSION MATRIX
44 new_b=[c_lse(view(Eoiy, :, i, y)) for i in 1:N, y in
45     ↵ 1:Γ]-c_lse.(eachcol(c_lse.([view(Eoiy, o, i, :) for o in 1:N, i in
46     ↵ 1:N])))
47 new_B=[Categorical(exp.(new_b[i, :])) for i in 1:N]
48
49 return BHMM(exp.(new_a), exp.(new_A), new_B, hmm.partition),
50     ↵ lps(log_pobs)
51 end
52
53 #LINEAR_STEP SUBFUNCS

```

```

39      function backwards_sweep!(hmm, A, N, Γ, βoi_T, βoi_t,
40      ↵ Tijm_T, Tijm_t, Eoyim_T, Eoyim_t, observations, mask,
41      ↵ obs_lengths)
42      for t in maximum(obs_lengths)-1:-1:1
43          last_β=copy(βoi_T)
44          lls = c_llhs(hmm,observations[:,t+1])
45          omask = findall(mask[:,t+1])
46          βoi_T[omask,:] .+= view(lls,omask,:)
47          Threads.@threads for m in 1:N
48              βoi_t[omask,m] =
49                  ↵ c_lse.(eachrow(view(βoi_T,omask,:).+transpose(view(A,
50                  ↵ Tijm_t[omask, i, j, m] .=
51                  ↵ c_lse.(eachrow(lps.(view(Tijm_T,omask,i,j,:),
52                  ↵ view(lls,omask,:),
53                  ↵ transpose(view(A,m,:))))))
54                  i==m && (Tijm_t[omask, i, j, m] .=
55                  ↵ logaddexp.(Tijm_t[omask, i, j, m],
56                  ↵ lps.(last_β[omask,j],
57                  ↵ A[m,j],lls[omask,j])))
58              end
59              for i in 1:N, y in 1:Γ
60                  Eoyim_t[omask, y, i, m] .=
61                      ↵ c_lse.(eachrow(lps.(view(Eoyim_T,omask,y,i,:)),view(
62                      ↵ if i==m
63                          symmask =
64                              ↵ findall(observations[:,t]==y)
65                          Eoyim_t[symmask, y, i, m] .=
66                              ↵ logaddexp.(Eoyim_t[symmask, y, i,
67                              ↵ m], βoi_t[symmask,m]))
68                      end
69                  end
70              end
71          end
72          βoi_T=copy(βoi_t); Tijm_T=copy(Tijm_t); Eoyim_T =
73              ↵ copy(Eoyim_t);
74          end
75          return βoi_T, Tijm_T, Eoyim_T
76      end
77
78      #logsumexp with single dispatch for speed
79      function c_lse(X::AbstractArray{T}; dims=:) where
80          {T<:Real}
81          u=maximum(X)

```

```

67         u isa AbstractArray || isnan(u) || return float(u)
68         return u + log(sum(x .-> exp(x-u), X))
69     end
70
71     #log likelihoods
72     function c_llhs(hmm, observation)
73         lls = zeros(length(observation),length(hmm.B))
74         Threads.@threads for d in 1:length(hmm.B)
75             lls[:,d] = logpdf.(hmm.B[d], observation)
76         end
77         return lls
78     end

```

### 17.5.10 /src/genome\_sampling/partition\_masker.jl

```
1 function get_partition_code_dict(dict_forward::Bool=true)
2     if dict_forward == true
3         return partition_code_dict =
4             Dict("intergenic"⇒1, "periexonic"⇒2, "exon"⇒3)
5     else
6         return code_partition_dict = Dict(1⇒"intergenic",
7             2⇒"periexonic", 3⇒"exon")
8     end
9 end
10
11 function make_padded_df(position_fasta::String, gff3_path::String,
12     genome_path::String, genome_index_path::String, pad::Integer)
13     position_reader = FASTA.Reader(open((position_fasta), "r"))
14     genome_reader = open(FASTA.Reader, genome_path,
15         index=genome_index_path)
16     scaffold_df = build_scaffold_df(gff3_path)
17     position_df = DataFrame(SeqID = String[], Start=Int[], End=Int[],
18         PadSeq = LongSequence[], PadStart=Int[], RelStart=Int[],
19         SeqOffset=Int[])
20     scaffold_seq_dict = build_scaffold_seq_dict(genome_path,
21         genome_index_path)
22
23     for entry in position_reader
24         scaffold = FASTA.identifier(entry)
25
26         if scaffold ≠ "MT"
27             desc_array = split(FASTA.description(entry))
```

```

21         pos_start = parse(Int, desc_array[2])
22         pos_end = parse(Int, desc_array[4])
23         scaffold_end = scaffold_df.End[findfirst(isequal(scaffold),
24                                         ↪ scaffold_df.SeqID)]
25
26         pad_start=max(1,pos_start-pad)
27         pad_length= pos_start - pad_start
28         seq_offset = pad - pad_length
29         padded_seq = fetch_sequence(scaffold, scaffold_seq_dict,
30                                       ↪ pad_start, pos_end, '+')
31
32         if !hasambiguity(padded_seq)
33             push!(position_df, [scaffold, pos_start, pos_end,
34                               ↪ padded_seq, pad_start, pad_length, seq_offset])
35         end
36     end
37
38     close(position_reader)
39     close(genome_reader)
40     return position_df
41 end
42
43 function add_partition_masks!(position_df::DataFrame, gff3_path::String,
44                             ↪ perigenic_pad::Integer=500,
45                             ↪ columns::Tuple{Symbol,Symbol,Symbol}=(:SeqID, :PadSeq, :PadStart))
46     partitions=["exon", "periexonic", "intergenic"]
47     partition_coords_dict = partition_genome_coordinates(gff3_path,
48                                         ↪ perigenic_pad)
49     partitioned_scaffolds =
50         ↪ divide_partitions_by_scaffold(partition_coords_dict)
51     maskcol = [zeros(Int64,0,0) for i in 1:size(position_df,1)]
52     position_df.MaskMatrix=maskcol
53
54     @Threads.threads for entry in eachrow(position_df)
55         scaffold = entry[columns[1]]
56         maskLength = length(entry[columns[2]])
57         seqStart = entry[columns[3]]
58
59         scaffold_coords_dict = Dict{String,DataFrame}()
60
61         for ((partition, part_scaffold), df) in partitioned_scaffolds
62             if scaffold == part_scaffold

```

```

57         scaffold_coords_dict[partition] = df
58     end
59   end
60
61   entry.MaskMatrix=mask_sequence_by_partition(maskLength, seqStart,
62       ↳ scaffold_coords_dict)
63 end
64
65 #add_partition_masks!() SUBFUNCTIONS
66 function
67   ↳ divide_partitions_by_scaffold(partition_coords_dict::Dict{String,
68       ↳ DataFrame})
69   scaffold_coords_dict = Dict{Tuple{String, String}, DataFrame}(),
70   for (partition_id, partition_df) in
71     ↳ partition_coords_dict
72     for scaffold_subframe in groupby(partition_df,
73         ↳ :SeqID)
74       scaffold_id = scaffold_subframe.SeqID[1]
75       scaffold_df = copy(scaffold_subframe)
76       scaffold_coords_dict[(partition_id,
77           ↳ scaffold_id)] = scaffold_df
78     end
79   end
80   return scaffold_coords_dict
81 end
82
83 function mask_sequence_by_partition(maskLength::Integer,
84   ↳ seqStart::Integer, scaffold_coords_dict::Dict{String,
85       ↳ DataFrame})
86   partition_code_dict = get_partition_code_dict()
87   seqMask = zeros(Integer, (maskLength, 2))
88   position = seqStart
89   while position ≤ seqStart+maskLength
90     position_partition, partition_extent,
91       ↳ position_strand =
92       ↳ find_position_partition(position,
93           ↳ scaffold_coords_dict)
94
95     partition_code =
96       ↳ partition_code_dict[position_partition]
97   mask_position = position - seqStart + 1

```

```

87
88         ↵ seqMask[mask_position:min(maskLength,mask_position
89             + partition_extent),1] *= partition_code
90
91         if position_strand == '+'
92
93             ↵ seqMask[mask_position:min(maskLength,mask_position
94                 + partition_extent),2] *= 1
95         elseif position_strand == '-'
96
97             ↵ seqMask[mask_position:min(maskLength,mask_position
98                 + partition_extent),2] *= -1
99         else
100
101             ↵ seqMask[mask_position:min(maskLength,mask_position
102                 + partition_extent),2] *= 0
103         end
104
105     position += partition_extent + 1
106 end
107
108     return seqMask
109 end
110
111
112 function find_position_partition(position::Integer,
113     ↵ partition_dict::Dict{String, DataFrame})
114     foundPos = false
115     position_partition_id = ""
116     three_prime_extent = 0
117     sample_strand = 0
118     for (partition_id, partition) in partition_dict
119         hitindex = findfirst(x→x ≥ position,
120             ↵ partition.End)
121         if hitindex ≠ nothing
122             if position ≥ partition.Start[hitindex]
123                 foundPos=true
124                 position_partition_id = partition_id
125                 three_prime_extent =
126                     ↵ partition.End[hitindex] - position
127                 if partition_id == "exon" || partition_id
128                     ↵ == "perixonic"
129                     sample_strand =
130                         ↵ partition.Strand[hitindex]
131             end
132         end
133     end
134
135     return position_partition_id, three_prime_extent, sample_strand
136 end

```

```

117         end
118     end
119   end
120
121   if foundPos == false
122     throw(DomainError("Position $position not found
123                         → among partition coordinates!"))
124   else
125     return position_partition_id, three_prime_extent,
126                         → sample_strand
127   end
128
129 #function to partition a genome into coordinate sets of:
130 #merged exons
131 #"periexonic" sequences (genes with 5' and 3' boundaries projected
132   ← -/+perigenic_pad bp, minus exons) - includes promoter elements,
133   ← introns, 3' elements
134 #intergenic sequences (everything else)
135 #given a valid gff3
136
137 function partition_genome_coordinates(gff3_path::String,
138   ← perigenic_pad::Integer=500)
139   # construct dataframes of scaffolds and metacoordinates
140   scaffold_df = build_scaffold_df(gff3_path)
141
142   #partition genome into intragenic, periexonic, and exonic coordinate
143   ← sets
144   #assemble exonic featureset
145   exon_df = DataFrame(SeqID = String[], Start = Integer[], End =
146     ← Integer[], Strand=Char[])
147   build_feature_df!(gff3_path, "CDS", "MT", exon_df)
148
149   #project exon coordinates onto the scaffold bitwise, merging
150   ← overlapping features and returning the merged dataframe
151   merged_exon_df = DataFrame(SeqID = String[], Start = Integer[], End =
152     ← Integer[], Strand=Char[])
153   @showprogress 1 "Partitioning exons..." for scaffold_subframe in
154     ← DataFrames.groupby(exon_df, :SeqID) # for each scaffold subframe
155     ← that has exon features
156     scaffold_id = scaffold_subframe.SeqID[1] #get the scaffold id
157     scaffold_bitarray = init_scaffold_bitarray(scaffold_id,
158       ← scaffold_df, false, stranded=true) #init a stranded bitarray
159       ← of scaffold length

```

```

147      ↵ project_features_to_bitarray!(scaffold_subframe,scaffold_bitarray)
148      ↵ #project features as Trues on bitarray of falses
149 merged_subframe = get_feature_df_from_bitarray(scaffold_id,
150     ↵ scaffold_bitarray) #get a feature df from the projected
151     ↵ bitarray
152 append!(merged_exon_df,merged_subframe) #append the merged
153     ↵ scaffold df to the overall merged df
154 end
155
156 #assemble gene featureset
157 gene_df = DataFrame(SeqID = String[], Start = Integer[], End =
158     ↵ Integer[], Strand = Char[])
159 build_feature_df!(gff3_path, "gene", "MT", gene_df)
160
161 perigenic_pad > 0 && add_pad_to_coordinates!(gene_df, scaffold_df,
162     ↵ perigenic_pad) #if a perigenic pad is specified (to capture
163     ↵ promoter/downstream elements etc in the periexonic set), apply it
164     ↵ to the gene coords
165
166 #build intergenic coordinate set by subtracting gene features from
167     ↵ scaffold bitarrays
168 intergenic_df = DataFrame(SeqID = String[], Start = Integer[], End =
169     ↵ Integer[])
170 @showprogress 1 "Partitioning intergenic regions ... " for
171     ↵ scaffold_subframe in DataFrames.groupby(scaffold_df, :SeqID) #
172     ↵ for each scaffold subframe that has exon features
173 scaffold_id = scaffold_subframe.SeqID[1] #get the scaffold id
174 scaffold_bitarray = init_scaffold_bitarray(scaffold_id,
175     ↵ scaffold_df, true) #init a bitarray of scaffold length
176 if any(isequal(scaffold_id),gene_df.SeqID) #if any genes on the
177     ↵ scaffold
178 scaffold_genes=gene_df[findall(isequal(scaffold_id),
179     ↵ gene_df.SeqID), :] #get the gene rows by finding the
180     ↵ scaffold_id
181
182     ↵ subtract_features_from_bitarray!(scaffold_genes,scaffold_bitarray)
183     ↵ #subtract the gene positions from the scaffold bitarray
184 end
185 intragenic_subframe = get_feature_df_from_bitarray(scaffold_id,
186     ↵ scaffold_bitarray) #get a feature df from the projected
187     ↵ bitarray

```

```

168     append!(intergenic_df,intragenic_subframe) #append the merged
169     → scaffold df to the overall merged df
170 end
171
172 #build periexonic set by projecting gene coordinates onto the
173 → scaffold bitwise, then subtracting the exons
174 periexonic_df = DataFrame(SeqID = String[], Start = Integer[], End =
175 → Integer[], Strand=Char[])
176 @showprogress 1 "Partitioning periexonic regions ... " for
177 → gene_subframe in DataFrames.groupby(gene_df, :SeqID) # for each
178 → scaffold subframe that has exon features
179 scaffold_id = gene_subframe.SeqID[1] #get the scaffold id
180 scaffold_bitarray = init_scaffold_bitarray(scaffold_id,
181 → scaffold_df, false, stranded=true) #init a bitarray of
182 → scaffold length
183 project_features_to_bitarray!(gene_subframe,scaffold_bitarray)
184 → #project features as Trues on bitarray of falses
185 if any(isequal(scaffold_id),merged_exon_df.SeqID)
186 scaffold_exons=merged_exon_df[findall(isequal(scaffold_id),
187 → merged_exon_df.SeqID), :]
188
189 → subtract_features_from_bitarray!(scaffold_exons,scaffold_bitarray)
190 end
191 periexonic_subframe = get_feature_df_from_bitarray(scaffold_id,
192 → scaffold_bitarray) #get a feature df from the projected
193 → bitarray
194 append!(periexonic_df,periexonic_subframe) #append the merged
195 → scaffold df to the overall merged df
196 end
197
198 return Dict("exon"⇒merged_exon_df, "periexonic"⇒periexonic_df,
199 → "intergenic"⇒intergenic_df)
200 end
201
202 #partition_genome_coordinates() SUBFUNCTIONS
203 #BITARRAY SCAFFOLD REPRESENTATION SUBFUNCTIONS
204 function init_scaffold_bitarray(scaffold_id::String,
205 → scaffold_df, value::Bool; stranded::Bool=false)
206 scaffold_length =
207 → scaffold_df.End[findfirst(isequal(scaffold_id),
208 → scaffold_df.SeqID)]
209 if stranded

```

```

193     value ? (return rscaffold_bitarray =
194         ↵ trues(scaffold_length,2)) : (return
195         ↵ rscaffold_bitarray = falses(scaffold_length,2))
196     else
197         value ? (return rscaffold_bitarray =
198             ↵ trues(scaffold_length,1)) : (return
199                 ↵ rscaffold_bitarray = falses(scaffold_length,1))
200     end
201 end

202
203 function
204     → project_features_to_bitarray!(scaffold_feature_sf::SubDataFrame,
205         → scaffold_bitarray::BitArray)
206         @inbounds for item in eachrow(scaffold_feature_sf)
207             scaffold_bitarray[item.Start:item.End,1] = [true for
208                 → base in item.Start:item.End]
209             if size(scaffold_bitarray)[2] == 2 #if the bitarray
210                 → is stranded
211                 if item.Strand == '+'
212                     scaffold_bitarray[item.Start:item.End,2] =
213                         → [true for base in item.Start:item.End]
214             end
215         end
216     end
217 end

218
219 function
220     → subtract_features_from_bitarray!(scaffold_feature_sf::DataFrame,
221         → scaffold_bitarray::BitArray)
222         @inbounds for item in eachrow(scaffold_feature_sf)
223             scaffold_bitarray[item.Start:item.End,1] = [false for
224                 → base in item.Start:item.End]
225         end
226     end
227 end

228
229 function get_feature_df_from_bitarray(scaffold_id::String,
230     → scaffold_bitarray::BitArray)
231     size(scaffold_bitarray)[2] == 2 ? scaffold_feature_df =
232         → DataFrame(SeqID = String[], Start = Integer[], End =
233             → Integer[], Strand=Char[]) : scaffold_feature_df =
234             → DataFrame(SeqID = String[], Start = Integer[], End =
235                 → Integer[])

```

```

219         new_feature_start = findnext(view(scaffold_bitarray,:,:1),
220                                     ↵ 1)
220     while new_feature_start ≠ nothing # while new features
221         ↵ are still found on the bitarray
221         if size(scaffold_bitarray)[2] = 2 #if stranded,
222             ↵ get strand info
222             scaffold_bitarray[new_feature_start,2] = 1 ?
223                 ↵ new_feature_strand = '+' :
223                 ↵ new_feature_strand = '-'
223             end
224         if
225             ↵ findnext(!eval,view(scaffold_bitarray,:,:1),new_feature_start)
225             ↵ ≠ nothing
225             new_feature_end =
226                 ↵ findnext(!eval,view(scaffold_bitarray,:,:1),new_feature_start)
226                 ↵ #find next false after feature start and
226                 ↵ subtract 1 for feature end
226         else
227             new_feature_end = size(scaffold_bitarray)[1]
227                 ↵ #if none is found, the end of the feature
227                 ↵ is the end of hte scaffold
227         end
228     size(scaffold_bitarray)[2] = 2 ?
229         ↵ push!(scaffold_feature_df,[scaffold_id,
229          ↵ new_feature_start, new_feature_end,
229          ↵ new_feature_strand]) :
229         ↵ push!(scaffold_feature_df,[scaffold_id,
229          ↵ new_feature_start, new_feature_end]) #push
229          ↵ stranded feature info as appropriate
230     new_feature_start =
231         ↵ findnext(view(scaffold_bitarray,:,:1),
231         ↵ new_feature_end+1)
231     end
232     return scaffold_feature_df
233 end

```

---

### 17.5.11 /src/genome\_sampling/sequence\_sampler.jl

---

```

1 #####SAMPLING FUNCTIONS#####
2 #function for a Distributed worker to produce a set of samples of given
2   ↵ parameters from genomic sequences

```

```

3 function get_sample_set(input_sample_jobs :: RemoteChannel,
4   ↵ completed_sample_jobs :: RemoteChannel,
5   ↵ progress_updates :: RemoteChannel)
6   ↵ while isready(input_sample_jobs)
7     genome_path, genome_index_path, partition_df, partitionid,
8       ↵ sample_set_length, sample_window_min, sample_window_max,
9       ↵ deterministic = take!(input_sample_jobs)
10
11    stranded :: Bool = get_strand_dict()[partitionid]
12    scaffold_sequence_record_dict :: Dict{String, LongSequence} =
13      ↵ build_scaffold_seq_dict(genome_path, genome_index_path)
14
15    sample_df =
16      ↵ DataFrame(SampleScaffold=String[], SampleStart=Integer[], SampleEnd=Integer[])
17    metacoordinate_bitarray = trues(partition_df.MetaEnd[end])
18    sample_set_counter = 0
19
20    while sample_set_counter < sample_set_length #while we don't yet
21      ↵ have enough sample sequence
22        sample_scaffold :: String, sample_Start :: Integer,
23          ↵ sample_End :: Integer, sample_MetaStart :: Integer,
24          ↵ sample_MetaEnd :: Integer, sample_sequence :: LongSequence,
25          ↵ strand :: Char = get_sample(metacoordinate_bitarray,
26          ↵ sample_window_min, sample_window_max, partition_df,
27          ↵ scaffold_sequence_record_dict; stranded=stranded,
28          ↵ deterministic=deterministic)
29        push!(sample_df, [sample_scaffold, sample_Start, sample_End,
30          ↵ sample_sequence, strand]) #push the sample to the df
31        sample_length = sample_End - sample_Start + 1
32        sample_set_counter += sample_length #increase the counter by
33          ↵ the length of the sampled sequence
34        metacoordinate_bitarray[sample_MetaStart:sample_MetaEnd] =
35          ↵ [false for base in 1:sample_length] #mark these residues
36          ↵ as sampled
37        put!(progress_updates, (partitionid,
38          ↵ min(sample_set_counter, sample_set_length)))
39      end
40      put!(completed_sample_jobs, (partitionid, sample_df))
41    end
42  end
43
44  #get_sample_set() SUBFUNCTIONS

```

```

26      #function defining whether partitions respect stranding
27      #→ upon fetching sequence (ie is the sequence fetched in
28      #→ the feature strand orientation, or are we agnostic
29      #→ about the strand we sample?)
30
31      function get_strand_dict()
32          return
33          #→ Dict("exon"⇒true,"periexonic"⇒true,"intergenic"⇒false)
34
35      end
36
37      #function to obtain a dict of scaffold sequences from a
38      #→ FASTA reader
39
40      function build_scaffold_seq_dict(genome_fa, genome_index)
41          genome_reader = open(FASTA.Reader, genome_fa,
42          #→ index=genome_index)
43          seq_dict::Dict{String, LongSequence} =
44          #→ Dict{String,FASTA.Record}()
45          @inbounds for record in genome_reader
46              id = FASTA.identifier(record)
47
48              #→ seq_dict[rectify_identifier(id)]=FASTA.sequence(record)
49
50          end
51
52          close(genome_reader)
53          return seq_dict
54
55      end
56
57
58      # function to convert scaffold ID from that observed by
59      #→ the masked .fna to the more legible one observed by
60      #→ the GRCz11 GFF3
61
62      function rectify_identifier(scaffold_id::String)
63          if length(scaffold_id) ≥ 4 && scaffold_id[1:4] =
64          #→ "CM00" #marks chromosome scaffold
65              chr_code = scaffold_id[5:10]
66              chr_no = "$(Int((parse(Float64,chr_code)) -
67              #→ 2884.2))"
68              return chr_no
69
70          else
71              return scaffold_id
72          end
73
74      end
75
76
77      #function to produce a single sample from a
78      #→ metacoordinate set and the feature df

```

```

54     function get_sample(metacoordinate_bitarray::BitArray,
55         sample_window_min::Integer,
56         sample_window_max::Integer, partition_df::DataFrame,
57         scaffold_seq_dict::Dict{String,LongSequence};
58         stranded::Bool=false, deterministic::Bool=false)
59         proposal_acceptance = false
60         sample_metaStart = 0
61         sample_metaEnd = 0
62         sample_Start = 0
63         sample_End = 0
64         sample_sequence = LongSequence{DNAAlphabet{2}}("")"
65         sample_scaffold = ""
66         strand = nothing
67
68     while proposal_acceptance == false
69         available_indices =
70             findall(metacoordinate_bitarray) #find all
71             unsampled indices
72             window = "FAIL"
73             start_index,feature_metaStart,feature_metaEnd,
74                 feature_length = 0,0,0,0
75             strand = '0'
76             while window == "FAIL"
77                 start_index = rand(available_indices)
78                     #randomly choose from the unsampled
79                     indices
80                     feature_metaStart, feature_metaEnd, strand =
81                         get_feature_params_from_metacoord(start_index,
82                             partition_df, stranded)
83                     feature_length =
84                         length(feature_metaStart:feature_metaEnd)
85                         #find the metaboundaries of the feature
86                         the index occurs in
87                         if feature_length ≥ sample_window_min #don't
88                             bother finding windows on features
89                             smaller than min

```

```

74
    window =
    ↵  determine_sample_window(feature_metaStart,
    ↵  feature_metaEnd, start_index,
    ↵  metacoordinate_bitarray,
    ↵  sample_window_min, sample_window_max)
    ↵  #get an appropriate sampling window
    ↵  around the selected index, given the
    ↵  feature boundaries and params
75
    end
76
end
77
78
sample_scaffoldid, sample_scaffold_start,
    ↵  sample_scaffold_end =
    ↵  meta_to_feature_coord(window[1],window[2],partition_df)
79
80
proposal_sequence =
    ↵  fetch_sequence(sample_scaffoldid,
    ↵  scaffold_seq_dict, sample_scaffold_start,
    ↵  sample_scaffold_end, strand;
    ↵  deterministic=deterministic) #get the
    ↵  sequence associated with the sample window
81
82
if mask_check(proposal_sequence)
    proposal_acceptance = true #if the sequence
    ↵  passes the mask check, accept the
    ↵  proposed sample
83
    sample_scaffold = sample_scaffoldid
    sample_Start = sample_scaffold_start
    sample_End = sample_scaffold_end
    sample_metaStart = window[1]
    sample_metaEnd = window[2]
    sample_sequence=proposal_sequence
84
    end
85
end
86
87
return sample_scaffold, sample_Start, sample_End,
    ↵  sample_metaStart, sample_metaEnd,
    ↵  sample_sequence, strand
88
89
90
91
92
93
94
#function to find a valid sampling window
95

```

```

96      function
97        ← determine_sample_window(feature_metaStart::Integer,
98          ← feature_metaEnd::Integer, metacoord::Integer,
99            ← metacoord_bitarray::BitArray,
100           ← sample_window_min::Integer,
101             ← sample_window_max::Integer)
102   window_start = 0
103   window_end = 0
104   feature_size = feature_metaEnd - feature_metaStart +
105     ← 1
106   feature_sampled = any(!eval,
107     ← metacoord_bitarray[feature_metaStart:feature_metaEnd])
108   #attempt to construct the sample window by taking the
109     ← whole feature
110   if !feature_sampled && feature_size <
111     ← sample_window_max && feature_size >
112       ← sample_window_min
113     window_start = feature_metaStart
114     window_end = feature_metaEnd
115     return window_start, window_end
116   else #if this fails, build the biggest window we can
117     ← from the sampling point, within the feature
118   featurepos = metacoord - feature_metaStart + 1
119   next_sampled_index = findnext(!eval,
120     ← metacoord_bitarray[feature_metaStart:feature_metaEnd],
121     ← featurepos)
122   prev_sampled_index = findprev(!eval,
123     ← metacoord_bitarray[feature_metaStart:feature_metaEnd],
124     ← featurepos)
125   next_sampled_index ≡ nothing ? (window_end =
126     ← feature_metaEnd) : (window_end =
127     ← next_sampled_index + feature_metaStart - 1)
128   prev_sampled_index ≡ nothing ? (window_start =
129     ← feature_metaStart) : (window_start =
130     ← prev_sampled_index + feature_metaStart - 1)
131   windowsize = window_end - window_start + 1
132   #check to see if this window is bigger than the
133     ← min, if not, return a failure code
134   windowsize < sample_window_min && return "FAIL"
135   #check to see if this window is bigger than the
136     ← max, if so, trim it before returning,
137     ← removing as evenly as possible around the
138     ← metacoordinate

```

```

116         if windowsize > sample_window_max
117             bases_to_trim = windowsize -
118                 ↳ sample_window_max
119             clearance_5P = metacoord - window_start + 1
120             clearance_3P = window_end - metacoord + 1
121             trimmed=0
122             while trimmed < bases_to_trim
123                 if clearance_5P ≥ clearance_3P &&
124                     ↳ clearance_5P > 0
125                         clearance_5P -= 1
126                     elseif clearance_3P > clearance_5P &&
127                         ↳ clearance_3P > 0
128                             clearance_3P -= 1
129                         end
130                         trimmed +=1
131                     end
132                     return window_start, window_end
133                 end
134             end
135
136
137             # function to check for repetitive stretches or
138                 ↳ degenerate bases in proposal sequence
139             function mask_check(proposal_sequence::LongSequence)
140                 proposal_acceptance = true
141                 if hasambiguity(proposal_sequence) ||
142                     isrepetitive(proposal_sequence,
143                         (length(proposal_sequence) ÷ 10))
144                         proposal_acceptance = false
145                     end
146                     return proposal_acceptance
147             end
148
149 ####SHARED SEQUENCE FETCHER#####
150 # function to get proposal sequence from dict of scaffold sequences,
151     ↳ given coords and scaffold id
152 function fetch_sequence(scaffold_id::String,
153     ↳ scaffold_seq_dict::Dict{String, LongSequence},
154     ↳ proposal_start::Integer, proposal_end::Integer, strand::Char;
155     ↳ deterministic=false)

```

```

149     if strand == '0' #unstranded samples may be returned with no
150         ↪ preference in either orientation
151         deterministic ? strand = '+' : (rand(1)[1] ≤ .5 ? strand = '+' :
152             ↪ strand = '-')
153     end
154     if strand == '+'
155         proposal_sequence =
156             ↪ scaffold_seq_dict[scaffold_id][proposal_start:proposal_end]
157     elseif strand == '-'
158         proposal_sequence =
159             ↪ reverse_complement(scaffold_seq_dict[scaffold_id][proposal_start:proposal_end])
160     else
161         throw(ArgumentError("Invalid sample code! Must be '+', '-' , or
162             ↪ '0' (random strand)"))
163     end
164
165     return proposal_sequence
166 end
167
168 ####SHARED BASIC COORDINATE SUBFUNCTIONS#####
169 # function to push scaffold ID, start, and end points of given
170     ↪ featuretype to supplied dataframe
171 function build_feature_df!(GFF3_path::String, feature_type::String,
172     ↪ scaffold_exclusion::String, feature_df::DataFrame)
173     reader = open(GFF3.Reader, GFF3_path) # access the GFF3
174     @inbounds for record in reader # iterate over Gff3 records
175         if GFF3.isfeature(record) # if the record is a feature
176             if GFF3.featuretype(record) == feature_type #if the features
177                 ↪ is of the requested type, get the following info
178                 seqID = GFF3.seqid(record)
179                 if seqID ≠ scaffold_exclusion
180                     seq_start = GFF3.seqstart(record)
181                     seq_end = GFF3.seqend(record)
182                     if feature_type == "CDS" || feature_type == "gene"
183                         seq_strand = convert(Char, GFF3.strand(record))
184                         push!(feature_df, [seqID, seq_start, seq_end,
185                             ↪ seq_strand])
186                     else
187                         push!(feature_df, [seqID, seq_start, seq_end]) #
188                             ↪ push relevant info to the df
189                     end
190                 end
191             end
192         end
193     end

```

```

182     end
183   end
184   close(reader)
185 end
186
187 #function to assemble dataframe of scaffold coords + metacoords given
188 #→ gff3
189 function build_scaffold_df(gff3_path)
190   scaffold_df = DataFrame(SeqID = String[], Start = Integer[], End =
191   #→ Integer[])
192   build_feature_df!(gff3_path, "supercontig", "MT", scaffold_df)
193   build_feature_df!(gff3_path, "chromosome", "MT", scaffold_df)
194   add_metacoordinates!(scaffold_df)
195   return scaffold_df
196 end
197
198 #function to add pad to either side of some featureset
199 function add_pad_to_coordinates!(feature_df::DataFrame,
200   #→ scaffold_df::DataFrame, pad_size::Integer;
201   #→ col_symbols::Array{Symbol}=[:Start, :End])
202   pad_start_array = zeros(Integer,size(feature_df,1))
203   pad_end_array = zeros(Integer,size(feature_df,1))
204   feature_df[!, col_symbols[1]] = [max(feature_df.Start[i]-pad_size,1)
205   #→ for i in 1:size(feature_df,1)] #truncate pads at beginning and
206   #→ end of scaffolds
207   feature_df[!, col_symbols[2]] =
208   #→ [min(feature_df.End[i]+pad_size,scaffold_df.End[findfirst(isequal(feature_df.
209   #→ for i in 1:size(feature_df,1)])
210 end
211
212 #####SHARED METACOORDINATE FUNCTIONS#####
213 # function to add a metacoordinate column to a dataframe of scaffold
214 #→ positions, allowing sampling across all scaffolds
215 function add_metacoordinates!(feature_df::DataFrame)
216   meta_start_array = Integer[]
217   meta_end_array = Integer[]
218   metaposition = 1
219   @inbounds for feature in eachrow(feature_df)
220     push!(meta_start_array, (metaposition))
221     metaposition += feature.End - feature.Start
222     push!(meta_end_array, (metaposition))
223     metaposition += 1
224   end
225 end

```

```

216     feature_df.MetaStart = meta_start_array # metacoordinate contains
217         ↳ 'start' metaposition across all genomic material
218     feature_df.MetaEnd = meta_end_array # metacoordinate contains 'end'
219         ↳ metaposition across all genomic material
220 end
221
222 # function to convert metacoordinate set to scaffold-relative coordinates
223 function meta_to_feature_coord(meta_start::Integer, meta_end::Integer,
224     ↳ feature_df::DataFrame)
225     feature_row = get_feature_row_index(feature_df, meta_start)
226     seqid = feature_df.SeqID[feature_row]
227     scaffold_start = feature_df.Start[feature_row] + (meta_start -
228         ↳ feature_df.MetaStart[feature_row])
229     scaffold_end = feature_df.End[feature_row] -
230         ↳ (feature_df.MetaEnd[feature_row] - meta_end)
231     return seqid, scaffold_start, scaffold_end
232 end
233
234 #function to obtain the feature boundaries and strand of the feature that
235     ↳ a metacoordinate falls within
236 function get_feature_params_from_metacoord(metacoordinate::Integer,
237     ↳ feature_df::DataFrame, stranded::Bool)
238     feature_row = get_feature_row_index(feature_df, metacoordinate)
239     feature_metaStart = feature_df.MetaStart[feature_row]
240     feature_metaEnd = feature_df.MetaEnd[feature_row]
241     stranded ? feature_strand = feature_df.Strand[feature_row] :
242         ↳ feature_strand = '0'
243     return feature_metaStart, feature_metaEnd, feature_strand
244 end
245
246 #function obtain the index of a feature given its metacoordinate
247 function get_feature_row_index(feature_df::DataFrame,
248     ↳ metacoordinate::Integer)
249     total_feature_bases = feature_df.MetaEnd[end]
250     if metacoordinate == total_feature_bases #edge case of metacoord at
251         ↳ end of range
252         return size(feature_df,1) #last index in df
253     else
254         index =
255             ↳ findfirst(end_coord->end_coord>metacoordinate, feature_df.MetaEnd)
256             ↳ #find the index of the feature whose end metacoord is > the
257             ↳ query metacoord

```

```

245     if index > 1 && metacoordinate == feature_df.MetaEnd[index-1] #if
      ↵ the metacoordinate is the last base of the previous feature
246         if metacoordinate >= feature_df.MetaStart[index-1] &&
            ↵ metacoordinate <= feature_df.MetaEnd[index-1] #confirm
            ↵ that the metacoord is in the previous feature
            return index-1 #return the previous feature index
247     end
248
249     elseif metacoordinate >= feature_df.MetaStart[index] &&
            ↵ metacoordinate <= feature_df.MetaEnd[index] #else confirm
            ↵ that the metacoordinate is in the found feature
            return index #return the found feature index
250
251     else
252         throw(DomainError("Unexpected metacoordinate $metacoordinate
            ↵ in partition of $total_feature_bases bases, with feature
            ↵ start $(feature_df.MetaStart[index]), end
            ↵ $(feature_df.MetaEnd[index]))")
253     end
254 end
255 end

```

---

### 17.5.12 /src/likelihood\_funcs/bg\_lh\_matrix.jl

```

1 #function to obtain positional likelihoods for a sequence under a given
  ↵ BHMM.
2 function get_BGHMM_symbol_lh(seq::AbstractMatrix, hmm::BHMM)
3     @assert size(seq)[1] == 1
4     (seq=Array(transpose(seq))) # one sequence at a time only
5     symbol_lhs = zeros(length(seq))
6     length_mask = [length(seq)-1]
7
8     lls = bw_llhs(hmm, seq) #obtain log likelihoods for sequences and
  ↵ states
9     log_α = messages_forwards_log(hmm.a, hmm.A, lls, length_mask) #get
  ↵ forward messages
10    log_β = messages_backwards_log(hmm.A, lls, length_mask) # get
  ↵ backwards messages
11
12    #calculate observation probability and γ weights
13    K, Tmaxplus1, Strand = size(lls) #the last T value is the 0 end marker
  ↵ of the longest T
14
15    #transforms to cut down log_ξ, log_γ assignment times

```

```

16 lls = permutedims(lls, [2,1,3]) # from (K,T) to (T,K)
17 log_α = permutedims(log_α, [2,1,3])
18 log_β = permutedims(log_β, [2,1,3])
19
20 log_γ = fill(-Inf, Tmaxplus1,K)
21 log_pobs = logsumexp(lps.(log_α[1,:], log_β[1,:]))
22
23 @inbounds for i = 1:K, t = 1:length_mask[1]
24     log_γ[t,i] = lps(log_α[t,i],log_β[t,i],-log_pobs)
25 end
26
27 for t in 1:length_mask[1]
28     symbol_lh::AbstractFloat = -Inf #ie log(p=0)
29     for k = 1:K #iterate over states
30         state_symbol_lh::AbstractFloat = lps(log_γ[t,k],
31             ↳ log(hmm.B[k].p[seq[t]])) #state symbol likelihood is
32             ↳ the γ weight * the state symbol probability (log
33             ↳ implementation)
34         symbol_lh = logaddexp(symbol_lh, state_symbol_lh) #sum
35             ↳ the probabilities over states
36     end
37     symbol_lhs[t] = symbol_lh
38 end
39
40 return symbol_lhs[1:end-1] #remove trailing index position
41 end
42
43 #function to calculate BGHMM from an observation set and a dict of BGHMMs
44 function BGHMM_likelihood_calc(observations::DataFrame, BGHMM_dict::Dict,
45     ↳ code_partition_dict = get_partition_code_dict(false); symbol=:PadSeq)
46     lh_matrix_size = ((findmax(length.(collect(values(observations[!, symbol]))))[1]), length(observations[!, symbol]))
47     BGHMM_lh_matrix = zeros(lh_matrix_size) #T, Strand, 0
48
49     BGHMM_fragments = fragment_observations_by_BGHMM(observations[!, symbol], observations.MaskMatrix)
50
51
52 #@showprogress 1 "Writing frags to matrix.."
53 Threads.@threads for (jobid, frag) in BGHMM_fragments
54     (frag_start, o, partition, strand) = jobid
55 end

```

```

51     partition_BGHMM::BHMM =
52         ↵ BGHMM_dict[code_partition_dict[partition]]
53     no_symbols = length(partition_BGHMM.B[1].p)
54     order = Int(log(4,no_symbols) - 1)
55
56     order_seq = get_order_n_seqs([frag], order)
57     coded_seq = code_seqs(order_seq)
58
59     subseq_symbol_lh = get_BGHMM_symbol_lh(coded_seq,
60         ↵ partition_BGHMM)
61
62     if strand == -1
63         subseq_symbol_lh = reverse(subseq_symbol_lh)
64     end #positive, unstranded frags are inserted as-is
65     BGHMM_lh_matrix[frag_start:frag_start+length(frag)-1,o] =
66         ↵ subseq_symbol_lh
67 end
68
69 function fragment_observations_by_BGHMM(seqs :: AbstractVector,
70     ↵ masks :: AbstractVector)
71     likelihood_jobs = Vector{Tuple{Tuple,LongSequence{DNAAlphabet{2}}}}()
72     @showprogress 1 "Fragmenting observations by partition ..." for (o,
73         ↵ obs_seq) in enumerate(seqs)
74     mask = masks[o]
75     frags = Vector{LongSequence{DNAAlphabet{2}}}() #container for all
76         ↵ subsequences in observation
77     frag = LongSequence{DNAAlphabet{2}}()
78
79     frag_end=0
80     frag_start = 1
81
82     while frag_start < length(obs_seq) # while we're not at the
83         ↵ sequence end
84         curr_partition = mask[frag_start,1] #get the partition code
85             ↵ of the frag start
86         curr_strand = mask[frag_start,2] #get the strand of the frag
87             ↵ start
88
89         #JOBID COMPOSED HERE

```

```

84         jobid = (frag_start, o, curr_partition, curr_strand) #compose
85             ↳ an identifying index for this frag
86
86     findnext(!isequal(curr_partition),mask[:,1],frag_start) ≠
87         ↳ nothing ? frag_end =
88             findnext(!isequal(curr_partition),mask[:,1],frag_start)
89             ↳ -1 : frag_end = length(obs_seq) #find the next position
90                 ↳ in the frag that has a different partition mask value
91                 ↳ from hte current one and set that position-1 to frag end,
92                 ↳ alternately frag end is end of the overall sequence
93     frag = obs_seq[frag_start:frag_end] #get the frag bases
94     if curr_strand == -1 #if the fragment is reverse stranded
95         reverse_complement!(frag) #use the reverse complement
96             ↳ sequence
97     end
98
99     push!(likelihood_jobs,(jobid, frag)) #put the frag in the
100        ↳ jobs vec
101     frag_start = frag_end + 1 #move on
102
103 end
104
105 return likelihood_jobs
106
107 end

```

---

### 17.5.13 /src/likelihood\_funcs/hmm.jl

```

1 function obs_lh_given_hmm(observations, hmm; linear=true)
2     linear ? (return linear_likelihood(observations, hmm)) : (return
3             ↳ bw_likelihood(Matrix(transpose(observations)), hmm))
4 end
5
5 function bw_likelihood(observations, hmm)
6     obs_lengths = [findfirst(iszero, observations[:,o])-1 for o in
7             ↳ 1:size(observations,2)]
8
8     lls = bw_llhs(hmm, observations)
9     log_α = messages_forwards_log(hmm.a, hmm.A, lls, obs_lengths)
10    log_β = messages_backwards_log(hmm.A, lls, obs_lengths)
11
11    0 = size(lls)[3] #the last T value is the 0 end marker of the longest
12        ↳ T
13    log_pobs = zeros(0)

```

```

14     Threads.@threads for o in 1:O
15         log_pobs[o] = logsumexp(lps.(log_α[:,1,o], log_β[:,1,o]))
16     end
17
18     return sum(log_pobs)
19 end
20
21 function linear_likelihood(observations,hmm)
22     O= size(observations,1);
23     obs_lengths = [findfirst(iszero,observations[o,:])-1 for o in
24     ↳ 1:size(observations,1)] #mask calculations here rather than
25     ↳ mle_step to prevent recalculation every iterate
26
27     a = transpose(log.(hmm.a)); A = log.(hmm.A)
28     N = length(hmm.B)
29     mask=observations.≠0
30     #INITIALIZATION
31     βoi_T = zeros(O,N); βoi_t = zeros(O,N) #log betas at T initialised as
32     ↳ zeros
33
34     #RECURRENCE
35     βoi_T = backwards_lh_sweep!(hmm, A, N, βoi_T, βoi_t, observations,
36     ↳ mask, obs_lengths)
37
38     #TERMINATION
39     lls = c_llhs(hmm,observations[:,1])
40     α1om = lls .+ a #first position forward msgs
41
42     return lps([logsumexp(lps.(α1om[o,:], βoi_T[o,:])) for o in 1:O])
43 end
44     #LINEAR_STEP SUBFUNCS
45     function backwards_lh_sweep!(hmm, A, N, βoi_T, βoi_t,
46     ↳ observations, mask, obs_lengths)
47         for t in maximum(obs_lengths)-1:-1:1
48             last_β=copy(βoi_T)
49             lls = c_llhs(hmm,observations[:,t+1])
50             omask = findall(mask[:,t+1])
51             βoi_T[omask,:] .+= view(lls,omask,:)
52             Threads.@threads for m in 1:N
53                 βoi_t[omask,m] =
54                 ↳ logsumexp.(eachrow(view(βoi_T,omask,:).+transpose(view
55                 ↳ omask,:)))
56             end
57             βoi_T=copy(βoi_t)
58         end
59     end
60 end

```

```
51                     end  
52             return boi_T  
53         end
```

### 17.5.14 /src/reports/chain\_report.jl

```

1 struct Chain_Report
2     id::Chain_ID
3     final_hmm::BHMM
4     test_lh::AbstractFloat
5     naive_lh::AbstractFloat
6     final_delta::AbstractFloat
7     state_run_lengths::AbstractVector{AbstractFloat}
8     convergence_values::Chains
9     convergence_diagnostic::ChainDataFrame
10    converged::Bool
11 end
12
13 function Base.show(io::IO, report::Chain_Report)
14     nominal_dict=Dict(0=>"th",1=>"st",2=>"nd",3=>"rd",4=>"th",5=>"th")
15     haskey(nominal_dict,report.id.order) ?
16         (nom_str=nominal_dict[report.id.order]) : (nom_str="th")
17     printstyled(io, stdout, "BHMM EM Chain Results\n"; bold=true)
18     println(io, "$(report.id.K)-state, $(report.id.order)$nom_str order")
19     ↵   BHMM")
20     println(io, "Trained on observation set \\"$(report.id.obs_id)\\"")
21     report.test_lh > report.naive_lh ? (lh_str="greater";
22         ↵   lh_color=:green) : (lh_str="less"; lh_color=:red)
23     printstyled(io, "Test logP(O|θ): $(report.test_lh), $lh_str than the
24         ↵   naive model's $(report.naive_lh)\n"; color=lh_color)
25     println(io, "Replicate $(report.id.replicate)")
26     report.converged ? println(io, "Converged with final step δ
27         ↵   $(report.final_delta)") : println(io, "Failed to converge!")
28     println(io, " -----")
29     ↵   -----
30     ↵   -----")
31     println(io, "Last θ:")
32     show(report.final_hmm)
33     println(io, "State Mean Feature Length (bp)")
34     for i in 1:length(report.state_run_lengths)
35         println(io, "K$i:
36             ↵   $(report.state_run_lengths[i].*float(report.id.order+1))")

```

```

29     end
30     println(io, " -----")
31     ↪   -----
32     ↪   -----")
33
34     ↪   zidx=findfirst(!iszero,report.convergence_values["logP(0|θ)"].data)[1]
35     ↪   lh_vec=Vector([report.convergence_values["logP(0|θ)"].data ... ][zidx:end])
36
37     lh_plot=lineplot([zidx:length(lh_vec)+zidx-1 ... ],lh_vec;title="Chain
38     ↪   likelihood evolution", xlabel="Training iterate",
39     ↪   xlim=(0,length(lh_vec)+zidx+1), ylim=
40     ↪   (floor(minimum(lh_vec),sigdigits=2),ceil(maximum(lh_vec),sigdigits=2)),
41     ↪   name="logP(0|θ)")
42     lineplot!(lh_plot,[report.naive_lh for i in 1:length(lh_vec)],
43     ↪   color=:magenta,name="naive")
44     show(io, lh_plot)
45     println(io, "\n")
46
47
48     k1vec=Vector([report.convergence_values["K1"].data ... ])
49     k_plot=lineplot(k1vec, title="State p(Auto) evolution",
50     ↪   xlabel="Training iterate", ylabel="prob",
51     ↪   name="K1",ylim=(0,1),xlim=(0,length(k1vec)))
52     for k in 2:report.id.K
53         kvec=Vector([report.convergence_values["K$k"].data ... ])
54         lineplot!(k_plot,kvec,name="K$k")
55     end
56     show(io,k_plot)
57     println(io)
58
59
60     printstyled(io,"Convergence Diagnostics\n",bold=true)
61     if report.convergence_diagnostic.name == "short"
62         printstyled(io,"Convergence diagnostics unavailable for chains
63         ↪   <10 steps!\n", color=:yellow)
64     elseif report.convergence_diagnostic.name == "error"
65         printstyled(io,"Convergence diagnostics errored! Zeros in
66         ↪   autotransition matrix?\n", color=:red)
67     else

```

```

54     all(Bool.(report.convergence_diagnostic.nt.stationarity)) &&
55         all(Bool.(report.convergence_diagnostic.nt.test)) ?
56             printstyled(io, "Likelihood and autotransition probabilites
57             converged and passing tests.\n", color=:green) :
58             printstyled(io, "Not all parameters converged or passing
59             tests!\n",color=:red)
60         display(report.convergence_diagnostic)
61     end
62 end
63
64 function latex_report(report::Chain_Report)
65
66 end
67
68 function report_chains(chains::Dict{Chain_ID,Vector{EM_step}},
69     test_sets::Dict{String,Vector{LongSequence{DNAAlphabet{2}}}});
70     naive_hmm=BHMM(ones(1,1),[Categorical(4)])
71     job_ids=[id for (id,chain) in chains] # make list of chain_ids
72     partitions=unique([id.obs_id for id in job_ids]) #get list of unique
73         partitions represented among chain_ids
74     naive_order = Integer(log(4,length(naive_hmm.B[1].support))-1)
75     for partition in partitions #check that all partitions are available
76         for testing and make sure codes for naive hmm are generated by
77         adding job_ids
78             !(partition in keys(test_sets)) &&
79                 throw(ArgumentError("Observation set $partition required by
80                 chains for testing not present in test sets!"))
81             push!(job_ids,
82                 Chain_ID(partition,length(naive_hmm.a),naive_order, 1))
83         end
84
85         code_dict=code_job_obs(job_ids, test_sets) #code all the obs sets
86             that are necessary ot perform our tests
87         naive_lhs=Dict{String,Float64}() #construct dict of naive likelihoods
88             for all partitions to be tested prior to individual tests
89             @showprogress 1 "Testing naive hmm..." for
90                 ((partition,order),obs_set) in code_dict
91                 order=naive_order &&
92                     (naive_lhs[partition]=obs_lh_given_hmm(code_dict[(partition,naive_order)])
93             end
94
95         reports=Dict{Chain_ID,Chain_Report}()
96         @showprogress 1 "Testing chains ... " for (id, chain) in chains

```

```

80     chains_array=zeros(length(chain),1+id.K)
81     for (n,step) in enumerate(chain)
82         chains_array[n,1]=step.log_p
83         chains_array[n,2:end]=get_diagonal_array(step.hmm)
84     end
85     convergence_values=Chains(chains_array,[ "logP(0|θ)",
86         → ["K" * string(i) for i in 1:id.K] ... ])
86     if length(chain) ≥ 10
87         try
88             convergence_diagnostic=heideldiag(convergence_values)[1]
89         catch
90             → convergence_diagnostic=ChainDataFrame("error",(:;a⇒[1.]))
91         end
92     else
93         convergence_diagnostic=ChainDataFrame("short",(:;a⇒[1.]))
94     end
95
96     id.K > 1 ?
97         → (mean_run_lengths=sim_run_lengths(get_diagonal_array(chain[end].hmm),1000
98         → : (mean_run_lengths=[Inf])
97         test_lh=obs_lh_given_hmm(code_dict[(id.obs_id,id.order)],
98         → chain[end].hmm)
98         reports[id]=Chain_Report(id, chain[end].hmm, test_lh,
98         → naive_lhs[id.obs_id], chain[end].delta, mean_run_lengths,
98         → convergence_values, convergence_diagnostic,
98         → chain[end].converged)
99     end
100
101     return reports
102 end
103
104 function get_diagonal_array(hmm::BHMM)
105     k = length(hmm.a)
106     diagonal = zeros(k)
107     @inbounds for i in 1:k
108         diagonal[i] = hmm.A[i,i]
109     end
110     return diagonal
111 end
112
113 #function to simulate run lengths for vector of diagonal values
114 function sim_run_lengths(diagonal_value::AbstractArray, samples::Integer)

```

```

115     mean_run_lengths = zeros(length(diagonal_value))
116     for (i, value) in enumerate(diagonal_value)
117         if value == 0.
118             mean_run_lengths[i]=0.
119         elseif value ==1.
120             mean_run_lengths[i]=Inf
121         else
122             runlengths = zeros(Integer, samples)
123             for s in 1:samples
124                 run = true
125                 runlength = 0
126                 while run
127                     runlength += 1
128                     if rand(1)[1] > value
129                         run = false
130                     end
131                 end
132                 runlengths[s] = runlength
133             end
134             mean_run_lengths[i] = mean(runlengths)
135         end
136     end
137     return mean_run_lengths
138 end

```

---

### 17.5.15 /src/reports/partition\_report.jl

```

1 struct Order_Report
2     converged_K::Vector{Float64}
3     converged_lh::Vector{Float64}
4     failed_K::Vector{Float64}
5     failed_lh::Vector{Float64}
6 end
7
8 struct Partition_Report
9     partition_id::String
10    naive_lh::Float64
11    orddict::Dict{Integer,Order_Report}
12    best_model::Tuple{Chain_ID,BHMM}
13    best_repset::Vector{Chain_ID}
14 end
15

```

```

16 function Base.show(io::IO, report::Partition_Report)
17     printstyled(io, "BACKGROUND HMM TRAINING REPORT\n", bold=true)
18     printstyled(io, "Genome partition id: $(report.partition_id)\n",
19                 ↳ bold=true, color=:magenta)
20
21     for (order, ordreport) in report.orddict
22         if length(ordreport.failed_lh)>0
23             ordplot=scatterplot(ordreport.converged_K,
24                                 ↳ ordreport.converged_lh; name="Converged", title="Order
25                                 $order HMMs", xlabel="K# States", ylabel="P(O|θ)",
26                                 ↳ color=:green, xlim=(1,maximum(ordreport.converged_K)),
27                                 ↳ ylim=(floor(min(minimum(ordreport.converged_lh),minimum(ordreport.fai
28             length(ordreport.failed_K) ≥ 1 && scatterplot!(ordplot,
29                     ↳ ordreport.failed_K, ordreport.failed_lh;
30                     ↳ name="Unconverged",color=:red)
31             lineplot!(ordplot, [report.naive_lh for i in
32                     ↳ 1:maximum(ordreport.converged_K)],
33                     ↳ name="Naive",color=:magenta)
34             show(ordplot)
35             println()
36         else
37             ordplot=scatterplot(ordreport.converged_K,
38                                 ↳ ordreport.converged_lh; name="Converged", title="Order
39                                 $order HMMs", xlabel="K# States", ylabel="P(O|θ)",
40                                 ↳ color=:green, xlim=(1,maximum(ordreport.converged_K)),
41                                 ↳ ylim=(floor(min(minimum(ordreport.converged_lh),report.naive_lh),sigd
42             lineplot!(ordplot, [report.naive_lh for i in
43                     ↳ 1:maximum(ordreport.converged_K)],
44                     ↳ name="Naive",color=:magenta)
45             show(ordplot)
46             println()
47         end
48     end
49 end
50
51
52 function report_partitions(chain_reports::Dict{Chain_ID,Chain_Report})
53     partitions=Vector{String}()
54     orders=Vector{Integer}()
55     Ks=Vector{Integer}()
56     reps=Vector{Integer}()
57     for id in keys(chain_reports)
58         !in(id.obs_id, partitions) && push!(partitions, id.obs_id)
59         !in(id.order, orders) && push!(orders, id.order)
60     end
61 end

```

```

44         !in(id.K, Ks) && push!(Ks, id.K)
45         !in(id.replicate, reps) && push!(reps, id.replicate)
46     end
47
48     reports=Vector{Partition_Report}()
49     for partition in partitions
50         best_lh,best_rep=-Inf,Chain_ID("empty",1,0,1)
51         naive_lh=1.
52         orddict=Dict{Integer,Order_Report}()
53         for order in orders
54             conv_statevec=Vector{Float64}()
55             fail_statevec=Vector{Float64}()
56             conv_lh_vec=Vector{Float64}()
57             fail_lh_vec=Vector{Float64}()
58             for (id,report) in chain_reports
59                 if id.obs_id==partition && id.order==order
60                     report.test_lh > best_lh &&
61                         ↵ (best_lh=report.test_lh;best_rep=id)
62                     naive_lh=1. && (naive_lh=chain_reports[id].naive_lh)
63                     chain_reports[id].converged ?
64                         ↵ (push!(conv_statevec,id.K);
65                         push!(conv_lh_vec,chain_reports[id].test_lh)) :
66                         ↵ (push!(fail_statevec,id.K);
67                         push!(fail_lh_vec,chain_reports[id].test_lh))
68                 end
69             end
70             ↵ orddict[order]=Order_Report(conv_statevec,conv_lh_vec,fail_statevec,f
71         end
72         best_model=(best_rep,chain_reports[best_rep].final_hmm)
73         best_repset=[Chain_ID(partition, best_rep.K, best_rep.order, rep)
74             ↵ for rep in reps]
75
76             ↵ push!(reports,Partition_Report(partition,naive_lh,orddict,best_model,best_
77         end
78         return reports
79     end
80 
```

---

### 17.5.16 /src/reports/replicate\_convergence.jl

---

```

1 struct Replicate_Report
2     ids::Vector{Chain_ID} #replicates

```

```

3 state_vecs::Dict{Chain_ID,Matrix{Float64}} #vectors of autotransition
4   ↵ probabilities evolving by iterate, concatenated to matrices,
5   ↵ indexed by Chain_ID
6 emission_arrays::Dict{Chain_ID,Array{Float64}}
7   ↵ #iterate*symbol*state_vecs
8 sorted_id1_states::Vector{Integer}
9 sort_dicts::Dict{Chain_ID,Dict{Integer, Integer}}#emission channels
10  ↵ sorted on the basis of euclidean closeness to ids[1]
11 sorted_symbols::Dict{Integer,Vector{Integer}}
12
13 Replicate_Report(ids,state_vecs,emission_arrays,
14   ↵ sorted_id1_states,sort_dicts,sorted_symbols) =
15   ↵ assert_repreport(ids) &&
16   ↵ new(ids,state_vecs,emission_arrays,sorted_id1_states,sort_dicts,sorted_symbols)
17
18 end
19
20 function assert_repreport(ids::Vector{Chain_ID})
21   Ks=Vector{Int64}()
22   orders=Vector{Int64}()
23   replicates=Vector{Int64}()
24   obsids=Vector{String}()
25
26   for id in ids
27     push!(Ks,id.K)
28     push!(orders,id.order)
29     push!(obsids,id.obs_id)
30     push!(replicates, id.replicate)
31   end
32
33   length(unique(Ks)) > 1 && throw(ArgumentError("Replicate set includes
34   ↵ chains with different K state numbers! All chains must have HMMs
35   ↵ with the same number of states."))
36   length(unique(orders)) > 1 && throw(ArgumentError("Replicate set
37   ↵ includes chains with different order coding numbers! All chains
38   ↵ must have HMMs with the same order coding."))
39   !allunique(replicates) && throw(ArgumentError("Replicate set includes
40   ↵ chains with the same replicate #. Replicates should be unique!"))
41
42   return true
43 end

```

```

33 function
34   → report_replicates(repset::Vector{Chain_ID},chains::Dict{Chain_ID,Vector{EM_step}})
35   assert_repreport(repset) #check that the repset will pass its
   → constructor before doing the work
36   emission_arrays=Dict{Chain_ID,AbstractArray{AbstractFloat}} }()
37   state_arrays=Dict{Chain_ID,AbstractArray{AbstractFloat}} }()
38
39   for id in repset
40     emission_array=zeros(length(chains[id]),4^(id.order+1),id.K)
41     state_array=zeros(length(chains[id]),id.K)
42     for (it, step) in enumerate(chains[id])
43       for k in 1:id.K
44         emission_array[it,:,k]=step.hmm.B[k].p
45         state_array[it,k]=step.hmm.A[k,k]
46       end
47     end
48     emission_arrays[id]=emission_array
49     state_arrays[id]=state_array
50   end
51
52   println(repset)
53   println(emission_arrays)
54   println(state_arrays)
55
56   → sort_dicts,sorted_id1_states,sorted_symbols=sort_emitters_by_distance!(repset)
57
58   return
59   → Replicate_Report(repset,state_arrays,emission_arrays,sorted_id1_states,
60   → sort_dicts,sorted_symbols)
61 end
62
63 function sort_emitters_by_distance!(repset,emission_arrays)
64   id1=repset[1]
65   length(repset)==1 && (return
66   → Dict{Chain_ID,Dict{Integer,Integer}}() ,[1:id1.K ... ],[1:4^id1.order+1])
67
68   final_emissions =
69   → zeros(length(repset),size(emission_arrays[id1])[2:3] ... )
70   for (n,id) in enumerate(repset)
71     final_emissions[n,:,:] = emission_arrays[id][end,:,:]
72   end
73
74   distances=zeros(length(repset)-1,id1.K,id1.K)

```

```

69   for rep in 1:length(repset)-1
70     for k in 1:id1.K, j in 1:id1.K
71
72       → distances[rep,k,j]=euclidean(final_emissions[1,:,:k],final_emissions[1,:,:j])
73     end
74   end
75
76   sortdicts=Dict{Chain_ID,Dict{Integer,Integer}}(){}
77   sorted_id1_states=Vector{Integer}(){}
78   sorted_symbols=Dict{Integer,Vector{Integer}}(){}
79   for rep in 1:length(repset)-1
80     sortdicts[repset[rep+1]]=Dict{Integer,Integer}(){}
81   end
82
83   for i in 1:id1.K
84     id1_mindist_K=findmin([sum([minimum(distances[rep,k,:]) for rep
85       in 1:length(repset)-1] for k in 1:id1.K])[2]#find the state
86       → from the first replicate that has minimum cumulative distance
87       → to the closest state in the other replicates, excluding
88       → already-chosen states by spiking their values
89     push!(sorted_id1_states,id1_mindist_K)
90   end
91
92
93   → symbol_distances=sum(hcat([euclidean.(final_emissions[1,:,:id1_mindist_K],
94     → for (n,id) in enumerate(repset[2:end])]...),dims=2)
95   sorted_symbols[id1_mindist_K]=sortperm(symbol_distances[:,1])
96
97   return sortdicts,sorted_id1_states,sorted_symbols
98
99
100 function Base.show(io::IO, report::Replicate_Report;
101   → top_states::Integer=report.ids[1].K, top_symbols::Integer=2)
102   printstyled(io, "REPLICATE CONVERGENCE REPORT\n", color=:green)

```

```

102 println("-----")
103   ↵ -----)
104 for i in 1:top_states
105   id=report.ids[1]
106   id1_state=report.sorted_id1_states[i]
107   itmax=maximum([size(report.state_vecs[id],1) for id in
108     ↵ report.ids])
109
110   autoplt=lineplot(report.state_vecs[id][:,id1_state],
111     title="Autotransition prob. convergence",
112     name="$(report.ids[1]) K$id1_state",
113     xlim=(0,ceil(itmax, sigdigits=2)),
114     ylim=(0,1),
115     xlabel="iterate",
116     ylabel="p")
117
118   ↵ symplot=lineplot(report.emission_arrays[id][:,report.sorted_symbols[id1_s
119
120     ↵ report.emission_arrays[id][:,report.sorted_symbols[id1_st
121     title="Symbol prob. convergence",
122     name="$(report.ids[1]) K$id1_state",
123     xlim=(0,1),
124     ylim=(0,1),
125     xlabel="Symbol 1 p",
126     ylabel="S2 p")
127
128 for (n,id) in enumerate(report.ids)
129   if n > 1
130     lineplot!(autoplt,
131       ↵ report.state_vecs[id][:,report.sort_dicts[id][id1_state]],
132       name="$(report.ids[n])"
133       ↵ K$(report.sort_dicts[id][id1_state])))
134     lineplot!(symplot,
135       ↵ report.emission_arrays[id][:,report.sorted_symbols[id
136
137       ↵ report.emission_arrays[id][:,report.sorted_symbols[id
138       name="$(report.ids[n])"
139       ↵ K$(report.sort_dicts[id][id1_state])))
140
141 end
142
143 end

```

```

135     show(autopl)
136     println("\n")
137     show(symplot)
138     println("\n")
139
140   end
141 end

```

---

### 17.5.17 /src/utilities/BBG\_analysis.jl

```

1 #function to produce matrix of hmm chain parameter coordinate evolution-
2   → selects 3 matched emission parameters from 3+ state HMMs, one per
3   → state for 3 states. Matching is performed by minimizing euclidean
4   → distance between the state emission probability vectors B for the
5   → states to be matched (as EM-optimised replicates will have similar
6   → states organized in different orders)
7
8 function chain_3devo_coords(chains :: Vector{Vector{Any}})
9   length(chains) ≤ 0 && throw(ArgumentError, "Argument must be vector of
10   → more than one hmm chain")
11   coords=[Vector{Tuple{AbstractFloat,AbstractFloat,AbstractFloat}}]()
12   → for i in 1:length(chains)]
13
14   for step in chains[1]
15     hmm=step[2]
16     length(hmm.B)<3 && throw(ArgumentError, "3- or greater state hmms
17     → required")
18     push!(coords[1], (hmm.B[1].p[1],hmm.B[2].p[1],hmm.B[3].p[1]))
19   end
20
21   for (c, chain) in enumerate(chains[2:end])
22     ref_idxs=Vector{Integer}()
23
24     → ref_vecs=[chains[1][end][2].B[1].p,chains[1][end][2].B[2].p,chains[1][end]
25     end_hmm=chain[end][2]
26     length(end_hmm.B)<3 && throw(ArgumentError, "3- or greater state
27     → hmms required")
28
29     for vec in ref_vecs
30       euclideans=Vector{AbstractFloat}()
31       for d in 1:length(end_hmm.B)
32         push!(euclideans, euclidean(vec, end_hmm.B[d].p))
33     end
34   end
35 end

```

```

23         end
24         min_euc_idx = findmin(euclidean)[2]
25         while min_euc_idx in ref_idxs
26             euclidean[min_euc_idx]=1.
27             min_euc_idx = findmin(euclidean)[2]
28         end
29         push!(ref_idxs,findmin(euclidean)[2])
30     end
31
32     for step in chain
33         hmm=step[2]
34         push!(coords[c+1],
35             → (hmm.B[ref_idxs[1]].p[1],hmm.B[ref_idxs[2]].p[1],hmm.B[ref_idxs[3]].p
36         end
37
38     return coords
39 end

```

---

### 17.5.18 /src/utilities/BBG\_progressmeter.jl

```

1 #UTILITY PROGRESSMETER, REPORTS WORKER NUMBER AND CURRENT ITERATE
2 mutable struct ProgressHMM{T<:Real} <: ProgressMeter.AbstractProgress
3     thresh::T
4     dt::AbstractFloat
5     val::T
6     counter::Integer
7     triggered::Bool
8     tfirst::AbstractFloat
9     tlast::AbstractFloat
10    printed::Bool           # true if we have issued at least one status
11    → update
12    desc::AbstractString # prefix to the percentage, e.g. "Computing ... "
13    color::Symbol          # default to green
14    output::IO              # output stream into which the progress is
15    → written
16    numprintedvalues::Integer # num values printed below progress in
17    → last iteration
18    offset::Integer          # position offset of progress bar
19    → (default is 0)
20    steptime::AbstractFloat
21    function ProgressHMM{T}(thresh;

```

```

18             dt::Real=0.1,
19             desc::AbstractString="Progress: ",
20             color::Symbol=:green,
21             output::IO=stderr,
22             offset::Integer=0,
23             start_it::Integer=1) where T
24
25         tfirst = tlast = time()
26         printed = false
27         new{T}(thresh, dt, typemax(T), start_it, false, tfirst, tlast,
28             ↳ printed, desc, color, output, 0, offset, 0.0)
29     end
30 end
31
32 ProgressHMM(thresh::Real, dt ::Real=0.1, desc ::AbstractString="Progress:
33     " ,
34             ↳
35             ↳ color::Symbol=:green, output::IO=stderr;
36             ↳ offset::Integer=0, start_it::Integer=1) =
37             ↳ ProgressHMM{typeof(thresh)}(thresh, dt=dt, desc=desc,
38             ↳ color=color, output=output, offset=offset,
39             ↳ start_it=start_it)
40
41 ProgressHMM(thresh::Real, desc ::AbstractString, offset::Integer=0,
42             ↳ start_it::Integer=1) = ProgressHMM{typeof(thresh)}(thresh, desc=desc,
43             ↳ offset=offset, start_it=start_it)
44
45 function update!(p::ProgressHMM, val, steptime; options ... )
46     p.val = val
47     p.counter += 1
48     p.steptime = steptime
49     updateProgress!(p; options ... )
50 end
51
52 function updateProgress!(p::ProgressHMM; showvalues = Any[], valuecolor =
53     ↳ :blue, offset::Integer = p.offset, keep = (offset == 0))
54     p.offset = offset
55     t = time()
56     if p.val ≤ p.thresh && !p.triggered
57         p.triggered = true
58         if p.printed
59             p.triggered = true
60             dur = ProgressMeter.durationstring(t-p.tfirst)
61             msg = @sprintf "%s Time: %s (%d iterations)" p.desc dur
62             ↳ p.counter

```

```

52     print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
53     ProgressMeter.move_cursor_up_while_clearing_lines(p.output,
54         ↪ p.numprintedvalues)
55     ProgressMeter.printover(p.output, msg, p.color)
56     ProgressMeter.printvalues!(p, showvalues; color = valuecolor)
57     if keep
58         println(p.output)
59     else
60         print(p.output, "\r\u1b[A" ^ (p.offset +
61             ↪ p.numprintedvalues))
62     end
63   end
64
65   if t > p.tlast+p.dt && !p.triggered
66     elapsed_time = t - p.tfirst
67     msg = @sprintf "%s (convergence: %g => %g, iterate: %g, step
68       ↪ time: %s)" p.desc p.val p.thresh p.counter hmss(p steptime)
69     print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
70     ProgressMeter.move_cursor_up_while_clearing_lines(p.output,
71         ↪ p.numprintedvalues)
72     ProgressMeter.printover(p.output, msg, p.color)
73     ProgressMeter.printvalues!(p, showvalues; color = valuecolor)
74     print(p.output, "\r\u1b[A" ^ (p.offset + p.numprintedvalues))
75     # Compensate for any overhead of printing. This can be
76     # especially important if you're running over a slow network
77     # connection.
78     p.tlast = t + 2*(time()-t)
79     p.printed = true
80   end
81
82   function hmss(dt)
83     isnan(dt) && return "NaN"
84     (h,r) = divrem(dt,60*60)
85     (m,r) = divrem(r, 60)
86     (isnan(h)||isnan(m)||isnan(r)) && return "NaN"
87     string(Int(h),":",Int(m),":",Int(ceil(r)))
88   end

```

---

### 17.5.19 /src/utilities/HMM\_init.jl

---

```

1 function autotransition_init(K::Integer, order::Integer,
2   ↵ partition::String="")
3   a = rand(Dirichlet(ones(K)/K)) #uninformative prior on initial
4   ↵ state probabilities
5   A = strong_autotrans_matrix(K)
6   no_emission_symbols = Int(4^(order+1)) #alphabet size for the
7   ↵ order
8   emission_dists = [generate_emission_dist(no_emission_symbols) for
9     ↵ i in 1:K]
10  #generate the BHMM with the appropriate transition matrix and
11  ↵ emissions distributions
12  hmm = BHMM(a, A, emission_dists, partition)
13 end
14
15 #function to construct BHMM transition matrix with strong
16  ↵ priors on auto-transition
17
18 function strong_autotrans_matrix(states::Integer,
19   ↵ prior_dope::AbstractFloat=(states*250.0),
20   ↵ prior_background::AbstractFloat=.1)
21   transition_matrix=zeros(states,states)
22   for k in 1:states
23     dirichlet_params = fill(prior_background, states)
24     dirichlet_params[k] = prior_dope
25     transition_matrix[k,:] =
26       ↵ rand(Dirichlet(dirichlet_params))
27   end
28   return transition_matrix
29 end
30
31 #function to construct BHMM state emission distribution from
32  ↵ uninformative dirichlet over the alphabet size
33
34 function generate_emission_dist(no_emission_symbols,
35   ↵ prior=Dirichlet(ones(no_emission_symbols)/no_emission_symbols))
36   return Categorical(rand(prior))
37 end

```

---

### 17.5.20 /src/utilities/load\_balancer.jl

---

```

1 struct LoadConfig
2   k_range ::UnitRange
3   o_range ::UnitRange

```

---

```

4   blacklist::Vector{Chain_ID}
5   whitelist::Vector{Chain_ID}
6   function LoadConfig(k_range, o_range; blacklist=Vector{Chain_ID}(),
7     ↪ whitelist=Vector{Chain_ID}())
8     assert_loadconfig(k_range, o_range) && new(k_range, o_range,
9       ↪ blacklist, whitelist)
10    end
11  end
12
13
14
15
16
17 #subfunc to handle balancing memory load on dissimilar machines in
18   ↪ cluster
19
20
21
22
23
24
25
26
27

```

---

### 17.5.21 /src/utilities/log\_prob\_sum.jl

---

```

1      #subfuncs to handle sums of log probabilities that may
2      ↳ include -Inf (ie p=0), returning -Inf in this case
3      ↳ rather than NaNs
4      function lps(adjuvants)
5          prob = sum(adjuvants) ; isnan(prob) ? - Inf : prob
6      end
7
8      function lps(base, adjuvants ... )
9          prob = base+sum(adjuvants) ; isnan(prob) ? -Inf :
10         ↳ prob
11     end

```

---

### 17.5.22 /src/utilities/model\_display.jl

---

```

1 cs_dict=Dict('A'=:green, 'C'=:blue, 'G'=:yellow, 'T'=:red)
2
3 function print_emitters(state::Categorical) #print informative symbols
4     ↳ from the state's emission distribution, if any
5     order=Integer(log(4,length(state.p))-1)
6     alphabet=CompoundAlphabet(ACGT,order)
7     bits=log(2,length(state.p)) #higher order hmms express more
8     ↳ information per symbol
9     dummy_p=state.p.+10^-99 #prevent -Inf or NaN from zero probability
10    ↳ symbols
11    infoscore=(bits+sum(x*log(2,x) for x in dummy_p))
12    infovec=[x*infoscore for x in dummy_p]
13    infovec./=bits
14    print("<<< ")
15    for (symbol,symbol_prob) in enumerate(infovec)
16        if symbol_prob ≥ .05
17            str=string(alphabet.integers[symbol])
18            if symbol_prob ≥ .7
19                for char in str
20                    printstyled(stdout, char; bold=true,
21                               ↳ color=cs_dict[char])
22                end
23            elseif .7 > symbol_prob ≥ .25
24                for char in str
25                    printstyled(stdout, char; color=cs_dict[char])
26                end

```

---

```

23         elseif .25 > symbol_prob
24             for char in str
25                 printstyled(stdout, lowercase(char));
26                     ← color=cs_dict[char])
27             end
28         print(" ")
29     end
30 end
31 println("">>>>")
32 end

```

---

### 17.5.23 /src/utilities/observation\_coding.jl

```

1 #KMER ORDER/SEQUENCE INTEGER CODING UTILITIES
2 #higher order DNA alphabet
3 struct CompoundAlphabet
4     symbols::Dict{Mer{DNAAlphabet{2}}, Integer}
5     integers::Dict{Integer,Mer{DNAAlphabet{2}}}
6     #build a CompoundAlphabet for DNA of some order_no
7     function CompoundAlphabet(alphabet::Tuple, order_no::Integer)
8         symbols = Dict{Mer{DNAAlphabet{2}}, Integer}()
9         integers = Dict{Integer,Mer{DNAAlphabet{2}}}()
10
11         tuples = Array{Tuple}
12         if order_no > 0
13             tuples = collect(Iterators.product([alphabet for order in
14                     ← 0:order_no] ... ))
15         else #0th order compound product of an alphabet is that alphabet
16             tuples = collect(Iterators.product(alphabet))
17         end
18
19         @inbounds for index in eachindex(tuples)
20             tuple_seq = Mer{DNAAlphabet{2}}(collect(tuples[index]))
21             integers[index] = tuple_seq
22             symbols[tuple_seq] = index
23         end
24
25         new(symbols,integers)
26     end
27 end

```

```

28
29 struct N_Order_ntSequence
30     alphabet::CompoundAlphabet
31     seq_lengths::Vector{Integer}
32     order_kmers::Vector{Vector{Mer{DNAAlphabet{2}}}}
33 end
34
35
36 function code_job_obs(job_ids::Vector{Chain_ID},
37     ← obs_sets::Dict{String,Vector{LongSequence{DNAAlphabet{2}}}})
38     code_jobs=Vector{Tuple{String, Integer}}()
39     for id in job_ids #assemble a vector of observation encoding jobs
40         code_job=(id.obs_id, id.order)
41         !(code_job in code_jobs) && push!(code_jobs, code_job)
42     end
43
44     code_dict = Dict{Tuple{String, Integer}, AbstractArray}()
45     @showprogress 1 "Encoding observations ... " for (obs_id, order) in
46         ← code_jobs #build the appropriate sample sets once
47         order_seqs = get_order_n_seqs(obs_sets[obs_id],order) #get the
48             ← kmer sequences at the appropriate order
49         coded_seqs = code_seqs(order_seqs, sorted=true) #numerically code
50             ← the sequences in trainable format
51         code_dict[(obs_id, order)] = coded_seqs
52     end
53     return code_dict
54 end
55
56 #from a vector of LongSequences, get
57
58 function get_order_n_seqs(seqs::Vector{LongSequence{DNAAlphabet{2}}},
59     ← order_no::Integer, base_tuple::Tuple=ACGT)
60     kmer_vecs = Vector{Vector{Mer{DNAAlphabet{2}}}}()
61     length_vec = Vector{Integer}()
62     window = order_no + 1
63
64     for seq in seqs
65         kmer_vec = Vector{Mer{DNAAlphabet{2}}}()
66         @inbounds for (i, kmer) in
67             ← collect(each(Mer{DNAAlphabet{2}},window),seq))
68             push!(kmer_vec, kmer)
69         end
70
71         push!(kmer_vecs, kmer_vec)
72     end
73
74     return kmer_vecs
75 end

```

```

65         push!(length_vec, length(seq))
66     end
67
68     return nordseqs = N_Order_ntSequence(CompoundAlphabet(base_tuple,
69     ↳ order_no), length_vec, kmer_vecs)
70 end
71 #convert tuple kmers to symbol codes
72 function code_seqs(input::N_Order_ntSequence, offsets::Vector=[0 for i in
73     ↳ 1:length(input.order_kmers)]; sorted::Bool=false)
74     symbol_no=length(input.alphabet.symbols)
75     if symbol_no ≤ typemax(UInt8)
76         integer_type = UInt8
77     elseif typemax(UInt8) < symbol_no < typemax(UInt16)
78         integer_type = UInt16
79     else
80         integer_type = UInt32
81     end
82
83     alphabet = input.alphabet
84     output = zeros(integer_type, length(input.order_kmers),
85     ↳ (maximum([length(seq) for seq in input.order_kmers])+1)) #leave 1
86     ↳ missing value after the longest sequence for indexing sequence
87     ↳ length in CLHMM messages
88     sorted && (sort_idxs = sortperm(input.seq_lengths, rev=true))
89     sorted ? input_seqs = input.order_kmers[sort_idxs] : input_seqs =
90     ↳ input.order_kmers
91
92     for (i, seq) in enumerate(input_seqs)
93         for t in 1:length(seq)
94             curr_kmer = seq[t]
95             curr_code = alphabet.symbols[curr_kmer]
96             output[i,t+offsets[i]]=curr_code
97         end
98     end
99     return output
100 end

```

---

### 17.5.24 /src/utilities/utilities.jl

---

```

1 """
2 Utility functions for learning and using background genomic hidden markov
3     models
4 """
5
6 #function to split random sample dataframe into training and test sets
7     (divide total sequence length by half)
8
9 function split_obs_sets(sample_dfs :: Dict{String,DataFrame})
10    training_sets = Dict{String,Vector{LongSequence{DNAAlphabet{2}}}}()
11    test_sets = Dict{String,Vector{LongSequence{DNAAlphabet{2}}}}()
12
13    for (partition_id, partition) in sample_dfs
14        partition = partition[shuffle(1:size(partition, 1)),:] #shuffle
15            → the samples to avoid any effect from sampling without
16            → replacement
17        partition.sampleLength = (partition.SampleEnd -
18            → partition.SampleStart) .+ 1
19        midway = sum(partition.sampleLength)÷2
20        split_index = 0
21        counter = 0
22        while split_index == 0
23            counter += 1
24            length_sum = sum(partition.sampleLength[1:counter])
25            if length_sum > midway
26                split_index = counter
27            end
28        end
29
30        training_sets[partition_id] =
31            → partition.SampleSequence[1:split_index-1]
32        test_sets[partition_id] =
33            → partition.SampleSequence[split_index:end]
34    end
35    return training_sets, test_sets
36 end

```

---

### 17.5.25 /test/ref\_fns.jl

---

```

1 #slow algo written in mimicry of Churbanov & Winters
2 function old_linear(hmm, observations, obs_lengths)
3     O = size(observations)[2]

```

```

4     a = log.(hmm.A); a = log.(hmm.a)
5     N = length(hmm.B); B = length(hmm.B[1].support); b =
6         ↪ [log(hmm.B[m].p[y]) for m in 1:N, y in 1:B]
7     α1oi = zeros(0,N); β1oi = zeros(0,N); Eoi = zeros(0,N,B); Toij =
8         ↪ zeros(0,N,N); Aoi = zeros(0,N); log_pobs=zeros(0); γt=0
9
10    for o in 1:O
11        #INITIALIZATION
12        T = obs_lengths[o]; βoi_T = zeros(N) #log betas at T initialised
13            ↪ as zeros
14        EiT = fill(-Inf,B,N,N); TijT = fill(-Inf,N,N,N) #Ti,j(T,m) = 0
15            ↪ for all m; in logspace
16
17        @inbounds for m in 1:N, i in 1:N, y in 1:B
18            observations[T, o] = y && m == i ? (EiT[y, i, m] = 0) :
19                (EiT[y, i, m] = -Inf) #log Ei initialisation
20        end
21
22        #RECURRENCE
23        @inbounds for t in T-1:-1:1
24            βoi_t = similar(βoi_T); Tijt = similar(TijT); Eit =
25                ↪ similar(EiT)
26            Γ = observations[t+1,o]; γt = observations[t,o]
27            Γ==0 && println("hooo !! $o $t")
28            for m in 1:N
29                βoi_t[m] = logsumexp([lps(a[m,j], b[j,Γ], βoi_T[j]) for j
30                    ↪ in 1:N])
31                for i in 1:N
32                    for j in 1:N
33                        Tijt[i, j, m] = logsumexp([lps(a[m,n], TijT[i, j,
34                            ↪ n], b[n,Γ]) for n in 1:N])
35                        i==m && (Tijt[i, j, m] = logaddexp(Tijt[i, j, m],
36                            ↪ lps(βoi_T[j], a[m,j], b[j, Γ])))
37                    end
38                    for y in 1:B
39                        Eit[y, i, m] = logsumexp([lps(b[n,Γ], a[m,n],
40                            ↪ EiT[y, i, n]) for n in 1:N])
41                        i==m && y==γt && (Eit[y, i, m] = logaddexp(Eit[y,
42                            ↪ i, m], βoi_t[m])))
43                    end
44                end
45            end
46            βoi_T = βoi_t; TijT = Tijt; EiT = Eit

```

```

37     end
38
39     #TERMINATION
40     Γ = observations[1,o]
41     α1oi[o,:]= [lps(a[i], b[i, Γ]) for i in 1:N]
42     β1oi[o,:]= βoi_T
43     log_pobs[o] = logsumexp(lps.(α1oi[o,:], βoi_T[:]))
44     Eoi[o,:,:]= [logsumexp([lps(EiT[y,i,m], a[m], b[m,yt]) for m in
45     ↪ 1:N]) for i in 1:N, y in 1:B]
46     Toij[o,:,:]= [logsumexp([lps(TijT[i,j,m], a[m], b[m,yt]) for m
47     ↪ in 1:N]) for i in 1:N, j in 1:N]
48   end
49
50   #INTEGRATE ACROSS OBSERVATIONS AND SOLVE FOR NEW BHMM PARAMS
51   obs_penalty=log(0)
52   a_o=α1oi.+β1oi.-logsumexp.(eachrow(α1oi.+β1oi))
53   new_a=logsumexp.(eachcol(a_o))-obs_penalty
54
55   a_int = Toij.-logsumexp.([Toij[o,i,:] for o in 1:O, i in 1:N])
56   new_a = logsumexp.([a_int[:,i,j] for i in 1:N, j in
57     ↪ 1:N])-obs_penalty
58
59   e_int=Eoi.-logsumexp.([Eoi[o,j,:] for o in 1:O, j in 1:N])
60   new_b=logsumexp.([e_int[:,j,d] for d in 1:B, j in 1:N])-obs_penalty
61   new_D=[Categorical(exp.(new_b[:,i])) for i in 1:N]
62
63   return typeof(hmm)(exp.(new_a), exp.(new_a), new_D), lps(log_pobs)
64 end
65
66
67
68
69
70
71
72
73
74
75
76
#HMMBase
function mouchet_mle_step(hmm::BHMM, observations) where F
    # NOTE: This function works but there is room for improvement.

    log_likelihoods = mouchet_log_likelihoods(hmm, observations)

    log_α = mouchet_messages_forwards_log(hmm.a, hmm.A, log_likelihoods)
    log_β = mouchet_messages_backwards_log(hmm.A, log_likelihoods)
    log_A = log.(hmm.A)

    normalizer = logsumexp(log_α[1,:] + log_β[1,:])

    # E-step

```

```

77
78     T, K = size(log_likelihoods)
79     log_ξ = zeros(T, K, K)
80
81     @inbounds for t = 1:T-1, i = 1:K, j = 1:K
82         log_ξ[t,i,j] = log_α[t,i] + log_A[i,j] + log_β[t+1,j] +
83             ↳ log_likelihoods[t+1,j] - normalizer
84     end
85
86     ξ = exp.(log_ξ)
87     ξ ./= sum(ξ, dims=[2,3])
88
89     # M-step
90     new_A = sum(ξ[1:end-1,:,:], dims=1)[1,:,:] #index fix-MM
91     new_A ./= sum(new_A, dims=2)
92
93     new_a = exp.((log_α[1,:] + log_β[1,:]) ∘- normalizer)
94     new_a ./= sum(new_a)
95
96     # TODO: Cleanup/optimize this part
97     γ = exp.((log_α .+ log_β) ∘- normalizer)
98
99     B = Categorical[]
100    for (i, d) in enumerate(hmm.B)
101        # Super hacky ...
102        # https://github.com/JuliaStats/Distributions.jl/issues/809
103        push!(B, fit_mle(Categorical, permutedims(observations), γ[:,i]))
104    end
105
106    typeof(hmm)(new_a, new_A, B), normalizer
107
108
109 function mouchet_log_likelihoods(hmm, observations)
110     hcat(map(d → logpdf.(d, observations), hmm.B) ... )
111 end
112
113
114
115 function mouchet_messages_forwards_log(init_distn, trans_matrix,
116     ↳ log_likelihoods)
117     # OPTIMIZE
118     log_alphas = zeros(size(log_likelihoods))

```

```

118 log_trans_matrix = log.(trans_matrix)
119 log_alphas[1,:] = log.(init_distn) .+ log_likelihoods[1,:]
120 @inbounds for t = 2:size(log_alphas)[1]
121     for i in 1:size(log_alphas)[2]
122         log_alphas[t,i] = logsumexp(log_alphas[t-1,:] .+
123             → log_trans_matrix[:,i]) + log_likelihoods[t,i]
124     end
125 end
126 log_alphas
127
128 function mouchet_messages_backwards_log(trans_matrix, log_likelihoods)
129     # OPTIMIZE
130     log_betas = zeros(size(log_likelihoods))
131     log_trans_matrix = log.(trans_matrix)
132     @inbounds for t = size(log_betas)[1]-1:-1:1
133         tmp = view(log_betas, t+1, :) .+ view(log_likelihoods, t+1, :)
134         @inbounds for i in 1:size(log_betas)[2]
135             log_betas[t,i] = logsumexp(view(log_trans_matrix, i, :) .+
136                 → tmp)
137         end
138     log_betas
139 end

```

---

### 17.5.26 /test/runtests.jl

---

```

1 using
   → BioBackgroundModels,BioSequences,DataFrames,Distributed,Distributions,FASTX,Program
2 import BioBackgroundModels: autotransition_init, load_balancer,
   → CompoundAlphabet, get_order_n_seqs, code_seqs, code_job_obs,
   → make_padded_df, add_partition_masks!, partition_genome_coordinates,
   → divide_partitions_by_scaffold, mask_sequence_by_partition,
   → find_position_partition, setup_sample_jobs, rectify_identifier,
   → add_metacoordinates!, meta_to_feature_coord, get_feature_row_index,
   → get_strand_dict, build_scaffold_seq_dict,
   → get_feature_params_from_metacoord, determine_sample_window,
   → fetch_sequence, get_sample_set, get_sample, assert_hmm, linear_step,
   → get_BGHMM_symbol_lh,make_padded_df,add_partition_masks!,BGHMM_likelihood_calc,lin
   → t_add_categorical_counts!
3 import Serialization: deserialize
4 include("synthetic_sequence_gen.jl")

```

```

5 include("ref_fns.jl")
6
7 #JOB FILEPATHS
8 #GFF3 feature database, FASTA genome and index paths
9 Sys.islinux() ? genome = (@__DIR__) * "/synthetic.fna" : genome =
    ↵ (@__DIR__) * "\\synthetic.fna"
10 Sys.islinux() ? index = (@__DIR__) * "/synthetic.fna.fai" : index =
    ↵ (@__DIR__) * "\\synthetic.fna.fai"
11 Sys.islinux() ? gff = (@__DIR__) * "/synthetic.gff3" : gff = (@__DIR__)
    ↵ * "\\synthetic.gff3"
12 Sys.islinux() ? posfasta = (@__DIR__) * "/syntheticpos.fa" : posfasta =
    ↵ (@__DIR__) * "\\syntheticpos.fa"
13
14 !isfile(genome) && print_synthetic_fasta(genome)
15 !isfile(index) && print_synthetic_index(index)
16 !isfile(gff) && print_synthetic_gff(gff)
17 !isfile(posfasta) && print_synthetic_position(posfasta)
18
19 Random.seed!(1)
20
21 @testset "BHMM/BHMM.jl constructors" begin
22     good_a = [.20,.30,.20,.30]
23     bad_probvec_a = [.25,.25,.25,.27]
24     good_A = fill(.25,4,4)
25     bad_probvec_A = deepcopy(good_A)
26     bad_probvec_A[:,3] *= .27
27     bad_size_A = fill(.25,5,4)
28     good_D = Categorical([.25,.25,.25,.25])
29     bad_size_D = Categorical([.2,.2,.2,.2,.2])
30     good_Dvec = [good_D for i in 1:4]
31     bad_size_Dvec =[good_D, good_D, good_D, bad_size_D]
32     bad_length_Dvec=[good_D for i in 1:5]
33
34     @test_throws ArgumentError BHMM(bad_probvec_a, good_A, good_Dvec)
35     @test_throws ArgumentError BHMM(good_a, bad_probvec_A, good_Dvec)
36     @test_throws ArgumentError BHMM(good_a, bad_size_A, good_Dvec)
37     @test_throws ArgumentError BHMM(good_a, good_A, bad_size_Dvec)
38     @test_throws ArgumentError BHMM(good_a, good_A, bad_length_Dvec)
39
40     hmm=BHMM(good_a, good_A, good_Dvec)
41     @test typeof(hmm)=BHMM{Float64}
42     @test hmm.a==good_a
43     @test hmm.A==good_A

```

```

44 @test hmm.B==good_Dvec
45 @test size(hmm) == (4,4)

46
47 hmm_copy=copy(hmm)
48 @test hmm.a==hmm_copy.a
49 @test hmm.A==hmm_copy.A
50 @test hmm.B==hmm_copy.B

51
52 hmm2=BHMM(good_A, good_Dvec)
53 @test hmm2.a==[.25,.25,.25,.25]
54 @test hmm2.A==good_A
55 @test hmm2.B==good_Dvec

56 end

57
58 @testset "BHMM/chain.jl Chain_ID and EM_step constructors" begin
59   obs_id="test"
60   K,zero_k,neg_K=4,0,-1
61   order,badorder=0,-1
62   replicate,zero_rep,neg_rep=1,0,-1

63
64   @test_throws ArgumentError Chain_ID(obs_id, zero_k, order, replicate)
65   @test_throws ArgumentError Chain_ID(obs_id, neg_K, order, replicate)
66   @test_throws ArgumentError Chain_ID(obs_id, K, badorder, replicate)
67   @test_throws ArgumentError Chain_ID(obs_id, K, order, zero_rep)
68   @test_throws ArgumentError Chain_ID(obs_id, K, order, neg_rep)

69
70   id=Chain_ID(obs_id, K, order, replicate)
71   @test typeof(id)==Chain_ID
72   @test id.obs_id==obs_id
73   @test id.K==K
74   @test id.order==order
75   @test id.replicate==replicate

76
77   good_A = fill(.25,4,4)
78   good_D = Categorical([.25,.25,.25,.25])
79   good_Dvec = [good_D for i in 1:4]
80   hmm=BHMM(good_A,good_Dvec)
81   log_p=-.001

82
83   @test_throws ArgumentError EM_step(0,hmm,0.0,0.0,true)
84   @test_throws ArgumentError EM_step(-1,hmm,0.0,0.0,true)
85   @test_throws ArgumentError EM_step(1,hmm,1.0,0.0,true)

86

```

```

87  steppe=EM_step(1,hmm,log_p,0.0,false)
88  @test typeof(steppe)=EM_step
89  @test steppe.iterate==1
90  @test steppe.hmm=hmm
91  @test steppe.log_p=log_p
92  @test steppe.delta==0.0
93  @test steppe.converged==false
94 end
95
96 @testset "utilities/log_prob_sum.jl Log probability summation functions"
97   begin
98     @test isapprox(lps([.1, .1, .1]), .3)
99     @test lps([-Inf, .1, .1]) == -Inf
100    @test isapprox(lps(.1, .1, .1), .3)
101    @test lps(-Inf, .1, .1) == -Inf
102    @test lps(.1, -Inf, .1) == -Inf
103 end
104 @testset "utilities/HMM_init.jl initialization functions" begin
105   K=4
106   order=2
107   hmm=autotransition_init(K,order)
108   @test typeof(hmm)==BHMM{Float64}
109   @test size(hmm)==(4,64)
110 end
111
112 @testset "utilities/load_balancer.jl EM_converge load balancing structs
113   and functions" begin
114   @test_throws ArgumentError LoadConfig(0:5,1:5)
115   @test_throws ArgumentError LoadConfig(1:5,-1:5)
116
117   job_ids=[Chain_ID("test",6,2,1),Chain_ID("test",4,2,1),Chain_ID("test",2,0,1)]
118
119   obs_sets=Dict("test"=>[LongSequence{DNAAlphabet{2}}]("AAAAAAAAAAAAAA"))
120
121   K_exclusive_config=LoadConfig(1:1,0:0)
122   O_exclusive_config=LoadConfig(1:6,3:4)
123   blacklist_config=LoadConfig(1:6,0:2,blacklist=job_ids)
124
125   whitelist_config=LoadConfig(1:6,0:2,whitelist=[Chain_ID("test",6,2,2)])
126   high_config=LoadConfig(4:6,0:2,whitelist=job_ids)
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
937
938
939
939
940
941
942
943
944
945
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
957
958
959
959
960
961
962
963
964
965
966
967
967
968
969
969
970
971
972
973
974
975
976
977
977
978
979
979
980
981
982
983
984
985
986
986
987
988
988
989
989
990
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1710
1711
1711
1712
1712
1713
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1720
1721
1721
1722
1722
1723
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1730
1731
1731
1732
1732
1733
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1740
1741
1741
1742
1742
1743
1743
1744
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1750
1751
1751
1752
1752
1753
1753
1754
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1760
1761
1761
1762
1762
1763
1763
1764
1764
1765
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1770
1771
1771
1772
1772
1773
1773
1774
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1780
1781
1781
1782
1782
1783
1783
1784
1784
1785
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1790
1791
1791
1792
1792
1793
1793
1794
1794
1795
1795
1796
1796
17
```

```

125     no_input_hmms, chains, input_channel, output_channel =
126         ↳ setup_EM_jobs!(job_ids,obs_sets)
127 @test load_balancer(no_input_hmms, input_channel, K_exclusive_config,
128         ↳ 3) = (0,0,0,0,0)
129
130     no_input_hmms, chains, input_channel, output_channel =
131         ↳ setup_EM_jobs!(job_ids,obs_sets)
132 @test load_balancer(no_input_hmms, input_channel, O_exclusive_config,
133         ↳ 3) = (0,0,0,0,0)
134
135     no_input_hmms, chains, input_channel, output_channel =
136         ↳ setup_EM_jobs!(job_ids,obs_sets)
137 @test load_balancer(no_input_hmms, input_channel, blacklist_config,
138         ↳ 3) = (0,0,0,0,0)
139
140     no_input_hmms, chains, input_channel, output_channel =
141         ↳ setup_EM_jobs!(job_ids,obs_sets)
142 high_set=load_balancer(no_input_hmms, input_channel, high_config)
143
144     @test high_set[1]=Chain_ID("test",6,2,1)
145     @test high_set[2]=1
146     @test typeof(high_set[3])=BHMM{Float64}
147     @test
148         ↳ high_set[4]=obs_lh_given_hmm(high_set[5],high_set[3],linear=false)
149     @test typeof(high_set[5])=Matrix{UInt8}
150
151 end
152
153
154 @testset "utilities/observation_coding.jl Order coding structs and
155     ↳ functions" begin
156     compound_alphabet=CompoundAlphabet(ACGT, 2)
157     @test
158         ↳ typeof(compound_alphabet.symbols)=Dict{Mer{DNAAlphabet{2}},Integer}
159     @test length(compound_alphabet.symbols)=64
160     @test compound_alphabet.symbols[Mer{DNAAlphabet{2}}("AAA")]=1
161     @test compound_alphabet.symbols[Mer{DNAAlphabet{2}}("TTT")]=64
162
163     test_seqs =
164         ↳ [BioSequences.LongSequence{DNAAlphabet{2}}("ACGTACGTACGTACGT"),BioSequences.L

```

```

155 target0= [1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 0; 4 4 4 4 4 4 4 0 0 0 0 0 0
156   ↵ 0 0 0 0 0]
157 target2= [37 58 15 20 37 58 15 20 37 58 15 20 37 58 0; 64 64 64 64 64
158   ↵ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
159
160 order0_seqs = get_order_n_seqs(test_seqs,0)
161 @test typeof(order0_seqs)=BioBackgroundModels.N_Order_ntSequence
162 for (mer, code) in CompoundAlphabet(ACGT,0).symbols
163   @test order0_seqs.alphabet.symbols[mer]=code
164 end
165 @test order0_seqs.seq_lengths=[16,7]
166 @test order0_seqs.order_kmers[1][1]=Mer{DNAAlphabet{2}}("A")
167 code0_seqs = code_seqs(order0_seqs)
168 @test target0 = code0_seqs
169
170 order2_seqs = get_order_n_seqs(test_seqs,2)
171 @test typeof(order2_seqs)=BioBackgroundModels.N_Order_ntSequence
172 for (mer, code) in CompoundAlphabet(ACGT,2).symbols
173   @test order2_seqs.alphabet.symbols[mer]=code
174 end
175 @test order2_seqs.seq_lengths=[16,7]
176 @test order2_seqs.order_kmers[1][1]=Mer{DNAAlphabet{2}}("ACG")
177 code2_seqs = code_seqs(order2_seqs)
178 @test target2 = code2_seqs
179
180 bw_o0_seqs = get_order_n_seqs(test_seqs[end:-1:1], 0)
181 sorted_code_seqs = code_seqs(bw_o0_seqs, sorted=true)
182 @test target0 = sorted_code_seqs
183
184 obs_sets=Dict("test1"⇒test_seqs,
185                 "test2"⇒test_seqs)
186
187   ↵ job_ids=[Chain_ID("test1",4,0,1),Chain_ID("test2",4,0,1),Chain_ID("test1",2,2
188
189 code_dict=code_job_obs(job_ids,obs_sets)
190 @test ("test1",0) in keys(code_dict)
191 @test ("test1",2) in keys(code_dict)
192 @test ("test2",0) in keys(code_dict)
193 @test !(("test2",2) in keys(code_dict))
194 @test code_dict[("test1",0)]=target0
195 @test code_dict[("test1",2)]=target2
196 @test code_dict[("test2",0)]=target0

```

```

195 #test type selection for high order numbers
196 o5_nos=get_order_n_seqs(test_seqs,5)
197 o5_codes=code_seqs(o5_nos)
198 @test typeof(o5_codes)=Matrix{UInt16}

199
200 o7_nos=get_order_n_seqs(test_seqs,7)
201 o7_codes=code_seqs(o7_nos)
202 @test typeof(o7_codes)=Matrix{UInt32}

203
204 end

205
206 @testset "genome_sampling/partition_masker.jl functions" begin
207 synthetic_seq =
208     → BioSequences.LongSequence{DNAAlphabet{2}}(generate_synthetic_seq())
209 position_start = 501
210 position_length=141
211 position_pad=350
212 perigenic_pad = 250

213 position_df = make_padded_df(posfasta, gff, genome, index,
214     → position_pad)
215 add_partition_masks!(position_df, gff, perigenic_pad)

216 @test position_df.SeqID[1]=="1"
217 @test length(position_df.PadSeq[1]) = position_pad+position_length
218     → =
219     → length(position_df.PadStart[1]:position_df.PadStart[1]+position_length+positi
220     → = position_df.End[1]-position_df.Start[1]+1+position_pad =
221     → size(position_df.MaskMatrix[1])[1]
222 @test
223     → synthetic_seq[position_df.PadStart[1]:position_df.End[1]]=position_df.PadSeq

224 mm = position_df.MaskMatrix[1]
225 idx=1#pos 151
226 @test sum(mm[idx:idx+99,1] .== 1)=length(mm[idx:idx+99,1])
227 idx+=100#pos 251
228 @test sum(mm[idx:idx+259,1] .== 2)=length(mm[idx:idx+259,1])
229 idx+=260 #pos 511
230 @test sum(mm[idx:idx+59,1] .== 3)=length(mm[idx:idx+59,1])
231 idx+=60#pos 571
232 @test sum(mm[idx:idx+29,1] .== 2)=length(mm[idx:idx+29,1])
233 idx+=30#pos601
234 @test sum(mm[idx:idx+40,1] .== 3)=length(mm[idx:idx+40,1])

```

```

231
232     #test exception
233     position_df = make_padded_df(posfasta, gff, genome, index,
234     ↳ position_pad)
235
236     partition_coords_dict=partition_genome_coordinates(gff,
237     ↳ perigenic_pad)
238
239     ↳ partitioned_scaffolds=divide_partitions_by_scaffold(partition_coords_dict)
240     scaffold_coords_dict = Dict{String,DataFrame}()
241     scaffold="1"
242     for ((partition, part_scaffold), df) in partitioned_scaffolds
243         if scaffold == part_scaffold
244             scaffold_coords_dict[partition] = df
245         end
246     end
247
248     @test_throws DomainError
249     ↳ mask_sequence_by_partition(1001,151,scaffold_coords_dict)
250 end
251
252 @testset "genome_sampling/sequence_sampler.jl functions &
253     ↳ API/genome_sampling.jl interface" begin
254     #rectifier tests
255     @test rectify_identifier("1")=="1"
256
257     partitions = 3 #exonic, periexonic, intragenic
258     sample_set_length=100
259     min_sample_window=5
260     max_sample_window=25
261     perigenic_pad=250
262     syn_intergenic_starts = [1]
263     syn_intergenic_ends = [250]
264     syn_periexonic_starts = [251,571,661,761,861,961]
265     syn_periexonic_ends = [510,600,700,800,900,1000]
266     syn_periexonic_strands = [ '-', '-' , '-' , '-' , '-' , '-' ]
267     syn_exonic_starts = [511,601,701,801,901]
268     syn_exonic_ends = [570,660,760,860,960]
269     syn_exonic_strands = [ '+', '-' , '-' , '-' , '-' ]
270     partition_lengths =
271     ↳ Dict("exon"⇒5*60,"intergenic"⇒250,"periexonic"⇒450)
272
273     # "Testing sequence sampler fns ... "

```

```

268     synthetic_seq =
269         → BioSequences.LongSequence{DNAAlphabet{2}}(generate_synthetic_seq())
270
271     # "Partitioning synthetic genome coordinates ... "
272     coordinate_partitions = partition_genome_coordinates(gff,
273         → perigenic_pad)
274
275     @test coordinate_partitions["intergenic"].Start =
276         → syn_intergenic_starts
277     @test coordinate_partitions["intergenic"].End = syn_intergenic_ends
278
279     @test coordinate_partitions["periexonic"].Start =
280         → syn_periexonic_starts
281     @test coordinate_partitions["periexonic"].End = syn_periexonic_ends
282     @test coordinate_partitions["periexonic"].Strand =
283         → syn_periexonic_strands
284
285     # "Checking sampling functions at all synthetic indices ... "
286
287     input_test_channel, completed_test_channel, progress_channel,
288         → setlength = setup_sample_jobs(genome, index, gff,
289             → sample_set_length, min_sample_window, max_sample_window,
290             → perigenic_pad; deterministic=true)
291     while isready(input_test_channel)
292         genome_path, genome_index_path, partition_df, partitionid,
293             → sample_set_length, sample_window_min, sample_window_max,
294             → deterministic = take!(input_test_channel)
295
296         for feature in eachrow(partition_df)
297             seqid, scaffold_start, scaffold_end =
298                 → meta_to_feature_coord(feature.MetaStart, feature.MetaEnd,
299                     → partition_df)
300             @test scaffold_start == feature.Start && scaffold_end ==
301                 → feature.End
302         end
303
304         stranded = get_strand_dict()[partitionid]
305         @test typeof(stranded) == Bool
306
307

```

```

298     scaffold_sequence_record_dict = build_scaffold_seq_dict(genome,
299     ↪ index)
300
301     @test scaffold_sequence_record_dict["1"] == synthetic_seq
302
303
304     partition_length = partition_df.MetaEnd[end]
305     @test partition_length == partition_lengths[partitionid]
306
307
308     metacoordinate_bitarray = trues(partition_df.MetaEnd[end])
309
310
311     for bitindex in findall(metacoordinate_bitarray)
312         feature_metaStart, feature_metaEnd, strand =
313             ↪ get_feature_params_from_metacoord(bitindex, partition_df,
314             ↪ stranded)
315
316         @test 1 ≤ feature_metaStart < feature_metaEnd ≤
317             ↪ length(metacoordinate_bitarray)
318         @test feature_metaStart in partition_df.MetaStart
319         @test feature_metaEnd in partition_df.MetaEnd
320
321         if partitionid == "periexonic"
322             @test strand == '-'
323         elseif partitionid == "exon"
324             @test strand in ['-','+']
325         end
326
327         feature_length = length(feature_metaStart:feature_metaEnd)
328
329         window = determine_sample_window(feature_metaStart,
330             ↪ feature_metaEnd, bitindex, metacoordinate_bitarray,
331             ↪ sample_window_min, sample_window_max) #get an appropriate
332             ↪ sampling window around the selected index, given the
333             ↪ feature boundaries and params
334
335         @test 1 ≤ feature_metaStart ≤ window[1] <
336             ↪ window[1]+sample_window_min-1 <
337             ↪ window[1]+sample_window_max-1 ≤ window[2] ≤
338             ↪ feature_metaEnd ≤ length(metacoordinate_bitarray)
339
340         sample_scaffoldid, sample_scaffold_start, sample_scaffold_end
341             ↪ = meta_to_feature_coord(window[1],window[2],partition_df)
342
343         @test sample_scaffoldid == "1"
344
345         @test 1 ≤ sample_scaffold_start ≤
346             ↪ sample_scaffold_start+sample_window_min ≤
347             ↪ sample_scaffold_end ≤
348             ↪ min(sample_scaffold_start+sample_window_max,1000) ≤ 1000
349
350
351         strand == '-' ?
352             ↪ target_seq=reverse_complement(synthetic_seq[sample_scaffold_start:sam
353             ↪ :

```

```
324     target_seq =
325         → synthetic_seq[sample_scaffold_start:sample_scaffold_end]
326
326     proposal_sequence = fetch_sequence(sample_scaffoldid,
327         → scaffold_sequence_record_dict, sample_scaffold_start,
328         → sample_scaffold_end, strand; deterministic=deterministic)
329         → #get the sequence associated with the sample window
330
331
332     @test proposal_sequence == target_seq
333 end
334
335 # "Verifying sampling channels ... "
336
337 input_sample_channel, completed_sample_channel, progress_channel,
338     → setlength = setup_sample_jobs(genome, index, gff,
339     → sample_set_length, min_sample_window, max_sample_window,
340     → perigenic_pad; deterministic=true)
341 get_sample_set(input_sample_channel, completed_sample_channel,
342     → progress_channel)
343
344 #collect sample dfs by partition id when ready
345 collected_counter = 0
346 sample_record_dfs = Dict{String,DataFrame}()
347 while collected_counter < partitions
348     wait(completed_sample_channel)
349     partition_id, sample_df = take!(completed_sample_channel)
350     sample_record_dfs[partition_id] = sample_df
351     collected_counter += 1
352 end
353
354 #get_sample entire feature selection
355 coordinate_partitions = partition_genome_coordinates(gff,
356     → perigenic_pad)
357 partition_id="exon"
358 partition=coordinate_partitions[partition_id]
359 add_metacoordinates!(partition)
360 metacoordinate_bitarray=true(partition.MetaEnd[end])
361 scaffold_sequence_record_dict = build_scaffold_seq_dict(genome,
362     → index)
363
364 #test fetch_sequence strand char ArgumentError
```



```

392 testseq[1:4] = [1,2,3,4]
393 @test isapprox(get_BGHMM_symbol_lh(testseq, hmm)[1], log.(pvec[1]))
394 @test
395     ↳ isapprox(obs_lh_given_hmm(testseq,hmm),obs_lh_given_hmm(testseq,hmm,linear=false))
396 @test isapprox(sum(get_BGHMM_symbol_lh(testseq,hmm)),
397     ↳ obs_lh_given_hmm(testseq,hmm))
398 @test isapprox(sum(get_BGHMM_symbol_lh(testseq,hmm)),
399     ↳ obs_lh_given_hmm(testseq,hmm,linear=false))

400 pvec=[.25,.25,.25,.25]
401 trans = [.9 .1
402             .1 .9]
403 B = [Categorical(pvec), Categorical(pvec)]
404 hmm = BHMM(trans, B)

405 @test
406     ↳ isapprox(obs_lh_given_hmm(testseq,hmm),obs_lh_given_hmm(testseq,hmm,linear=false))
407 @test isapprox(sum(get_BGHMM_symbol_lh(testseq,hmm)),
408     ↳ obs_lh_given_hmm(testseq,hmm))
409 @test isapprox(sum(get_BGHMM_symbol_lh(testseq,hmm)),
410     ↳ obs_lh_given_hmm(testseq,hmm,linear=false))

411 testseq=zeros(Int64,1,1001)
412 testseq[1,1:1000]=rand(1:4,1000)

413 @test isapprox(sum(get_BGHMM_symbol_lh(testseq,
414     ↳ hmm)),obs_lh_given_hmm(testseq,hmm))

415 Dex = [Categorical([.3, .1, .3, .3]),Categorical([.15, .35, .35,
416             .15])]
417 Dper = [Categorical([.15, .35, .35, .15]),Categorical([.4, .1, .1,
418             .4])]
419 Dint = [Categorical([.4, .1, .1, .4]),Categorical([.45, .05, .05,
420             .45])]
421 BGHMM_dict = Dict{String,BHMM}()
422 BGHMM_dict["exon"] = BHMM(trans, Dex)
423 BGHMM_dict["periexonic"] = BHMM(trans, Dper)
424 BGHMM_dict["intergenic"] = BHMM(trans, Dint)

425 position_length=141;perigenic_pad=250;
426 position_df = make_padded_df(posfasta, gff, genome, index,
427     ↳ position_length)

```

```

424     reader=open(FASTA.Reader, genome, index=index)
425     seq=FASTA.sequence(reader[ "CM002885.2.1"])
426     @test position_df.Start[1]==501
427     @test position_df.End[1]==641
428     @test position_df.PadSeq[1]==seq[360:641]
429     @test position_df.PadStart[1]==360
430     @test position_df.RelStart[1]==141
431     @test position_df.SeqOffset[1]==0
432
433     offset_position_df = make_padded_df(posfasta, gff, genome, index, 600)
434     @test offset_position_df.Start[1]==501
435     @test offset_position_df.End[1]==641
436     @test offset_position_df.PadSeq[1]==seq[1:641]
437     @test offset_position_df.PadStart[1]==1
438     @test offset_position_df.RelStart[1]==500
439     @test offset_position_df.SeqOffset[1]==100
440
441     add_partition_masks!(position_df, gff, perigenic_pad)
442     @test all(position_df.MaskMatrix[1][1:151,1]==2)
443     @test all(position_df.MaskMatrix[1][152:211,1]==3)
444     @test all(position_df.MaskMatrix[1][212:241,1]==2)
445     @test all(position_df.MaskMatrix[1][242:end,1]==3)
446     @test all(position_df.MaskMatrix[1][1:151,2]==-1)
447     @test all(position_df.MaskMatrix[1][152:211,2]==1)
448     @test all(position_df.MaskMatrix[1][212:end,2]==-1)
449
450     lh_matrix=BGHMM_likelihood_calc(position_df,BGHMM_dict)
451     @test size(lh_matrix)==(position_length*2,1)
452
453
454     → periexonic_frag=reverse_complement!(LongSequence{DNAAlphabet{2}})(seq[360:510])
455     pno=get_order_n_seqs([periexonic_frag],0)
456     pcode=code_seqs(pno)
457     plh=get_BGHMM_symbol_lh(pcode, BGHMM_dict["periexonic"])
458     @test isapprox(lh_matrix[1:151,1], reverse(plh))
459
460     exonic_frag=LongSequence{DNAAlphabet{2}}(seq[511:570])
461     eno=get_order_n_seqs([exonic_frag],0)
462     ecode=code_seqs(eno)
463     elh=get_BGHMM_symbol_lh(ecode, BGHMM_dict["exon"])
464     @test isapprox(lh_matrix[152:211,1], elh)
465 end

```

```

466 @testset "EM/baum-welch.jl MS_HMMBase equivalency and functions" begin
467     A = [.5 .5
468            .5 .5]
469     B = [Categorical(ones(4)/4), Categorical([.7,.1,.1,.1])]
470     hmm = BHMM(A, B)
471     log_A = log.(hmm.A)
472
473     obs = zeros(UInt8, 22,1)
474     obs[1:21] = [4,3,3,2,3,2,1,1,2,3,3,3,4,4,2,3,2,3,4,3,2]
475     obs_lengths=[21]
476
477     lls = BioBackgroundModels.bw_llhs(hmm,obs)
478     m_lls = mouchet_log_likelihoods(hmm,obs[1:21])
479
480     K,Tmaxplus1,0 = size(lls)
481     T=Tmaxplus1-1
482
483     #TEST LOG LIKELIHOOD FN
484     @test transpose(lls[:,1:end-1,1]) == m_lls
485
486     log_α = BioBackgroundModels.messages_forwards_log(hmm.a, hmm.A, lls,
487             ↳ obs_lengths)
487     m_log_α = mouchet_messages_forwards_log(hmm.a, hmm.A, m_lls)
488
489     #TEST FORWARD MESSAGES FN
490     @test transpose(log_α[:,1:end-1,1]) == m_log_α
491
492     log_β = BioBackgroundModels.messages_backwards_log(hmm.A, lls,
493             ↳ obs_lengths)
493     m_log_β = mouchet_messages_backwards_log(hmm.A, m_lls)
494
495     #TEST BACKWARDS MESSAGES FN
496     @test isapprox(transpose(log_β[:,1:end-1,1]), m_log_β)
497
498     lls = permutedims(lls, [2,3,1]) # from (K,T,0) to (T,0,K)
499     log_α = permutedims(log_α, [2,3,1])
500     log_β = permutedims(log_β, [2,3,1])
501
502     # "Testing obs probability calcs ... "
503     #TEST OBSERVATION PROBABILITY CALC
504     normalizer = logsumexp(m_log_α[1,:] + m_log_β[1,:])
505     log_pobs = logsumexp(lps.(log_α[1,1,:], log_β[1,1,:]))
506

```

```

507 @test isapprox(normalizer, log_pobs)
508
509 # "Testing ξ, γ calcs ... "
510
511 log_ξ = fill(-Inf, Tmaxplus1,0,K,K)
512 log_γ = fill(-Inf, Tmaxplus1,0,K)
513
514 m_log_ξ = zeros(T, K, K)
515
516 for t = 1:T-1, i = 1:K, j = 1:K
517     m_log_ξ[t,i,j] = m_log_α[t,i] + log_A[i,j] + m_log_β[t+1,j] +
518         ↪ m_lls[t+1,j] - normalizer
519 end
520
521 for i = 1:K, j = 1:K, o = 1:0
522     obsl = obs_lengths[o]
523     for t = 1:obsl-1 #log_ξ & log_γ calculated to T-1 for each o
524         log_ξ[t,o,i,j] = lps(log_α[t,o,i], log_A[i,j], log_β[t+1,o,j],
525             ↪ lls[t+1,o,j], -log_pobs[o])
526         log_γ[t,o,i] = lps(log_α[t,o,i], log_β[t,o,i], -log_pobs[o])
527     end
528     t=obsl #log_ξ @ T = 0
529     log_ξ[t,o,i,j] = 0
530     log_γ[t,o,i] = lps(log_α[t,o,i], log_β[t,o,i], -log_pobs)
531 end
532
533 ξ = exp.(log_ξ)
534 k_ξ = sum(ξ, dims=[3,4])
535 nan_mask = k_ξ .== 0; k_ξ[nan_mask] .= Inf #prevent NaNs in dummy
536     ↪ renorm arising from multiple sequences indexing
537 ξ ./= k_ξ #dummy renorm across K to keep numerical creep from
538     ↪ causing isprobvec to fail on new new_A during hmm creation
539
540 m_ξ = exp.(m_log_ξ)
541 m_ξ ./= sum(m_ξ, dims=[2,3])
542
543 #check equivalency of xi calc methods
544 @test isapprox(reshape(ξ,Tmaxplus1,K,K)[1:21,:,:],m_ξ)
545
546 γ = exp.(log_γ)
547 k_γ = sum(γ, dims=3); k_γ[nan_mask[:, :, :]] .= Inf #prevent NaNs in
548     ↪ dummy renorm
549 γ ./= k_γ

```

```

545
546     m_γ = exp.((m_log_α .+ m_log_β) .- normalizer)
547
548     #check equivalency of gamma calculations
549     @test isapprox(reshape(y,Tmaxplus1,K)[1:21,:,:],m_γ)
550
551     # "Testing initial and transition matrix calcs ... "
552
553     m_new_A = sum(m_ξ[1:end-1,:,:], dims=1)[1,:,:]
554     m_new_A ./= sum(m_new_A, dims=2)
555
556     new_A = zeros(K,K)
557     for i=1:K, j=1:K
558         Σotξ_vec = zeros(0)
559         Σotγ_vec = zeros(0)
560         for o in 1:0
561             Σotξ_vec[o] = sum(ξ[1:obs_lengths[o]-1,o,i,j])
562             Σotγ_vec[o] = sum(y[1:obs_lengths[o]-1,o,i])
563         end
564         new_A[i,j] = sum(Σotξ_vec) / sum(Σotγ_vec)
565     end
566     new_A ./= sum(new_A, dims=[2]) #dummy renorm
567
568     #check equivalency of transition matrix calculations
569     @test isapprox(m_new_A,new_A)
570
571     m_new_a = exp.((m_log_α[1,:] + m_log_β[1,:]) .- normalizer)
572     m_new_a ./= sum(m_new_a)
573
574     new_a = (sum(y[1,:,:], dims=1)./sum(sum(y[1,:,:], dims=1)))[1,:]
575     new_a ./= sum(new_a) #dummy renorm
576
577     @test isapprox(m_new_a,new_a)
578
579     # "Testing distribution calcs ... "
580
581     F=Univariate
582     m_D = Distribution{F}[]
583     for (i, d) in enumerate(hmm.B)
584         # Super hacky ...
585         # https://github.com/JuliaStats/Distributions.jl/issues/809
586         push!(m_D, fit_mle(eval(typeof(d).name.name),
587             permutedims(obs[1:21]), m_γ[:,i]))

```

```

587     end
588
589     obs_mask = .!nan_mask
590     obs_collection = obs[obs_mask[:, :]]
591
592     B = Distribution{F}[]
593     @inbounds for (i, d) in enumerate(hmm.B)
594         # Super hacky ...
595         # https://github.com/JuliaStats/Distributions.jl/issues/809
596         γ_d = γ[:, :, i]
597         push!(B, fit_mle(eval(typeof(d).name.name), obs_collection,
598                           → γ_d[obs_mask[:, :]]))
599         #slowest call by orders of magnitude
600     end
601
602     #test maximization equivalency
603     for (d, dist) in enumerate(B)
604         @test isapprox(dist.support, m_D[d].support)
605         @test isapprox(dist.p, m_D[d].p)
606     end
607
608     # "Testing mle_step ... "
609
610     #verify that above methods independently produce equivalent output,
611     → and that this is true of multiple identical obs, but not true of
612     → different obs sets
613     mouchet_hmm = mouchet_mle_step(hmm, obs[1:21])
614
615     new_hmm = BioBackgroundModels.bw_step(hmm, obs, obs_lengths)
616
617     dblobs = zeros(UInt8, 22, 2)
618     dblobs[1:21, 1] = [4, 3, 3, 2, 3, 2, 1, 1, 2, 3, 3, 3, 4, 4, 2, 3, 2, 3, 4, 3, 2]
619     dblobs[1:21, 2] = [4, 3, 3, 2, 3, 2, 1, 1, 2, 3, 3, 3, 4, 4, 2, 3, 2, 3, 4, 3, 2]
620     dblobs_lengths=[21, 21]
621     dbl_hmm = BioBackgroundModels.bw_step(hmm, dblobs, dblobs_lengths)
622
623     otherobs = dblobs
624     otherobs[1:21, 2] = [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1]
625     other_hmm = BioBackgroundModels.bw_step(hmm, otherobs,
626                                             → dblobs_lengths)

for n in fieldnames(typeof(new_hmm[1]))
    if n == :B

```

```

626     for (d, dist) in enumerate(new_hmm[1].B)
627         @test dist.support == mouchet_hmm[1].B[d].support
628         @test isapprox(dist.p,mouchet_hmm[1].B[d].p)
629         @test dist.support == dbl_hmm[1].B[d].support
630         @test isapprox(dist.p,dbl_hmm[1].B[d].p)
631         @test dist.support == other_hmm[1].B[d].support
632         @test !isapprox(dist.p, other_hmm[1].B[d].p)
633     end
634 elseif n == :partition
635     @test getfield(new_hmm[1],n) == getfield(mouchet_hmm[1],n)
636     @test getfield(new_hmm[1],n) == getfield(dbl_hmm[1],n)
637     @test getfield(new_hmm[1],n) == getfield(other_hmm[1],n)
638 else
639     @test isapprox(getfield(new_hmm[1],n),
640                   getfield(mouchet_hmm[1],n))
641     @test isapprox(getfield(new_hmm[1],n),
642                   getfield(dbl_hmm[1],n))
643     @test !isapprox(getfield(new_hmm[1],n),
644                   getfield(other_hmm[1],n))
645 end
646
647 @test new_hmm[2] == mouchet_hmm[2] ≠ dbl_hmm[2] ≠ other_hmm[2]
648
649 # "Testing fit_mle! ... "
650
651 # "test fit_mle! function"
652 input_hmms= RemoteChannel(()→Channel{Tuple}(1))
653 output_hmms = RemoteChannel(()→Channel{Tuple}(30))
654 chainid=Chain_ID("Test",2,0,1)
655 put!(input_hmms, (chainid, 2, hmm, 0.0, transpose(obs)))
656 BioBackgroundModels.EM_converge!(input_hmms, output_hmms, 1;
657   ↳ EM_func=BioBackgroundModels.bw_step, max_iterates=4,
658   ↳ verbose=true)
659 wait(output_hmms)
660 workerid, jobid, iterate, hmm3, log_p, epsilon, converged =
661   ↳ take!(output_hmms)
662 @test jobid == chainid
663 @test iterate == 3
664 @test assert_hmm(hmm3.a, hmm3.A, hmm3.B)
665 @test size(hmm3) == size(hmm) == (2,4)
666 @test log_p < 1
667 @test log_p == obs_lh_given_hmm(transpose(obs),hmm, linear=false)

```

```

663 @test converged == false
664 wait(output_hmms)
665 workerid, jobid, iterate, hmm4, log_p, epsilon, converged =
666   ↳ take!(output_hmms)
667 @test jobid == chainid
668 @test iterate == 4
669 @test assert_hmm(hmm4.a, hmm4.A, hmm4.B)
670 @test size(hmm4) == size(hmm) == (2,4)
671 @test log_p < 1
672 @test log_p == obs_lh_given_hmm(transpose(obs),hmm3,linear=false)
673 @test converged == false
674
675 # "Test convergence.. "
676 obs=zeros(UInt8, 101, 2)
677 for i in 1:size(obs)[2]
678     obs[1:100,i]=rand(1:4,100)
679 end
680 input_hmms= RemoteChannel(()→Channel{Tuple}(1))
681 output_hmms = RemoteChannel(()→Channel{Tuple}(30))
682 put!(input_hmms, (chainid, 2, hmm, 0.0, transpose(obs)))
683 BioBackgroundModels.EM_converge!(input_hmms, output_hmms, 1;
684   ↳ EM_func=BioBackgroundModels.bw_step, delta_thresh=.05,
685   ↳ max_iterates=100, verbose=true)
686 wait(output_hmms)
687 workerid, jobid, iterate, hmm4, log_p, epsilon, converged =
688   ↳ take!(output_hmms)
689 while isready(output_hmms)
690     workerid, jobid, iterate, hmm4, log_p, epsilon, converged =
691       ↳ take!(output_hmms)
692 end
693 @test converged==1
694
695 #test t_add_categorical_counts DimensionMismatch
696 @test_throws DimensionMismatch t_add_categorical_counts!(zeros(4),
697   ↳ zeros(UInt8,2,3), zeros(2,2))
698
699 end
700
701 @testset "EM/churbanov.jl Baum-Welch equivalency and functions" begin
702     # "Setting up for MLE function tests.. "
703     A = fill((1/6),6,6)

```

```

698     B = [Categorical(ones(4)/4),
   ↳ Categorical([.7,.05,.15,.1]),Categorical([.15,.35,.4,.1]),
   ↳ Categorical([.6,.15,.15,.1]),Categorical([.1,.4,.4,.1]),
   ↳ Categorical([.2,.2,.3,.3])]
699     hmm = BHMM(A, B)
700     log_A = log.(hmm.A)
701
702     obs = zeros(Int64,1,250)
703     obs[1:249] = rand(1:4,249)
704     obs_lengths=[249]
705     # "Testing mle_step ... "
706
707     #verify that above methods independently produce equivalent output,
   ↳ and that this is true of multiple identical obs, but not true of
   ↳ different obs sets
708     mouchet_hmm = mouchet_mle_step(hmm, obs[1:249])
709
710     new_hmm = linear_step(hmm, obs, obs_lengths)
711
712     ms_sng = BioBackgroundModels.bw_step(hmm, Array(transpose(obs)),
   ↳ obs_lengths)
713
714     dbllobs = zeros(Int64, 2,250)
715     dbllobs[1,1:249] = obs[1:249]
716     dbllobs[2,1:249] = obs[1:249]
717     dbllobs_lengths=[249,249]
718     dbl_hmm = linear_step(hmm, dbllobs, dbllobs_lengths)
719
720
721     ms_dbl =BioBackgroundModels.bw_step(hmm,
   ↳ Array(transpose(dbllobs)),dblobs_lengths)
722
723     otherobs = deepcopy(dbllobs)
724     otherobs[2,1:249] = rand(1:4,249)
725     if otherobs[1,1]==otherobs[2,1]
726         otherobs[1,1]=1&&(otherobs[2,1]==2)
727         otherobs[1,1]=2&&(otherobs[2,1]==3)
728         otherobs[1,1]=3&&(otherobs[2,1]==4)
729         otherobs[1,1]=4&&(otherobs[2,1]==1)
730     end
731
732     other_hmm = linear_step(hmm, otherobs, dbllobs_lengths)
733

```

```

734     ms_hmm = BioBackgroundModels.bw_step(hmm,
735         ↳ Array(transpose(otherobs)),dblobs_lengths)
736
737     for n in fieldnames(typeof(new_hmm[1]))
738         if n == :B
739             for (d, dist) in enumerate(new_hmm[1].B)
740                 @test dist.support==mouchet_hmm[1].B[d].support
741                 @test isapprox(dist.p,mouchet_hmm[1].B[d].p)
742                 @test dist.support==ms_sng[1].B[d].support
743                 @test isapprox(dist.p,ms_sng[1].B[d].p)
744                 @test dist.support==dbl_hmm[1].B[d].support
745                 @test isapprox(dist.p,dbl_hmm[1].B[d].p)
746                 @test dist.support==ms_dbl[1].B[d].support
747                 @test isapprox(dist.p,ms_dbl[1].B[d].p)
748                 @test dist.support==other_hmm[1].B[d].support
749                 @test ms_hmm[1].B[d].support==other_hmm[1].B[d].support
750                 @test !isapprox(dist.p, other_hmm[1].B[d].p)
751                 @test isapprox(ms_hmm[1].B[d].p, other_hmm[1].B[d].p)
752
753         end
754     elseif n == :partition
755         @test getfield(new_hmm[1],n) == getfield(mouchet_hmm[1],n)
756         @test getfield(new_hmm[1],n) == getfield(dbl_hmm[1],n)
757         @test getfield(new_hmm[1],n) == getfield(other_hmm[1],n)
758     else
759         @test isapprox(getfield(new_hmm[1],n),
760             ↳ getfield(mouchet_hmm[1],n))
761         @test isapprox(getfield(new_hmm[1],n), getfield(ms_sng[1],n))
762
763         @test isapprox(getfield(new_hmm[1],n),
764             ↳ getfield(dbl_hmm[1],n))
765         @test isapprox(getfield(new_hmm[1],n), getfield(ms_dbl[1],n))
766
767         @test !isapprox(getfield(new_hmm[1],n),
768             ↳ getfield(other_hmm[1],n))
769         @test isapprox(getfield(ms_hmm[1],n),
770             ↳ getfield(other_hmm[1],n))
771
772     end
773
774     @test ms_sng[2] == new_hmm[2] == mouchet_hmm[2] ≠ dbl_hmm[2] ==
775         ↳ ms_dbl[2] ≠ other_hmm[2] == ms_hmm[2]

```

```

771
772     # "Testing fit_mle! ... "
773
774     #test fit_mle! function
775     input_hmms= RemoteChannel(()→Channel{Tuple}(1))
776     output_hmms = RemoteChannel(()→Channel{Tuple}(30))
777     chainid=Chain_ID("Test",6,0,1)
778     put!(input_hmms, (chainid, 2, hmm, 0.0, obs))
779     BioBackgroundModels.EM_converge!(input_hmms, output_hmms, 1;
780         → max_iterates=4, verbose=true)
781     wait(output_hmms)
782     workerid, jobid, iterate, hmm3, log_p, delta, converged =
783         → take!(output_hmms)
784     @test jobid == chainid
785     @test iterate == 3
786     @test BioBackgroundModels.assert_hmm(hmm3.a, hmm3.A, hmm3.B)
787     @test size(hmm3) == size(hmm) == (6,4)
788     @test log_p < 1
789     @test log_p == linear_likelihood(obs, hmm)
790     @test converged == false
791     wait(output_hmms)
792     workerid, jobid, iterate, hmm4, log_p, delta, converged =
793         → take!(output_hmms)
794     @test jobid == chainid
795     @test iterate == 4
796     @test BioBackgroundModels.assert_hmm(hmm4.a, hmm4.A, hmm4.B)
797     @test size(hmm4) == size(hmm) == (6,4)
798     @test log_p < 1
799     @test log_p == linear_likelihood(obs, hmm3)
800     @test converged == false
801
802     # "Test convergence.. "
803     obs=zeros(Int64, 4, 1001)
804     for o in 1:size(obs)[1]
805         obsl=rand(100:1000)
806         obs[o,1:obsl]=rand(1:4,obsl)
807     end
808     input_hmms= RemoteChannel(()→Channel{Tuple}(1))
809     output_hmms = RemoteChannel(()→Channel{Tuple}(Inf))
810     put!(input_hmms, (chainid, 2, hmm, 0.0, obs))
811     BioBackgroundModels.EM_converge!(input_hmms, output_hmms, 1;
812         → delta_thresh=.05, max_iterates=100, verbose=true)
813     wait(output_hmms)

```

```

810     workerid, jobid, iterate, hmm4, log_p, delta, converged =
811         → take!(output_hmms)
812     while isready(output_hmms)
813         workerid, jobid, iterate, hmm4, log_p, delta, converged =
814             → take!(output_hmms)
815     end
816     @test converged==1
817 end
818
819 @testset "API/EM_master.jl and reports.jl API tests" begin
820     #CONSTANTS FOR BHMM LEARNING
821     replicates = 4 #repeat optimisation from this many seperately
822         → initialised samples from the prior
823     Ks = [1,2,4,6] #mosaic class #s to test
824     order_nos = [0,1,2] #DNA kmer order #s to test
825     sample_set_length=100
826     min_sample_window=5
827     max_sample_window=25
828     perigenic_pad=250
829
830     wkpool=addprocs(2)
831     @everywhere using BioBackgroundModels
832
833     channels = setup_sample_jobs(genome, index, gff, sample_set_length,
834         → min_sample_window, max_sample_window, perigenic_pad;
835         → deterministic=true)
836     sample_record_dfs=execute_sample_jobs(channels, wkpool)
837     training_sets, test_sets = split_obs_sets(sample_record_dfs)
838
839     rmprocs(wkpool)
840
841
842     job_ids=Vector{Chain_ID}()
843     for (obs_id, obs) in training_sets, K in Ks, order in order_nos, rep
844         → in 1:replicates
845         push!(job_ids, Chain_ID(obs_id, K, order, rep))
846     end
847
848     no_input_hmms, chains, input_hmms, output_hmms =
849         → setup_EM_jobs!(job_ids, training_sets)
850     @test no_input_hmms ==
851         → replicates*length(Ks)*length(order_nos)*length(training_sets)
852
853
854

```

```

845 while isready(input_hmms)
846     jobid, start_iterate, hmm, last_norm, observations =
847         ↵ take!(input_hmms)
848     @test last_norm == linear_likelihood(observations, hmm)
849     obs_lengths = [findfirst(iszero, observations[o,:])-1 for o in
850         ↵ 1:size(observations)[1]]
851     #make sure input HMMs are valid and try to mle_step them and
852         ↵ ensure their 1-step children are valid
853     @test assert_hmm(hmm.a, hmm.A, hmm.B)
854     new_hmm, prob = linear_step(hmm, observations, obs_lengths)
855     @test assert_hmm(hmm.a, new_hmm.A, new_hmm.B)
856     @test prob < 0
857 end

858 genome_reader = open(FASTA.Reader, genome, index=index)

859     ↵ seq=LongSequence{DNAAlphabet{2}}(FASTA.sequence(genome_reader["CM002885.2.1"]))
860 seqdict = Dict("test"⇒[seq for i in 1:3])
861 seqdict["blacklist"] = seqdict["test"]

862 job_ids=Vector{Chain_ID}()
863 push!(job_ids, Chain_ID("blacklist", 1, 0, 1))
864 push!(job_ids, Chain_ID("blacklist", 1, 0, 2))
865 Ks=[1,2]; order_nos=[0,1]
866 for K in Ks, order in order_nos, replicate in 1:2
867     push!(job_ids, Chain_ID("test", K, order, replicate))
868 end

869 wkpool=addprocs(2, topology=:master_worker)
870 @everywhere using BioBackgroundModels

871 load_dict=Dict{Int64,LoadConfig}()
872 for (n,wk) in enumerate(wkpool)
873     n==1 &&
874         ↵ (load_dict[wk]=LoadConfig(1:2,0:1,blacklist=[Chain_ID("blacklist",1,0,1)])
875     n==2 && (load_dict[wk]=LoadConfig(1:2,0:1))
876 end
877 #test work resumption, load configs

878 em_jobset=setup_EM_jobs!(job_ids, seqdict, delta_thresh=10000.)
879 execute_EM_jobs!(wkpool, em_jobset..., "testchains",
880     ↵ delta_thresh=10000., verbose=true, load_dict=load_dict)

```

```

882     for (chain_id,chain) in em_jobset[2]
883         @test chain[end].converged == true
884         @test chain[end].delta ≤ 10000
885         @test typeof(chain[end].hmm)==BHMM{Float64}
886         @test 1 ≤ chain[end].iterate ≤ 5000
887     end
888
889     wkpool=addprocs(2, topology=:master_worker)
890     @everywhere using BioBackgroundModels
891
892     em_jobset=setup_EM_jobs!(job_ids, seqdict,
893         → chains=deserialize("testchains"), delta_thresh=.1)
894     execute_EM_jobs!(wkpool, em_jobset..., "testchains", delta_thresh=.1,
895         → verbose=true)
896
897     for (chain_id,chain) in em_jobset[2]
898         @test chain[end].converged == true
899         @test chain[end].delta ≤ .1
900         @test typeof(chain[end].hmm)==BHMM{Float64}
901         @test 1 ≤ chain[end].iterate ≤ 5000
902     end
903
904     #test for already converged warning
905     wkpool=addprocs(2, topology=:master_worker)
906     @everywhere using BioBackgroundModels
907     @test_throws ArgumentError execute_EM_jobs!(wkpool, em_jobset..., "testchains", delta_thresh=.01, verbose=true)
908
909     rm("testchains")
910
911     @test_throws ArgumentError
912         → generate_reports(Dict{Chain_ID,Vector{EM_step}}(),seqdict)
913     @test_throws ArgumentError
914         → generate_reports(em_jobset[2],Dict{String,Vector{LongSequence{DNAAlphabet{2}}}})
915
916     @test_throws ArgumentError
917         → BioBackgroundModels.report_chains(em_jobset[2],Dict{String,Vector{LongSequence{DNAAlphabet{2}}}})
918
919     report_folders=generate_reports(em_jobset[2],seqdict)
920     folder=report_folders["test"]
921     @test "test"==folder.partition_id
922     show(folder.partition_report)
923     show(folder.replicate_report)

```

```

919     for id in folder.partition_report.best_repset
920         show(folder.chain_reports[id])
921     end
922 end
923
924 rm(genome)
925 rm(index)
926 rm(gff)
927 rm(posfasta)

```

---

### 17.5.27 /test/synthetic\_sequence\_gen.jl

```

1 function print_synthetic_fasta(path::String,line_length::Integer=80)
2     header=>CM002885.2.1 BioBackgroundModels Synthetic chromosome
3     ↪   for tests\n
4     write(path, header,
5         ↪   format_lines(generate_synthetic_seq(),line_length))
6 end
7
8 function print_synthetic_index(path::String)
9     write(path,
10        ↪   "CM002885.2.1          1000          5          80          81\n")
11 end
12
13
14 function print_synthetic_position(path::String)
15     write(path, ">1 start 501 end 641 smt_pos 111 smt_value 205.11821
16     ↪   fuzziness_score
17     ↪   43.53698\n",generate_synthetic_seq()[501:641],"\n")
18 end
19
20
21 function print_synthetic_gff(path::String)
22     gff_txt = ##gff-version 3
23     ##sequence-region 1 1 1000
24     #!genome-build BGHMM Synthetic
25     #!genome-version Synth1.0
26     #!genome-date 2019-08
27     #!genome-build-accession NCBI:FAKE
28     #!genebuild-last-updated 2019-08
29     1      Ensembl      chromosome      1      1000      .
30     .      .
31     .      .
32     ####

```

```

24 1      ensembl_havana    gene      501      1000      .      -      .
25 1      ensembl_havana    mRNA      501      1000      .      -      .
26 1      ensembl_havana    three_prime_UTR 501      510      .      .
27 1      ensembl_havana    exon      501      570      .      +      .
28 1      ensembl_havana    CDS       511      570      .      +      0
29 1      ensembl_havana    exon      601      660      .      -      .
30 1      ensembl_havana    CDS       601      660      .      -      1
31 1      ensembl_havana    exon      701      760      .      -      .
32 1      ensembl_havana    CDS       701      760      .      -      0
33 1      ensembl_havana    exon      801      860      .      -      .
34 1      ensembl_havana    CDS       801      860      .      -      0
35 1      ensembl_havana    exon      901      975      .      -      .
36 1      ensembl_havana    CDS       901      960      .      -      1
37 1      ensembl_havana    five_prime_UTR 961      975      .      -      -
38 ###\n"
39     write(path,gff_txt)
40 end
41
42 function generate_synthetic_seq(gene_start::Integer=501,
43     ↪ UTR3L::Integer=10, exon_length::Integer=60,
44     ↪ intron_length::Integer=40, no_exons::Integer=5, UTR5L::Integer=25,
45     ↪ OAL::Integer=1000,
46     ↪ (iseq,pseq,eseq)::Tuple{String,String,String}=( "AT", "CG", "CAT");
47     ↪ verbose::Bool=false)
48     @assert length(iseq) == 2
49     @assert length(pseq) == 2
50     @assert length(eseq) == 3
51     @assert mod(gene_start-1,2) == 0
52     @assert mod(exon_length,3) == 0
53     @assert mod(intron_length,2) == 0
54     gene_length=gene_start+UTR3L+(no_exons*exon_length)+((no_exons-1)*intron_length)
55     @assert gene_length ≤ OAL
56     seq=""
57     for i in 1:((gene_start-1) / 2)
58         seq *= iseq
59     end
60     verbose && @info "Intergenic length $(length(seq))"
61     for i in 1:UTR3L / 2
62         seq *= pseq
63     end
64     verbose && @info "Added 3'UTR ... $(length(seq))"
```

```

60      for ex in 1:no_exons-1
61          for i in 1:(exon_length/3)
62              seq*=eseq
63          end
64          ex == 1 ? ilength = 30 : ilength = 40
65          for i in 1:(ilength/2)
66              seq*=pseq
67          end
68      end
69      for i in 1:(exon_length/3)
70          seq*=eseq
71      end
72      verbose && @info "Added exons and introns ... $(length(seq))"
73      for i in 1:UTR5L / 2
74          seq *= pseq
75      end
76      verbose && @info "Added 5'UTR ... $(length(seq))"
77
78      for i in 1:((OAL-length(seq))/2)
79          seq*=iseq
80      end
81
82      verbose && @info "Generated synthetic genome sequence of length
83      → $(length(seq))"
84
85      return seq
86  end
87
88  function format_lines(seq::String, line_length::Integer)
89      a=join((SubString(seq,i,min(i+line_length-1,length(seq)))) for
90      → i=1:line_length:length(seq)), '\n')
91  return(a)
92 end

```

---

## 17.6 BioMotifInference

Github repository: <https://github.com/mmattocks/BioMotifInference.jl>

### 17.6.1 /src/BioMotifInference.jl

---

```

1 module BioMotifInference
2     using BioBackgroundModels, BioSequences, Distributed, Distributions,
2     ↳ Serialization, UnicodePlots
3     import DataFrames:DataFrame
4     import ProgressMeter: AbstractProgress, Progress, @showprogress,
4     ↳ next!, move_cursor_up_while_clearing_lines, printover,
4     ↳ durationstring
5     import Printf: @sprintf
6     import StatsFuns: logaddexp, logsumexp, logsubexp
7     import Random: rand, seed!, shuffle!
8     import Distances: euclidean
9     import Measurements: measurement
10
11    #CONSTANTS AND PERMUTE FUNCTION ARGUMENT DEFAULTS GIVING RISE TO
11    ↳ IMPLEMENTATION-SPECIFIC SAMPLING EFFECTS
12    global MOTIF_EXPECT=1. #motif expectation- this value to be divided
12    ↳ by obs lengths to obtain penalty factor for scoring
13
14    global REVCOMP=true #calculate scores on both strands?
15
16    global TUNING_MEMORY=20 #coefficient multiplied by Permute_Instruct
16    ↳ function call limit to give total number of calls remembered by
16    ↳ tuner per function
17    global CONVERGENCE_MEMORY=500 #number of iterates to display for
17    ↳ convergence interval history
18
19    global SRC_PERM_FREQ=.5 #frequency with which random_decorrelate will
19    ↳ permute a source
20
21    global PWM_SHIFT_DIST=Weibull(.5,.1) #distribution of weight matrix
21    ↳ permutation magnitudes
22    global PWM_SHIFT_FREQ=.2 #proportion of positions in source to
22    ↳ permute weight matrix
23    global PWM_LENGTHPERM_FREQ=.2 #proportion of sources to permute
23    ↳ length
24    global LENGTHPERM_RANGE=1:3
25
26    global PRIOR_WT=3. #estimate prior dirichlets from product of this
26    ↳ constant and sample "mle" wm
27    global PRIOR_LENGTH_MASS=.8
28

```

```

29 global EROSION_INFO_THRESH=1.
30
31 global CONSOLIDATE_THRESH=.035
32
33 include("IPM/ICA_PWM_Model.jl")
34 export Model_Record
35 export ICA_PWM_Model
36 include("IPM/IPM_likelihood.jl")
37 include("IPM/IPM_prior_utilities.jl")
38 assemble_source_priors
39 include("ensemble/IPM_E ensemble.jl")
40 export IPM_E ensemble, assemble_IPMs
41 include("permutation/permute_utilities.jl")
42 include("permutation/orthogonality_helper.jl")
43 include("permutation/permute_functions.jl")
44 export full_perm_funcvec
45 include("permutation/permute_control.jl")
46 export Permute_Instruct
47 include("permutation/Permute_Tuner.jl")
48 include("ensemble/ensemble_utilities.jl")
49 export ensemble_history, reset_ensemble!, move_ensemble!,
   ↳ copy_ensemble!, rewind_ensemble, complete_evidence, get_model,
   ↳ show_models, reestimate_ensemble!
50 include("utilities/model_display.jl")
51 include("utilities/worker_diagnostics.jl")
52 include("utilities/ns_progressmeter.jl")
53 include("utilities/synthetic_genome.jl")
54 include("utilities/worker_sequencer.jl")
55 export synthetic_sample
56 include("nested_sampler/nested_step.jl")
57 include("nested_sampler/converge_ensemble.jl")
58 export converge_ensemble!
59
60 end # module

```

---

### 17.6.2 /src/IPM/ICA\_PWM\_Model.jl

---

```

1 struct Model_Record #record struct to associate a log_Li with a saved,
   ↳ calculated model
2     path::String
3     log_Li::AbstractFloat
4 end

```

```

5
6 struct ICA_PWM_Model #Independent component analysis position weight
  ↳ matrix model
7   name::String #designator for saving model to posterior
8   origin::String #functions instantiating IPMs should give an
    ↳ informative desc of the means by which the model was generated
9   sources::Vector{Tuple{<:AbstractMatrix{<:AbstractFloat},<:Integer}}}
  ↳ #vector of PWM signal sources (LOG PROBABILITY!!!) tupled with an
  ↳ index denoting the position of the first PWM base on the prior
  ↳ matrix- allows us to permute length and redraw from the
  ↳ appropriate prior position
10  source_length_limits::UnitRange{<:Integer} #min/max source lengths
    ↳ for init and permutation
11  mix_matrix::BitMatrix # obs x sources bool matrix
12  log_Li::AbstractFloat
13  permute_blacklist::Vector{Function} #blacklist of functions that
    ↳ ought not be used to permute this model (eg. because to do so
    ↳ would not generate a different model for IPMs produced from
    ↳ fitting the mix matrix)
14  function ICA_PWM_Model(name, origin, sources, source_length_limits,
  ↳ mix_matrix, log_Li, permute_blacklist=Vector{Function}())
15    verify_srcs(name, origin, sources)
16    new(name, origin, sources, source_length_limits, mix_matrix,
  ↳ log_Li, permute_blacklist)
17  end
18 end
19
20 #ICA_PWM_Model FUNCTIONS
21 ICA_PWM_Model(name::String,
  ↳ source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat}},<:AbstractVector{<:AbstractFloat}}},
  ↳ mix_prior::Tuple{BitMatrix,<:AbstractFloat},
  ↳ bg_scores::AbstractArray{<:AbstractFloat},
  ↳ observations::AbstractArray{<:Integer},
  ↳ source_length_limits::UnitRange{<:Integer}) = init_IPM(name,
  ↳ source_priors,mix_prior,bg_scores,observations,source_length_limits)
22
23 function verify_srcs(name, ori,
  ↳ sources::Vector{<:Tuple{<:AbstractMatrix{<:AbstractFloat},<:Integer}}})
24   for (n,(pwm,pi)) in enumerate(sources)
25     !all(isprobvec(exp.(pwm[pos,:])) for pos in 1:size(pwm,1)) &&
      ↳ throw(DomainError("Bad probvec in model $name, source $n,
      ↳ origin $(ori):$(exp.(pwm)))))
26 end

```



```

54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
length_dist=DiscreteNonParametric(
    [source_length_limits ... ],
    → [PRIOR_LENGTH_MASS,ones(length(source_length))
    → PWM_length=rand(length_dist)
elseif source_length_limits[1] <
    → length(prior) <
    → source_length_limits[end]
length_dist=DiscreteNonParametric(
    [source_length_limits ... ],
    [
    → (ones(length(prior))-source_length_limit)
    → (ones(source_length_limits[end]-length(prior)))
    ]
)
PWM_length=rand(length_dist)
else
    PWM_length=rand(source_length_limits)
end

if PWM_length>length(prior)
    → prior_coord=rand(-(PWM_length-length(prior)):(length(prior)-1))
else
    → prior_coord=rand(1:length(prior)-PWM_length+1)
end

PWM = zeros(PWM_length,4)

curr_pos=prior_coord
for pos in 1:PWM_length
    curr_pos < 1 || curr_pos >
    → length(prior) ?
    → dirichlet=Dirichlet(ones(4)/4) :
    → dirichlet=prior[curr_pos]
    PWM[pos, :] = rand(dirichlet)
end

```

```

85             push!(srcvec, (log.(PWM), prior_coord))
86             #push the source PWM to the source
87             # vector with the prior coord idx to
88             # allow drawing from the appropriate
89             # prior dirichlets on permuting source
90             # length
91         elseif prior=false
92             PWM_length=rand(source_length_limits)
93             PWM=zeros(PWM_length,4)
94             dirichlet=Dirichlet(ones(4)/4)
95             for pos in 1:PWM_length
96                 PWM[pos,:]=rand(dirichlet)
97             end
98             push!(srcvec, (log.(PWM), 1))
99         else
100             throw(ArgumentError("Bad prior supplied
101                   for ICA_PWM_Model!"))
102         end
103     end
104     return srcvec
105 end
106
107 function
108     → init_mix_matrix(mix_prior::Tuple{BitMatrix,<:AbstractFloat},
109     → no_observations::Integer, no_sources::Integer)
110     inform,uninform=mix_prior
111     if size(inform,2) > 0
112         @assert size(inform,1)=no_observations &&
113             → size(inform,2)≤no_sources "Bad informative
114             → mix prior dimensions!"
115     end
116     @assert 0.0 ≤ uninform ≤1.0 "Uninformative mix
117             → prior not between 0.0 and 1.0!"
118     mix_matrix = falses(no_observations, no_sources)
119     if size(inform,2)>0
120         mix_matrix[:,1:size(inform,2)]=inform
121     end
122     for index in
123         → CartesianIndices((1:no_observations,size(inform,2)+1:no_sources))
124         rand() ≤ uninform && (mix_matrix[index] = true)
125     end
126     return mix_matrix
127 end

```

```

116
117 function Base.show(io::IO, m::ICA_PWM_Model;
118   ↪ nsrc::Integer=length(m.sources), progress=false)
119   nsrc == 0 && (nsrc==length(m.sources))
120   nsrc>length(m.sources) && (nsrc==length(m.sources))
121   nsrc==length(m.sources) ? (srcstr="All") : (srcstr="Top $nsrc")
122
123   printidxs,printsrcs,printfreqs=sort_sources(m,nsrc)
124
125   printstyled(io, "ICA PWM Model $(m.name) w/ logLi $(m.log_Li)\n",
126   ↪ bold=true)
127   println(io, srcstr*" sources:")
128   for src in 1:nsrc
129     print(io, "$(printidxs[src]), $(printfreqs[src]*100)%: ")
130     pwmstr_to_io(io, printsrcs[src])
131     println(io)
132   end
133   println(m.origin)
134
135   progress && return(nsrc+4)
136 end
137
138 function sort_sources(m, nsrc)
139   printidxs=Vector{Integer}()
140   printsrcs=Vector{Matrix{Float64}}()
141   printfreqs=Vector{Float64}()
142
143   freqs=vec(sum(m.mix_matrix,dims=1)); total=size(m.mix_matrix,1)
144   sortfreqs=sort(freqs,rev=true); sortidxs=sortperm(freqs,rev=true)
145   for srcidx in 1:nsrc
146     push!(printidxs, sortidxs[srcidx])
147     push!(printsrcs, m.sources[sortidxs[srcidx]][1])
148     push!(printfreqs, sortfreqs[srcidx]/total)
149   end
150   return printidxs,printsrcs,printfreqs
151 end

```

---

### 17.6.3 /src/IPM/IPM\_likelihood.jl

---

```

1 #LIKELIHOOD SCORING FUNCS
2 function
3     ↳ IPM_likelihood(sources::AbstractVector{<:Tuple{<:AbstractMatrix{<:AbstractFloat},,
4         ↳ observations::AbstractMatrix{<:Integer},,
5         ↳ obs_lengths::AbstractVector{<:Integer},,
6         ↳ bg_scores::AbstractArray{<:AbstractFloat}, mix::BitMatrix,
7         ↳ revcomp::Bool=REVCOMP, returncache::Bool=false,
8         ↳ cache::AbstractVector{<:AbstractFloat}=zeros(0),
9         ↳ clean::AbstractVector{<:Bool}=Vector(falses(size(observations)[2])))
10    ↳ source_wmls=[size(source[1])[1] for source in sources]
11    ↳ 0 = size(bg_scores)[2]
12    ↳ source_stops=[obs_l-wml+1 for wml in source_wmls, obs_l in obs_lengths]
13        ↳ #stop scanning th source across the observation as the source
14        ↳ reaches the end
15    ↳ L=maximum(obs_lengths)+1
16
17    ↳ obs_src_idxs=mix_pull_idxs(mix) #get vectors of sources emitting in
18        ↳ each obs
19
20    ↳ revcomp ? (srcs=[cat(source[1],revcomp_pwm(source[1]),dims=3) for
21        ↳ source in sources]; motif_expectations = [(MOTIF_EXPECT/2)/obs_l)
22        ↳ for obs_l in obs_lengths; mat_dim=2) : (srcs=[source[1] for
23        ↳ source in sources]; ; motif_expectations = [(MOTIF_EXPECT/obs_l)
24        ↳ for obs_l in obs_lengths; mat_dim=1) #setup appropriate reverse
25        ↳ complemented sources if necessary and set
26        ↳ log_motif_expectation-nMica has 0.5 per obs for including the
27        ↳ reverse complement, 1 otherwise
28
29    ↳ lme_vec=zeros(length(sources))
30
31    ↳ obs_lhs=Vector{Vector{Float64}}() #setup likelihood vecs for threaded
32        ↳ operation-reassembled on return
33    ↳ nt=Threads.nthreads()
34    ↳ for t in 1:nt-1
35        ↳ push!(obs_lhs,zeros(Int(floor(0/nt))))
36    ↳ end
37    ↳ push!(obs_lhs, zeros(Int(floor(0/nt)+(0%nt))))
38
39    ↳ Threads.@threads for t in 1:nt
40        ↳ revcomp && (weavevec=zeros(3))

```

```

23     revcomp ? (score_mat=zeros(maximum(source_stops),2)) :
24         ↵ (score_mat=zeros(maximum(source_stops)))
25     opt = floor(0/nt) #obs per thread
26     score_matrices=Vector{typeof(score_mat)}(undef, length(sources))
27         ↵ #preallocate
28     osi_emitting=Vector{Int64}() #preallocate
29     lh_vec = zeros(L) #preallocated likelihood vector is one position
30         ↵ (0 initialiser) longer than the longest obs
31
32     for i in 1:Int(opt+(t==nt)*(0%nt))
33         o=Int(i+(t-1)*opt)
34         if clean[o]
35             obs_lhs[t][i]=cache[o]
36         else
37             obsl = obs_lengths[o]
38             oidxs=obs_src_idxs[o]
39             mixwmls=source_wmls[oidxs]
40
41             obs_cardinality = length(oidxs) #the more sources, the
42                 ↵ greater the cardinality_penalty
43             if obs_cardinality > 0
44                 revcomp ? score_sources_ds!(score_mat,
45                     ↵ score_matrices, view(srcts,oidxs),
46                     ↵ view(observations,:,:,o),
47                     ↵ view(source_stops,oidxs,o)) :
48                     score_sources_ss!(score_mat, score_matrices,
49                         ↵ view(srcts,oidxs), view(observations,:,:,o),
50                         ↵ view(source_stops,oidxs,o)) #get scores for
51                         ↵ this observation
52
53             lme_vec.=motif_expectations[o]
54             penalty_sum = sum(lme_vec[1:obs_cardinality])
55             penalty_sum > 1. && (penalty_sum=1.)
56             cardinality_penalty=log(1.0-penalty_sum)
57         else
58             cardinality_penalty=0.0
59         end
60
61     revcomp ? (obs_lhs[t][i]=weave_scores_ds!(weavevec,
62         ↵ lh_vec, obsl, view(bg_scores,:,:,o), score_matrices,
63         ↵ oidxs, mixwmls, log(motif_expectations[o]),
64         ↵ cardinality_penalty, osi_emitting)) :

```

```

53             (obs_lhs[t][i]=weave_scores_ss!(lh_vec, obsL,
54             ↳ view(bg_scores,:,:o), score_matrices, oidxs,
55             ↳ mixwmls, log(motif_expectations[o]),
56             ↳ cardinality_penalty, osi_emitting))
57
58         end
59     end
60
61     returncache ? (return lps([lps(obs_lhs[t]) for t in 1:nt]),
62     ↳ vcat(obs_lhs...)) : (return lps([lps(obs_lhs[t]) for t in 1:nt]))
63 end
64
65 @inline function mix_pull_idxs(A::AbstractArray{Bool})
66     n=count(A)
67     S=[Vector{Int64}() for o in 1:size(A,1)]
68     cnt=1
69     for (i,a) in pairs(A)
70         if a
71             push!(S[i[1]],i[2])
72             cnt+=1
73         end
74     end
75     return S
76 end
77
78 @inline function
79     ↳ revcomp_pwm(pwm::AbstractMatrix{<:AbstractFloat}) #in
80     ↳ order to find a motif on the reverse strand, we scan
81     ↳ the forward strand with the reverse complement of the
82     ↳ pwm, reordered 3' to 5', so that eg. an PWM for an
83     ↳ ATG motif would become one for a CAT motif
84     return pwm[end:-1:1,end:-1:1]
85 end
86
87 @inline function score_sources_ds!(score_mat,
88     ↳ score_matrices::AbstractVector{<:AbstractMatrix{<:AbstractFloat}},
89     ↳ sources, observation::AbstractVector{<:Integer},
90     ↳ source_stops)
91     for (s,source) in enumerate(sources)
92         for t in 1:source_stops[s]
93             for position in 1:size(source,1)

```

```

84             score_loc = t+position-1 #score_loc is
85             ← the position of the obs to be scored
86             ← by PWM
87             score_mat[t,1] +=
88             ← source[position,observation[score_loc],1]
89             ← #add the appropriate log PWM value
90             ← from the source to the score
91             score_mat[t,2] +=
92             ← source[position,observation[score_loc],2]
93             ← #add the appropriate log PWM value
94             ← from the source to the score
95
96         end
97     end
98
99     score_matrices[s]=score_mat[1:source_stops[s],:]
100    ← #copy score matrix to vector
101    score_mat.=0. #reset score matrix
102
103
104     @inline function score_sources_ss!(score_mat,
105         ← score_matrices::AbstractVector{<:AbstractArray{<:AbstractFloat}},)
106         ← sources, observation::AbstractVector{<:Integer},
107         ← source_stops)
108         for (s,source) in enumerate(sources)
109             for t in 1:source_stops[s]
110                 for position in 1:size(source,1)
111                     score_loc = t+position-1 #score_loc is
112                     ← the position of the obs to be scored
113                     ← by PWM
114                     score_mat[t] +=
115                     ← source[position,observation[score_loc]]
116                     ← #add the appropriate log PWM value
117                     ← from the source to the score
118
119                 end
120             end
121             score_matrices[s]=score_mat[1:source_stops[s]]
122             ← #copy score matrix to vector
123             score_mat.=0. #reset score matrix
124
125         end
126     end
127
128

```



```

131             weavevec *= score, lps(from_score,
132             ↳ f_emit_score, log_motif_expectation),
133             ↳ lps(from_score, r_emit_score,
134             ↳ log_motif_expectation)

135         score=logsumexp(weavevec)
136     end
137     lh_vec[i] = score
138   end
139
140   @inline function weave_scores_ss!(lh_vec, obsl::Integer,
141     ↳ bg_scores::SubArray,
142     ↳ score_mat::AbstractVector{<:AbstractVector{<:AbstractFloat}},
143     ↳ obs_source_indices::AbstractVector{<:Integer},
144     ↳ source_wmls::AbstractVector{<:Integer},
145     ↳ log_motif_expectation::AbstractFloat,
146     ↳ cardinality_penalty::AbstractFloat, osi_emitting)
147     for i in 2:obsl+1 #i=1 is ithe lh_vec initializing 0,
148       ↳ i=2 is the score of the first background position
149       ↳ (ie t=1)
150       t=i-1
151       score = lps(lh_vec[i-1], bg_scores[t],
152           ↳ cardinality_penalty)
153       #logic: all observations are scored from t=wml to
154       ↳ the end of the obs-therefore check at each
155       ↳ position for new sources to add (indexed by
156       ↳ vector position to retrieve source wml and
157       ↳ score matrix)
158       if
159         ↳ length(osi_emitting)<length(obs_source_indices)
160         for n in 1:length(obs_source_indices)
161           if !(n in osi_emitting)
162             t ≥ source_wmls[n] &&
163               ↳ (push!(osi_emitting,n))
164           end
165         end
166       end
167
168       for n in osi_emitting
169         wml = source_wmls[n]

```

```

155         from_score = lh_vec[i-wml+1] #score at the
156             ↵ first position of the PWM
157         score_array = score_mat[n] #get the source
158             ↵ score matrix
159         score_idx = t - wml + 1 #translate t to
160             ↵ score_array idx for emission score
161         f_emit_score = score_array[score_idx]
162             ↵ #emission score at the last position of
163             ↵ the PWM
164
165             score=logaddexp(score, lps(from_score,
166             ↵ f_emit_score, log_motif_expectation))
167         end
168     lh_vec[i] = score
169   end
170   return lh_vec[obsL+1]
171 end

```

---

#### 17.6.4 /src/IPM/IPM\_prior\_utilities.jl

```

1 function read_fa_wms_tr(path::String)
2     wms=Vector{Matrix{Float64}}(){}
3     wm=zeros(1,4)
4     f=open(path)
5     for line in eachline(f)
6         prefix=line[1:2]
7         prefix == "01" && (wm=transpose([parse(Float64,i) for i in
8             ↵ split(line)[2:end]]))
9         prefix != "01" && prefix != "NA" && prefix != "PO" && prefix !=
10            ↵ "//" && (wm=vcat(wm, transpose([parse(Float64,i) for i in
11            ↵ split(line)[2:end]])))
12         prefix == "//" && push!(wms, wm)
13     end
14     return wms
15 end
16
17 #wm_samples are in decimal probability space, not log space
18 function assemble_source_priors(no_sources::Integer,
19     ↵ wm_samples::AbstractVector{<:AbstractMatrix{<:AbstractFloat}}|,
20     ↵ prior_wt::AbstractFloat=PRIOR_WT) #estimate a dirichlet prior on
21     ↵ wm_samples inputs; if the number of samples is lower than the number
22     ↵ of sources, return a false bool for init and permutation functions

```

```

16 source_priors = Vector{Union{Vector{Dirichlet{Float64}}, Bool}}()
17 for source in 1:no_sources
18     if source ≤ length(wm_samples)
19         push!(source_priors,
20             ↳ estimate_dirichlet_prior_on_wm(wm_samples[source],
21             ↳ prior_wt))
22     else
23         push!(source_priors, false)
24     end
25 end
26 return source_priors
27
28
29 function
30     ↳ estimate_dirichlet_prior_on_wm(wm::AbstractMatrix{<:AbstractFloat},
31     ↳ wt::AbstractFloat=PRIOR_WT)
32     for i in 1:size(wm)[1]
33         !(isprobvec(wm[i,:])) && throw(DomainError("Bad
34             ↳ weight vec supplied to
35             ↳ estimate_dirichlet_prior_on_wm! $(wm[i,:])"))
36     end
37     prior = Vector{Dirichlet{Float64}}()
38     for position in 1:size(wm)[1]
39         normvec=wm[position,:]
40         zero_idxs=findall(isequal(0.),wm[position,:])
41         normvec[zero_idxs].+=10^-99
42         push!(prior, Dirichlet(normvec.*wt))
43     end
44     return prior
45 end
46
47 function cluster_mix_prior!(df::DataFrame,
48     ↳ wms::AbstractVector{<:AbstractMatrix{<:AbstractFloat}})
49     mix=false(size(df,1),length(wms))
50     for (o, row) in enumerate(eachrow(df))
51         row.cluster ≠ 0 && (mix[o, row.cluster]=true)
52     end
53
54     represented_sources=unique(df.cluster)
55     return mix[:,represented_sources]
56 end

```

```

51 function infocenter_wms_trim(wm::AbstractMatrix{<:AbstractFloat},
52   → trimsize::Integer)
53   !(size(wm,2)==4) && throw(DomainError("Bad wm! 2nd dimension should
54   → be size 4"))
55   infovec=get_pwm_info(wm, logsw=false)
56   maxval, maxidx=findmax(infovec)
57   upstream_extension=Int(floor((trimsize-1)/2))
58   downstream_extension=Int(ceil((trimsize-1)/2))
59   1+upstream_extension+downstream_extension > size(wm,1) &&
60   → throw(DomainError("Src too short for trim! $upstream_extension
61   → $downstream_extension"))
62   return
63   → wm[max(1,maxidx-upstream_extension):min(maxidx+downstream_extension,size(wm,1))]
64 end
65
66
67 function filter_priors(target_src_no::Integer, target_src_size::Integer,
68   → prior_wms::AbstractVector{<:AbstractMatrix{<:AbstractFloat}},
69   → prior_mix::BitMatrix)
70   wms=Vector{Matrix{Float64}}(undef, target_src_no)
71   freqsort_ids=sortperm([sum(prior_mix[:,s]) for s in
72   → 1:length(prior_wms)])
73   for i in 1:target_src_no
74     target_src_idx=freqsort_ids[i]
75     wms[i]=infocenter_wms_trim(prior_wms[target_src_idx],
76     → target_src_size)
77   end
78   return wms
79 end
80
81 function combine_filter_priors(target_src_no::Integer,
82   → target_src_size::Integer,
83   → prior_wms::Tuple{<:AbstractVector{<:AbstractMatrix{<:AbstractFloat}},AbstractVect
84   → prior_mix::Tuple{BitMatrix,BitMatrix})
85   wms=Vector{Matrix{Float64}}(undef, target_src_no)
86   cat_wms=vcat(prior_wms[1],prior_wms[2])
87   first_freq=[sum(prior_mix[1][:,s]) for s in 1:length(prior_wms[1])]
88   second_freq=[sum(prior_mix[2][:,s]) for s in 1:length(prior_wms[2])]
89   freqsort_ids=sortperm(vcat(first_freq,second_freq))
90   for i in 1:target_src_no
91     target_src_idx=freqsort_ids[i]
92     wms[i]=infocenter_wms_trim(cat_wms[target_src_idx],
93     → target_src_size)
94   end

```

```
81     return wms  
82 end
```

## 17.6.5 /src/ensemble/IPM\_Engineer.jl

```

1 mutable struct IPM_Ensemble
2     path::String #ensemble models and popped-out posterior samples
3         ↳ serialised here
4     models::Vector{Model_Record} #ensemble keeps paths to serialised
5         ↳ models and their likelihood tuples rather than keeping the
6         ↳ models in memory
7
8     contour::AbstractFloat#current log_Li[end]
9     log_Li::Vector{AbstractFloat} #likelihood of lowest-ranked model
10    ↳ at iterate i
11    log_Xi::Vector{AbstractFloat} #amt of prior mass included in
12        ↳ ensemble contour at Li
13    log_wi::Vector{AbstractFloat} #width of prior mass covered in
14        ↳ this iterate
15    log_Liwi::Vector{AbstractFloat} #evidentiary weight of the
16        ↳ iterate
17    log_Zi::Vector{AbstractFloat} #ensemble evidence
18    Hi::Vector{AbstractFloat} #ensemble information
19
20    obs_array::Matrix{Integer} #observations T x o
21    obs_lengths::Vector{Integer}
22
23    source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:Abstract
24        ↳ #source pwm priors
25    mix_prior::Tuple{BitMatrix,AbstractFloat} #prior on %age of
        ↳ observations that any given source contributes to
26
27    bg_scores::Matrix{AbstractFloat} #precalculated background HMM
        ↳ scores, same dims as obs
28
29    sample_posterior::Bool
30    posterior_samples::Vector{Model_Record} #list of posterior sample
        ↳ records
31
32    model_counter::Integer

```

```

26      naive_lh::AbstractFloat #the likelihood of the background model
27      ↵ without any sources
28 end
29 #####IPM_Engineer FUNCTIONS
30 IPM_Engineer(path::String, no_models::Integer,
31   ↵ source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat},
32   ↵ mix_prior::Tuple{BitMatrix,<:AbstractFloat},
33   ↵ bg_scores::AbstractMatrix{<:AbstractFloat},
34   ↵ obs::AbstractArray{<:Integer}, source_length_limits;
35   ↵ posterior_switch::Bool=false) =
36 IPM_Engineer(
37   ↵ path,
38   ↵ assemble_IPMs(path, no_models, source_priors, mix_prior,
39   ↵   ↵ bg_scores, obs, source_length_limits) ... ,
40   ↵ [-Inf], #L0 = 0
41   ↵ [0], #X0 = 1
42   ↵ [-Inf], #w0 = 0
43   ↵ [-Inf], #Liwi0 = 0
44   ↵ [-Inf], #Z0 = 0
45   ↵ [0], #H0 = 0,
46   ↵ obs,
47   ↵ [findfirst(iszero,obs[:,o])-1 for o in 1:size(obs)[2]],
48   ↵ source_priors,
49   ↵ mix_prior,
50   ↵ bg_scores, #precalculated background score
51   ↵ posterior_switch,
52   ↵ Vector{String}(),
53   ↵ no_models+1,
54   ↵ IPM_likelihood(init_logPWM_sources(source_priors,
55   ↵   ↵ source_length_limits), obs, [findfirst(iszero,obs[:,o])-1 for
56   ↵   ↵ o in 1:size(obs)[2]], bg_scores,
57   ↵   ↵ falses(size(obs)[2],length(source_priors)))))

58 IPM_Engineer(worker_pool::AbstractVector{<:Integer}, path::String,
59   ↵ no_models::Integer,
60   ↵ source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat},
61   ↵ mix_prior::Tuple{BitMatrix,<:AbstractFloat},
62   ↵ bg_scores::AbstractMatrix{<:AbstractFloat}, obs::AbstractArray{<:Integer},
63   ↵ source_length_limits; posterior_switch::Bool=false) =
64 IPM_Engineer(
65   ↵ path,

```

```

53     distributed_IPM_assembly(worker_pool, path, no_models,
54         ← source_priors, mix_prior, bg_scores, obs,
55         ← source_length_limits) ... ,
56         [-Inf], #L0 = 0
57         [0], #X0 = 1
58         [-Inf], #w0 = 0
59         [-Inf], #Liwi0 = 0
60         [-Inf], #Z0 = 0
61         [0], #H0 = 0,
62         obs,
63         [findfirst(iszero,obs[:,o])-1 for o in 1:size(obs)[2]],
64         source_priors,
65         mix_prior,
66         bg_scores, #precalculated background score
67         posterior_switch,
68         Vector{String}(),
69         no_models+1,
70         IPM_likelihood(init_logPWM_sources(source_priors,
71             ← source_length_limits), obs, [findfirst(iszero,obs[:,o])-1 for
72             o in 1:size(obs)[2]], bg_scores,
73             ← falses(size(obs)[2],length(source_priors)))))

74 function assemble_IPMs(path::String, no_models::Integer,
75     ← source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat}},<:BitMatrix,<:AbstractFloat}},
76     ← mix_prior::Tuple{BitMatrix,<:AbstractFloat},
77     ← bg_scores::AbstractArray{<:AbstractFloat},
78     ← obs::AbstractArray{<:Integer},
79     ← source_length_limits::UnitRange{<:Integer})
80     ensemble_records = Vector{Model_Record}()
81     !isdir(path) && mkpath(path)

82     @assert size(obs)[2]==size(bg_scores)[2]

83     @showprogress 1 "Assembling IPM ensemble ... " for model_no in
84         1:no_models
85             model_path = string(path,'/',model_no)
86             if !isfile(model_path)
87                 model = ICA_PWM_Model(string(model_no),
88                     ← source_priors, mix_prior, bg_scores, obs,
89                     ← source_length_limits)
90                 serialize(model_path, model) #save the model to
91                     ← the ensemble directory

```

```

81     push!(ensemble_records,
82         → Model_Record(model_path,model.log_Li))
83     else #interrupted assembly pick up from where we left off
84         model = deserialize(model_path)
85         push!(ensemble_records,
86             → Model_Record(model_path,model.log_Li))
87     end
88
89     return ensemble_records, minimum([record.log_Li for record in
90                                     → ensemble_records])
91 end
92
93 function distributed_IPM_assembly(worker_pool::Vector{Int64},
94                                     → path::String, no_models::Integer,
95                                     → source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat}},<:BitMatrix},<:AbstractFloat}},
96                                     → mix_prior::Tuple{BitMatrix,<:AbstractFloat},
97                                     → bg_scores::AbstractArray{<:AbstractFloat},
98                                     → obs::AbstractArray{<:Integer},
99                                     → source_length_limits::UnitRange{<:Integer})
100    ensemble_records = Vector{Model_Record}()
101    !isdir(path) && mkpath(path)
102
103    @assert size(obs)[2]==size(bg_scores)[2]
104
105    model_chan=
106        → RemoteChannel(()→Channel{ICA_PWM_Model}(length(worker_pool)))
107    job_chan = RemoteChannel(()→Channel{Union{Tuple,Nothing}}(1))
108        put!(job_chan,(source_priors, mix_prior, bg_scores, obs,
109                     → source_length_limits))
110
111    sequence_workers(worker_pool, worker_assemble, job_chan,
112                     → model_chan)
113
114    assembly_progress=Progress(no_models, desc="Assembling IPM
115                                → ensemble ... ")
116
117    model_counter=check_assembly!(ensemble_records, path, no_models,
118                                → assembly_progress)
119
120    while model_counter ≤ no_models
121        wait(model_chan)
122        candidate=take!(model_chan)
123
124        if candidate.log_Li >= assembly_progress.current
125            assembly_progress.current=candidate.log_Li
126        end
127
128        if assembly_progress.current == maximum([record.log_Li for record in
129                                         → ensemble_records])
130            break
131        end
132
133    end
134
135    return ensemble_records, assembly_progress.current
136
137 end

```

```

110     model = ICA_PWM_Model(string(model_counter),
111         ↳ candidate.origin, candidate.sources,
112         ↳ candidate.source_length_limits, candidate.mix_matrix,
113         ↳ candidate.log_Li)
114     model_path=string(path,'/',model_counter)
115     serialize(model_path,model)
116     push!(ensemble_records,
117         ↳ Model_Record(model_path,model.log_Li))
118     model_counter+=1
119     next!(assembly_progress)
120
121 end

122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140

```

```

141                                     params=fetch(job_chan)
142                                     while !(fetch(job_chan) ===
143                                         → nothing)
144                                         model=ICA_PWM_Model(string(myid()),pa
145                                         put!(models_chan,model)
146                                         end
147
148 function Base.show(io::IO, e::IPM_Engsemble; nsrc=0, progress=false)
149     livec=[model.log_Li for model in e.models]
150     maxLH=maximum(livec)
151     printstyled(io, "ICA PWM Model Ensemble @ $(e.path)\n",
152                 → bold=true)
153     msg = @sprintf "Contour: %3.6e MaxLH:%3.3e Max/Naive:%3.3e log
154                 → Evidence:%3.6e" e.contour maxLH (maxLH-e.naive_lh)
155                 → e.log_Zi[end]
156     println(io, msg)
157     hist=UnicodePlots.histogram(livec, title="Ensemble Likelihood
158                 → Distribution")
159     show(io, hist)
160     println()
161     progress && return(nrows(hist.graphics)+6)
162 end

```

---

### 17.6.6 /src/ensemble/ensemble\_utilities.jl

```

1 function ensemble_history(e::IPM_Engsemble, bins=25)
2     !e.sample_posterior && throw(ArgumentError("This ensemble has no
3         → posterior samples to show a history for!"))
4     livec=vcat([model.log_Li for model in e.models],[model.log_Li for
5         → model in e.posterior_samples])
6     show(histogram(livec, nbins=bins))
7 end
8
9 function e_backup(e::IPM_Engsemble, tuner::Permute_Tuner)
10    cp(string(e.path,'/','ens'),string(e.path,'/','ens.bak'), force=true)
11    serialize(string(e.path,'/','ens'), e)
12    serialize(string(e.path,'/','tuner'), tuner)
13 end
14
15 function clean_ensemble_dir(e::IPM_Engsemble, model_pad::Integer;
16     → ignore_warn=false)

```

```

14     !ignore_warn && e.sample_posterior && throw(ArgumentError("Ensemble
15       is set to retain posterior samples and its directory should not
16       be cleaned!"))
17   end
18 end
19
20 function complete_evidence(e::IPM_Ensemble)
21   log_Z=e.log_Zi[end]
22   H=e.Hi[end]
23   live_weight=lps(e.log_Xi[length(e.log_Li)], -log(length(e.models)))
24   lis=sort([model.log_Li for model in e.models])
25   liwls=sort(lps.(lis, live_weight))
26
27   for (li,liwi) in zip(lis,liwls)
28     last_Z=log_Z
29     log_Z=logaddexp(log_Z,liwi)
30     H=lps(
31       (exp(lps(liwi,-log_Z)) * li),
32       (exp(lps(last_Z,-log_Z)) * lps(H,last_Z)),
33       -log_Z)
34   end
35
36   return log_Z, H
37 end
38
39 function measure_evidence(e::IPM_Ensemble)
40   log_Z, H = complete_evidence(e)
41   return measurement(log_Z,sqrt(abs(H)/length(e.models)))
42 end
43
44 function reset_ensemble!(e::IPM_Ensemble)
45   new_e=deepcopy(e)
46   for i in 1:length(e.models)
47     if string(i) in [basename(record.path) for record in e.models]
48       new_e.models[i]=e.models[findfirst(isequal(string(i)),
49         [basename(record.path) for record in e.models])]
50     else

```

```

50
    ↪ new_e.models[i]=e.posterior_samples[findfirst(isequal(string(i)),
    ↪ [basename(record.path) for record in
    ↪ e.posterior_samples])]
51     end
52 end
53
54 new_e.contour=minimum([record.log_Li for record in new_e.models])
55
56 new_e.log_Li=[new_e.log_Li[1]]
57 new_e.log_Xi=[new_e.log_Xi[1]]
58 new_e.log_wi=[new_e.log_wi[1]]
59 new_e.log_Liwi=[new_e.log_Liwi[1]]
60 new_e.log_Zi=[new_e.log_Zi[1]]
61 new_e.Hi=[new_e.Hi[1]]
62
63 new_e.posterior_samples=Vector{Model_Record}()
64
65 new_e.model_counter=length(new_e.models)+1
66
67 clean_ensemble_dir(new_e, 0; ignore_warn=true)
68 isfile(e.path*/"tuner") && rm(e.path*/"tuner")
69 serialize(e.path*/"ens", new_e)
70
71 return new_e
72 end
73
74 function move_ensemble!(e::IPM_Ensemble,path::String)
75     !isdir(path) && mkdir(path)
76     for file in readdir(e.path)
77         mv(e.path*'*file,path*'*file)
78     end
79
80     for (n,model) in enumerate(e.models)
81         e.models[n]=Model_Record(path*'*basename(model.path),
82             ↪ model.log_Li)
83     end
84     if e.sample_posterior
85         for (n,model) in enumerate(e.posterior_samples)
86             ↪ e.posterior_samples[n]=Model_Record(path*'*basename(model.path),
87             ↪ model.log_Li)
88         end

```

```

87 end

88
89 rm(e.path)
90 e.path=path
91 serialize(e.path*"/ens",e)
92 return e
93 end

94
95 function copy_ensemble!(e::IPM_Ensemble,path::String)
96 new_e=deepcopy(e)
97 !isdir(path) && mkdir(path)
98 for file in readdir(e.path)
99     cp(e.path*'*file,path*'*file, force=true)
100 end

101
102 for (n,model) in enumerate(e.models)
103     new_e.models[n]=Model_Record(path*'*basename(model.path),
104         ↪ model.log_Li)
105 end
106 if e.sample_posterior
107     for (n,model) in enumerate(e.posterior_samples)
108         ↪ new_e.posterior_samples[n]=Model_Record(path*'*basename(model.path)
109             ↪ model.log_Li)
110     end
111 end

112 new_e.path=path
113 serialize(new_e.path*"/ens",new_e)
114 return new_e
115 end

116 function rewind_ensemble(e::IPM_Ensemble,rewind_idx)
117     !e.sample_posterior && throw(ArgumentError("An ensemble not retaining
118         ↪ posterior samples cannot be rewound!"))
119     rewind_idx ≥ length(e.log_Li) && throw(ArgumentError("rewind_idx
120         ↪ must be less than the current iterate!"))

121 n=length(e.models)
122 max_model_no=length(e.log_Li)+length(e.models)-1
123 rewind_model_no=rewind_idx+length(e.models)-1
124 new_e = deepcopy(e)

```

```

125     rm_models=[string(name) for name in rewind_model_no+1:max_model_no]
126
127     filter!(model→!(basename(model.path) in rm_models),new_e.models)
128     filter!(model→!(basename(model.path) in
129         ↳ rm_models),new_e.posterior_samples)
130
131     while length(new_e.models) < n
132         push!(new_e.models,pop!(new_e.posterior_samples))
133     end
134     new_e.contour=new_e.log_Li[rewind_idx]
135     new_e.log_Li=new_e.log_Li[1:rewind_idx]
136     new_e.log_Xi=new_e.log_Xi[1:rewind_idx]
137     new_e.log_wi=new_e.log_wi[1:rewind_idx]
138     new_e.log_Liwi=new_e.log_Liwi[1:rewind_idx]
139     new_e.log_Zi=new_e.log_Zi[1:rewind_idx]
140     new_e.Hi=new_e.Hi[1:rewind_idx]
141
142     new_e.model_counter=length(new_e.models)+rewind_idx
143
144     return new_e
145 end
146
147 function show_models(e::IPM_Ensemble,idxs)
148     liperm=sortperm([model.log_Li for model in e.models],rev=true)
149     for idx in idxs
150         m=deserialize(e.models[liperm[idx]].path)
151         show(m)
152     end
153 end
154 function get_model(e::IPM_Ensemble,no)
155     return deserialize(e.path*'/'*string(no))
156 end
157
158 function reestimate_ensemble!(e::IPM_Ensemble, wi_mode="trapezoidal")
159     ↳ Ni=Int64.(round.(collect(-1:-1:-length(e.log_Li)+1)./e.log_Xi[2:end]))
160     push!(Ni, length(e.models))
161
162     #fix any borked starting values
163     e.log_Li[1]=-Inf #L0 = 0
164     e.log_Xi[1]=0. #X0 = 1
165     e.log_wi[1]=-Inf #w0 = 0

```

```

166     e.log_Liwi[1]=-Inf #Liwi0 = 0
167     e.log_Zi[1]=-Inf #Z0 = 0
168     e.Hi[1]=0. #H0 = 0,
169
170     for i in 1:length(e.log_Li)-1
171         j=i+1
172
173         e.log_Li[j]=e.posterior_samples[i].log_Li
174
175         e.log_Xi[j]=-i/Ni[i]
176
177         if wi_mode=="trapezoidal"
178             e.log_wi[j]= logsubexp(e.log_Xi[i], -j/Ni[j]) - log(2) #log
179                         ↪ width of prior mass spanned by the last step-trapezoidal
180                         ↪ approx
181         elseif wi_mode=="simple"
182             e.log_wi[j]= logsubexp(e.log_Xi[i], e.log_Xi[j]) #log width
183                         ↪ of prior mass spanned by the last step-simple approx
184         else
185             throw(ArgumentError("Unsupported wi_mode!"))
186         end
187         e.log_Liwi[j]=lps(e.log_Li[j],e.log_wi[j]) #log likelihood + log
188                         ↪ width = increment of evidence spanned by iterate
189         e.log_Zi[j]=logaddexp(e.log_Zi[i],e.log_Liwi[j])      #log evidence
190                         #information- dimensionless quantity
191         Hj=lps(
192             (exp(lps(e.log_Liwi[j],-e.log_Zi[j])) * e.log_Li[j]),
193             (exp(lps(e.log_Zi[i],-e.log_Zi[j])) *
194                 ↪ lps(e.Hi[i],e.log_Zi[i])),
195                 -e.log_Zi[j])
196         Hj ≡ -Inf ? (e.Hi[j]=0.) : (e.Hi[j]=Hj)
197     end
198 end

```

---

### 17.6.7 /src/nested\_sampler/converge\_ensemble.jl

---

```

1 function converge_ensemble!(e::IPM_Ensemble,
2   instruction::Permute_Instruct; max_iterates=typemax(Int64),
3   backup::Tuple{Bool, Integer}=(false, 0),
4   clean::Tuple{Bool, Integer, Integer}=(false, 0, 0), verbose::Bool=false,
5   converge_criterion::String="standard",
6   converge_factor::AbstractFloat=.001, progargs ... )
7   N = length(e.models); curr_it=length(e.log_li)
8
9   curr_it>1 && isfile(e.path*"/tuner") ?
10    (tuner=deserialize(e.path*"/tuner")) : (tuner =
11      Permute_Tuner(instruction)) #restore tuner from saved if any
12   wk_mon = Worker_Monitor([1])
13   meter = ProgressNS(e, wk_mon, tuner, 0.; start_it=curr_it,
14    progargs ... )
15
16   converge_check = get_convfunc(converge_criterion)
17   while !converge_check(e, converge_factor) && (curr_it ≤
18     max_iterates)
19     warn,step_report = nested_step!(e, instruction) #step the
20     ensemble
21     warn == 1 && #"1" passed for warn code means no workers persist;
22     all have hit the permute limit
23     (@error "Failed to find new models, aborting at current
24       iterate."; return e) #if there is a warning, just
25     return the ensemble and print info
26   curr_it += 1
27
28   tune_weights!(tuner, step_report)
29   instruction = tuner.inst
30
31   backup[1] && curr_it%backup[2] == 0 && e_backup(e,tuner) #every
32     backup interval, serialise the ensemble and instruction
33   clean[1] && !e.sample_posterior && curr_it%clean[2] == 0 &&
34     clean_ensemble_dir(e,clean[3]) #every clean interval, remove
35     old discarded models
36
37   update!(meter, converge_check(e,converge_factor,vals=true) ... )
38 end
39
40 if converge_check(e,converge_factor)
41   final_logZ = measure_evidence(e)

```

```

26     @info "Job done, sampled to convergence. Final logZ
27     ↵  $(final_logZ.val) ± $(final_logZ.err)"
28
29     e_backup(e,tuner)
30     clean[1] && !e.sample_posterior && clean_ensemble_dir(e,0) #final
31     ↵  clean
32     return final_logZ
33
34     elseif curr_it==max_iterates
35     @info "Job done, sampled to maximum iterate $max_iterates.
36     ↵  Convergence criterion not obtained."
37
38 end
39
40 function converge_ensemble!(e::IPM_Engine,
41     ↵  instruction::Permute_Instruct, wk_pool::Vector{Int64};
42     ↵  max_iterates=typemax(Int64), backup::Tuple{Bool,Integer}=(false,0),
43     ↵  clean::Tuple{Bool,Integer,Integer}=(false,0,0), verbose::Bool=false,
44     ↵  converge_criterion::String="standard",
45     ↵  converge_factor::AbstractFloat=.001, progargs ... )
46     N = length(e.models)
47
48     converge_check = get_convfunc(converge_criterion)
49     model_chan=
50     ↵  RemoteChannel(()→Channel{Tuple{Union{ICA_PWM_Model,String},Integer,
51     ↵  AbstractVector{<:Tuple{}}}}(10*length(wk_pool))) #channel to take
52     ↵  EM iterates off of
53     job_chan =
54     ↵  RemoteChannel(()→Channel{Tuple{<:AbstractVector{<:Model_Record},
55     ↵  Float64, Union{Permute_Instruct,String}}}(1))
56     put!(job_chan,(e.models, e.contour, instruction))
57
58     if !converge_check(e,converge_factor) #sequence workers only if not
59     ↵  already converged
60     @async sequence_workers(wk_pool, permute_IPM, e, job_chan,
61     ↵  model_chan)
62 end
63
64 curr_it=length(e.log_Li)

```

```

53     wk_mon=Worker_Monitor(wk_pool)
54     curr_it>1 && isfile(e.path*/tuner) ?
55         (tuner=deserialize(e.path*/tuner)) : (tuner =
56             Permute_Tuner(instruction)) #restore tuner from saved if any
57     meter = ProgressNS(e, wk_mon, tuner, 0.; start_it=curr_it,
58         progargs ... )
59
60     while !converge_check(e, converge_factor) && (curr_it ≤
61         max_iterates)
62         warn, step_report = nested_step!(e, model_chan, wk_mon) #step the
63             ensemble
64         warn = 1 && #"1" passed for warn code means no workers persist;
65             all have hit the permute limit
66             (@error "All workers failed to find new models, aborting
67                 at current iterate."; return e) #if there is a
68                 warning, just return the ensemble and print info
69         curr_it += 1
70         tune_weights!(tuner, step_report)
71         instruction = tuner.inst
72         take!(job_chan); put!(job_chan,(e.models,e.contour,instruction))
73         backup[1] && curr_it%backup[2] == 0 && e_backup(e,tuner) #every
74             backup interval, serialise the ensemble and instruction
75         clean[1] && !e.sample_posterior && curr_it%clean[2] == 0 &&
76             clean_ensemble_dir(e,clean[3]) #every clean interval, remove
77             old discarded models
78
79         update!(meter, converge_check(e,converge_factor,vals=true) ... )
80     end
81
82     take!(job_chan); put!(job_chan, (e.models, e.contour, "stop")) #stop
83         instruction terminates worker functions
84
85     if converge_check(e,converge_factor)
86         final_logZ = measure_evidence(e)
87         @info "Job done, sampled to convergence. Final logZ
88             $(final_logZ.val) ± $(final_logZ.err)"
89
90         e_backup(e,tuner)
91         clean[1] && !e.sample_posterior && clean_ensemble_dir(e,0) #final
92             clean
93         return final_logZ
94     elseif curr_it==max_iterates

```

```

81     @info "Job done, sampled to maximum iterate $max_iterates.
82         ↪ Convergence criterion not obtained."
83
84     e_backup(e,tuner)
85     clean[1] && !e.sample_posterior && clean_ensemble_dir(e,0) #final
86         ↪ clean
87     return e.log_Zi[end]
88
89     end
90
91     function evidence_converge(e, evidence_fraction;
92         ↪ vals=false)
93         val=lps(findmax([model.log_Li for model in
94             ↪ e.models])[1], e.log_Xi[end])
95         thresh=lps(log(evidence_fraction),e.log_Zi[end])
96         vals ? (return val, thresh) : (return val<thresh)
97     end
98
99     function compress_converge(e, compression_ratio;
100        ↪ vals=false)
101        val=findmax([model.log_Li for model in
102            ↪ e.models])[1]-e.contour
103        thresh=compression_ratio
104        vals ? (return val, thresh) : (return val<thresh)
105    end
106
107    function get_convfunc(criterion)
108        if criterion == "standard"
109            ↪ return evidence_converge
110        elseif criterion == "compression"
111            ↪ return compress_converge
112        else
113            throw(ArgumentError("Convergence criterion
114                ↪ $criterion not supported! Try \"standard\" or
115                ↪ \"compression\"."))
116        end
117    end
118
119
```

---

### 17.6.8 /src/nested\_sampler/nested\_step.jl

---

```

1 ##### IMPLEMENTATION OF JOHN SKILLINGS' NESTED SAMPLING ALGORITHM #####
2 function nested_step!(e::IPM_Ensemble, instruction::Permute_Instruct)
```

```

3     N = length(e.models) #number of sample models/particles on the
   ↵ posterior surface
4     i = length(e.log_Li) #iterate number, index for last values
5     j = i+1 #index for newly pushed values
6
7     e.contour, least_likely_idx = findmin([model.log_Li for model in
   ↵ e.models])
8     Li_model = e.models[least_likely_idx]
9
10    #SELECT NEW MODEL, SAVE TO ENSEMBLE DIRECTORY, CREATE RECORD AND PUSH
   ↵ TO ENSEMBLE
11    model_selected=false; step_report=0
12    while !model_selected
13        candidate,step_report=permute_IPM(e, instruction)
14        if !(candidate==nothing)
15            if !(candidate.log_Li in [m.log_Li for m in e.models])
16                model_selected=true
17
18        new_model_record =
19            Model_Record(string(e.path,'/',e.model_counter),
   ↵ candidate.log_Li);
20        e.sample_posterior && push!(e.posterior_samples,
   ↵ Li_model)#if sampling posterior, push the model
   ↵ record to the ensemble's posterior samples vector
21        deleteat!(e.models, least_likely_idx)
22        push!(e.models, new_model_record);
23
24        final_model=ICA_PWM_Model(string(e.model_counter),
   ↵ candidate.origin, candidate.sources,
   ↵ candidate.source_length_limits, candidate.mix_matrix,
   ↵ candidate.log_Li, candidate.permute_blacklist)
25        serialize(new_model_record.path, final_model)
26
27        e.model_counter +=1
28    end
29    else
30        push!(e.models, Li_model)
31        return 1, step_report
32    end
33
34    #UPDATE ENSEMBLE QUANTITIES

```

```

35     push!(e.log_Li, e.contour) #log likelihood of the least likely model
      ↵ - the current ensemble ll contour at Xi
36     push!(e.log_Xi, -i/N) #log Xi - crude estimate of the iterate's
      ↵ enclosed prior mass
37     push!(e.log_wi, logsubexp(e.log_Xi[i], -j/N) - log(2)) #log width of
      ↵ prior mass spanned by the last step-trapezoidal approx
38     push!(e.log_Liwi, lps(e.log_Li[j],e.log_wi[j])) #log likelihood + log
      ↵ width = increment of evidence spanned by iterate
39     push!(e.log_Zi, logaddexp(e.log_Zi[i],e.log_Liwi[j])) #log
      ↵ evidence
40     #information- dimensionless quantity. cf. MultiNest @
      ↵ https://github.com/farhanferoz/MultiNest/blob/master/MultiNest_v3.12/nested.F
      ↵ MultiNest now gives info only at the final step
41     Hj=lps(
42         (exp(lps(e.log_Liwi[j],-e.log_Zi[j])) * e.log_Li[j]),
        ↵ #information contribution of this step
43         (exp(lps(e.log_Zi[i],-e.log_Zi[j])) * lps(e.Hi[i],e.log_Zi[i])),
        ↵ #rescale last information by evidence
44         -e.log_Zi[j])
45     Hj ≡ -Inf ? push!(e.Hi,0.) : push!(e.Hi, Hj) #prevent problems from
      ↵ early strings of models with -Inf log likelihoods
46
47     return 0, step_report
48 end
49
50 function nested_step!(e::IPM_Ensemble, model_chan::RemoteChannel,
51   ↵ wk_mon::Worker_Monitor)
52     N = length(e.models)+1 #number of sample models/particles on the
      ↵ posterior surface- +1 as one has been removed in the distributed
      ↵ dispatch for converge_ensemble
53     i = length(e.log_Li) #iterate number, index for last values
54     j = i+1 #index for newly pushed values
55
56     e.contour, least_likely_idx = findmin([model.log_Li for model in
      ↵ e.models])
57     Li_model = e.models[least_likely_idx]
58
59     #SELECT NEW MODEL, SAVE TO ENSEMBLE DIRECTORY, CREATE RECORD AND
      ↵ REPLACE LEAST LIKELY MODEL
60     model_selected=false;wk=0;step_report=0
61     while !model_selected
62       @async wait(model_chan)
63       candidate,wk,step_report = take!(model_chan)

```

```

63      if !(candidate=="quit")
64          if (candidate.log_Li > e.contour) && !(candidate.log_Li in
65              [m.log_Li for m in e.models])
66              model_selected=true
67
68          new_model_record =
69              → Model_Record(string(e.path, '/', e.model_counter),
70                  → candidate.log_Li);
71          e.sample_posterior && push!(e.posterior_samples,
72              → Li_model)#if sampling posterior, push the model
73              → record to the ensemble's posterior samples vector
74          deleteat!(e.models, least_likely_idx)
75          push!(e.models, new_model_record);
76
77          final_model=ICA_PWM_Model(string(e.model_counter),
78              → candidate.origin, candidate.sources,
79              → candidate.source_length_limits, candidate.mix_matrix,
80              → candidate.log_Li, candidate.permute_blacklist)
81          serialize(new_model_record.path, final_model)
82
83
84      e.model_counter +=1
85  end
86  update_worker_monitor!(wk_mon,wk,true)
87 else
88     update_worker_monitor!(wk_mon,wk,false)
89     !any(wk_mon.persist) && (return 1,step_report)
90 end
91
92 #UPDATE ENSEMBLE QUANTITIES
93 push!(e.log_Li, minimum([model.log_Li for model in e.models])) #log
94     → likelihood of the least likely model - the current ensemble ll
95     → contour at Xi
96 push!(e.log_Xi, -i/N) #log Xi - crude estimate of the iterate's
97     → enclosed prior mass
98 push!(e.log_wi, lps(e.log_Xi[i], -((j+1)/N)-log(2))) #log width of
99     → prior mass spanned by the last step-trapezoidal approx
100 push!(e.log_Liwi, lps(e.log_Li[j],e.log_wi[j])) #log likelihood + log
101     → width = increment of evidence spanned by iterate
102 push!(e.log_Zi, logaddexp(e.log_Zi[i],e.log_Liwi[j]))    #log
103     → evidence
104
105 #information- dimensionless quantity

```

```

92     push!(e.Hi, lps(
93         (exp(lps(e.log_Liwi[j], -e.log_Zi[j])) * e.log_Li[j]), #term1
94         (exp(lps(e.log_Zi[i], -e.log_Zi[j])) *
95             ↳ lps(e.Hi[i], e.log_Zi[i])), #term2
96             -e.log_Zi[j])) #term3
97
98     return 0, step_report
99 end

```

---

### 17.6.9 /src/permuation/Permute\_Tuner.jl

```

1 mutable struct Permute_Tuner
2     inst::Permute_Instruct
3     velocities::Matrix{Float64}
4     successes::BitMatrix #(memoryxfunc)
5     tabular_display::DataFrame
6     time_history::Vector{Float64}
7     override::Bool
8 end
9
10 """
11     Permute_Tuner(instruction)
12
13 Generate a Permute_Tuner for the given instruction.
14 """
15 function Permute_Tuner(instruction::Permute_Instruct)
16     nfuncs=length(instruction.funcs)
17     funcnames=Vector{String}()
18     vels=ones(TUNING_MEMORY*instruction.func_limit,nfuncs)
19     succs=true.(TUNING_MEMORY*instruction.func_limit,nfuncs)
20     for (idx,func) in enumerate(instruction.funcs)
21         length(instruction.args[idx])>0 ?
22             (kwstr="($length(instruction.args[idx]))kwa") : (kwstr="")
23         push!(funcnames, string(nameof(func), kwstr))
24     end
25     tabular_display=DataFrame("Function"⇒funcnames,
26         "Succeed"⇒zeros(Int64,nfuncs), "Fail"⇒zeros(Int64,
27         nfuncs), "Velocity"⇒ones(nfuncs), "Weights"⇒instruction.weights)
28     instruction.override_time>0. ? (override=true) : (override=false)

```

```

28     return
29     → Permute_Tuner(instruction, vels, succs, tabular_display, zeros(CONVERGENCE_MEMORY)
30 end
31 """
32     tune_weights!(tuner, call_report)
33
34 Given a call_report from permute_IPM(), adjust tuner's Permute_Instruct
35     → weights for function success rate and likelihood surface velocity.
36 """
37 function tune_weights!(tuner::Permute_Tuner,
38     → call_report::Vector{Tuple{Int64,Float64,Float64}})
39     for call in call_report
40         funcidx, time, distance=call
41         distance!==-Inf &&
42             → (tuner.velocities[:,funcidx]=update_velocity!(tuner.velocities[:,funcidx],
43             → #do not push velocity to array if it has -Inf probability
44             → (usu no new model found))
45         if call==call_report[end]
46
47             → tuner.successes[:,funcidx]=update_sucvec!(tuner.successes[:,funcidx],
48             tuner.tabular_display[funcidx, "Succeed"]+=1
49     else
50
51         → tuner.successes[:,funcidx]=update_sucvec!(tuner.successes[:,funcidx],
52             tuner.tabular_display[funcidx, "Fail"]+=1
53     end
54
55     tuner.override && mean(tuner.time_history)>tuner.inst.override_time ?
56         → (tuner.inst.weights=tuner.inst.override_weights;
57         → tuner.tabular_display[!, "Weights"]=tuner.inst.override_weights) :
58             update_weights!(tuner)
59 end
60
61 function update_velocity!(velvec, time, distance)
62     popfirst!(velvec) #remove first value
63     vel = distance - log(time)
64     push!(velvec, distance-log(time))
65 end
66
67 function update_sucvec!(sucvec, bool)
68     popfirst!(sucvec)
69     push!(sucvec, bool)

```

```

61 end

62

63 function update_weights!(t::Permute_Tuner)
64     mvels=[mean(t.velocities[:,n]) for n in 1:length(t.inst.funcs)]
65     t.tabular_display[!, "Velocity"]=copy(mvels)

66

67     any(i→i<0,mvels) && (mvels.=+minimum(mvels)+1.) #to calculate
       → weights, scale negative values into >1.
68     pvec=[mvels[n]*(sum(t.successes[:,n])/length(t.successes[:,n])) for n
       → in 1:length(t.inst.funcs)]
69     pvec./=sum(pvec)
70     (any(pvec.<t.inst.min_clmps) || any(pvec.>t.inst.max_clmps)) &&
       → clamp_pvec!(pvec,t.inst.min_clmps,t.inst.max_clmps)

71     @assert isprobvec(pvec)
72     t.inst.weights=pvec; t.tabular_display[!, "Weights"]=pvec
73 end

74

75 """
76     clamp_pvec(pvec, tuner)
77

78 Clamp the values of a probability vector between the minimums and
       → maximums provided by a Permute_Tuner.
79 """
80

81         function clamp_pvec!(pvec, min_clmps, max_clmps)
82             #logic- first accumulate on low values, then distribute
               → excess from high values
83             any(pvec.<min_clmps) ? (low_clamped=false) :
               → (low_clamped=true)
84             while !low_clamped
85                 vals_to_accumulate=pvec.<min_clmps
86                 vals_to_deplete=pvec.>min_clmps

87

88                 → depletion=sum(min_clmps[vals_to_accumulate]--pvec[vals_to_accu
89
               → pvec[vals_to_accumulate]=+min_clmps[vals_to_accumulate]
90
               → pvec[vals_to_deplete]=--depletion/sum(vals_to_deplete)

91             !any(pvec.<min_clmps)&&(low_clamped=true)
92         end
93
94

```

```
95     any(pvec.>max_clmps) ? (high_clamped=false) :
96         ↳ (high_clamped=true)
97     while !high_clamped
98         vals_to_deplete=pvec.>max_clmps
99         vals_to_accumulate=pvec.<max_clmps
100
101         ↳ depletion=sum(pvec[vals_to_deplete]~-max_clmps[vals_to_deplete])
102         pvec[vals_to_deplete]=~max_clmps[vals_to_deplete]
103
104         ↳ pvec[vals_to_accumulate].+=depletion/sum(vals_to_accumulate)
105
106         !any(pvec.>max_clmps)&&(high_clamped=true)
107     end
108 function Base.show(io::IO, tuner::Permute_Tuner; progress=false)
109     show(io, tuner.tabular_display, rowlabel=:I, summary=false)
110     progress && return(size(tuner.tabular_display,1)+4)
111 end
```

### 17.6.10 /src/permuation/orthogonality\_helper.jl

```

1 ##ORTHOGONALITY_HELPER
2 function consolidate_srcs(con_idxs::Dict{Integer,Vector{Integer}}|,
3   ↪ m::ICA_PWM_Model, obs_array::AbstractMatrix{<:Integer},
4   ↪ obs_lengths::AbstractVector{<:Integer},
5   ↪ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat,
6   ↪ models::AbstractVector{<:Model_Record};
7   ↪ iterates::Integer=length(m.sources)*2, remote=false)
8   new_log_Li=-Inf; iterate = 1
9   T,0 = size(obs_array); T=T-1; S = length(m.sources)
10  new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
11  a, cache = IPM_likelihood(new_sources, obs_array, obs_lengths,
12    ↪ bg_scores, new_mix, true, true)

13
14 while new_log_Li <= contour && iterate <= iterates #until we produce
15   ↪ a model more likely than the lh contour or exceed iterates
16   new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
17   clean=Vector{Bool}(trues(0))

```

```

12     for host_src in filter(!in(vcat(values(con_idxs) ... )),  

13         keys(con_idxs)) #copy mix information to the source to be  

14         consolidated on as host  

15         for cons_src in con_idxs[host_src]  

16             new_mix[:,host_src]=[new_mix[o,host_src] ||  

17             new_mix[o,cons_src] for o in 1:size(new_mix,1)]  

18         end  

19     end  

20  

21     remote ? (merger_m = deserialize(rand(models).path)) : (merger_m  

22         = remotecall_fetch(deserialize, 1, rand(models).path))  

23         #randomly select a model to merge  

24     used_m_srcs=Vector{Int64}()  

25  

26     for src in unique(vcat(values(con_idxs) ... )) #replace all  

27         non-host sources with sources from a merger model unlike the  

28         one being removed  

29         distvec=[pwm_distance(m.sources[src][1],m_src[1]) for m_src  

30             in merger_m.sources]  

31         m_src=findmax(distvec)[2]  

32         while m_src in used_m_srcs  

33             distvec[m_src]=0.; m_src=findmax(distvec)[2]  

34             length(used_m_srcs)=length(merger_m.sources) &&  

35             break; break  

36         end  

37  

38         clean[new_mix[:,src]]=false #mark dirty any obs that start  

39             with the source  

40         new_sources[src]=merger_m.sources[m_src]  

41         new_mix[:,src]=merger_m.mix_matrix[:,m_src]  

42         clean[new_mix[:,src]]=false #mark dirty any obs that end  

43             with the source  

44         push!(used_m_srcs, m_src)  

45     end  

46  

47     if consolidate_check(new_sources)[1] #if the new sources pass the  

48         consolidate check  

49         new_log_Li, cache = IPM_likelihood(new_sources, obs_array,  

50             obs_lengths, bg_scores, new_mix, true, true, cache,  

51             clean) #assess likelihood  

52     end  

53  

54     iterate += 1

```

```

41     end
42
43     return ICA_PWM_Model("candidate", "consolidated $(m.origin)",
44         ↳ new_sources, m.source_length_limits, new_mix, new_log_Li)
44 end
45
46 function
47     ↳ consolidate_check(sources :: AbstractVector{<: Tuple{<: AbstractMatrix{<: AbstractFloat}, ...}}, ...
48         ↳ thresh=CONSOLIDATE_THRESH, revcomp=REVCOMP)
49     pass=true
50     lengthδmat=[size(src1[1],1) - size(src2[1],1) for src1 in sources,
51         ↳ src2 in sources]
52     cons_idxs=Dict{Integer, Vector{Integer}}()
53     for src1 in 1:length(sources), src2 in src1+1:length(sources)
54         if lengthδmat[src1,src2]==0
55             revcomp ? (info_condition =
56                 ↳ pwm_distance(sources[src1][1],sources[src2][1]) < thresh
57                 ||
58                 ↳ pwm_distance(sources[src1][1],revcomp_pwm(sources[src2][1]))
59                 < thresh) : (info_condition =
60                 ↳ (pwm_distance(sources[src1][1],sources[src2][1]) <
61                     thresh))
62             if info_condition
63                 if !in(src1,keys(cons_idxs))
64                     cons_idxs[src1]=[src2]; pass=false
65                 else
66                     push!(cons_idxs[src1], src2)
67                 end
68             end
69         end
70     end
71     return pass, cons_idxs
72 end
73
74 function pwm_distance(pwm1,pwm2)
75     minwml=min(size(pwm1,1),size(pwm2,1))
76     return sum([euclidean(exp.(pwm1[pos,:]),
77         ↳ exp.(pwm2[pos,:])) for pos in 1:minwml])/minwml
78 end
79

```

---

### 17.6.11 /src/permuation/permute\_control.jl

---

```

1 mutable struct Permute_Instruct
2     funcs :: AbstractVector{<:Function}
3     weights :: AbstractVector{<:AbstractFloat}
4     args :: AbstractVector{<:AbstractVector{<:Tuple{<:Symbol,<:Any}}}
5     model_limit :: Integer
6     func_limit :: Integer
7     min_clmps :: AbstractVector{<:AbstractFloat}
8     max_clmps :: AbstractVector{<:AbstractFloat}
9     override_time :: AbstractFloat
10    override_weights :: AbstractVector{<:AbstractFloat}
11    Permute_Instruct(funcs,
12                      weights,
13                      model_limit,
14                      func_limit;
15                      min_clmps=fill(.01,length(funcs)),
16                      max_clmps=fill(1.,length(funcs)),
17                      override_time=0.,
18                      override_weights=zeros(length(funcs)),
19                      args=[Vector{Tuple{Symbol,Any}}() for i in
19                         1:length(funcs)])=assert_permute_instruct(funcs,weights,args,
20                         &&
21                         new(funcs,weights,args,model_limit,func_limit,min_clmps,max_c
22 end
23
24 function
25     ← assert_permute_instruct(funcs,weights,args,model_limit,func_limit,min_clmps,
26     ← max_clmps, override_time, override_weights)
27
28     ← !(length(funcs)==length(args)==length(weights)==length(min_clmps)==length(max_
29     ← && throw(ArgumentError("A valid Permute_Instruct must have as
30     ← many tuning weights and argument vectors as functions!")))
31     model_limit<1 && throw(ArgumentError("Permute_Instruct limit on
32     ← models to permute must be positive Integer!"))
33     func_limit<1 && throw(ArgumentError("Permute_Instruct limit on
34     ← fuction calls per model permuted must be positive Integer!"))
35     any(min_clmps.≥max_clmps) && throw(ArgumentError("Permute_Instruct
36     ← max_clmps must all be greater than corresponding min_clmps!"))
37     sum(min_clmps)>1. && throw(ArgumentError("Sum of minimum clamps must
38     ← be <1.0 to maintain a valid probability vector!"))
39     any(max_clmps.>1.) && throw(ArgumentError("Permute_Tuner maximum
40     ← clamps cannot be >1.0!"))

```

```

29     override_time<0. && throw(ArgumentError("Permute_Instruct
30         → override_time must be 0 (disabled) or positive float!"))
31     override_time>0. && !isprobvec	override_weights) &&
32         → throw(ArgumentError("Permute_Instruct override_weights must be
33             → valid probvec!"))
34     return true
35 end
36
37
38 function permute_IPM(e::IPM_Engineering, instruction::Permute_Instruct)
39     call_report=Vector{Tuple{Int64,Float64,Float64}}()
40     for model = 1:instruction.model_limit
41         m_record = rand(e.models)
42         m = deserialize(m_record.path)
43
44         model_blacklist=copy(m.permute_blacklist)
45         filteridxs, filtered_funcs, filtered_weights, filtered_args =
46             → filter_permutes(instruction, model_blacklist)
47
48         for call in 1:instruction.func_limit
49             start=time()
50             funcidx=rand(filtered_weights)
51             permute_func=filtered_funcs[funcidx]
52
53                 → pos_args,kw_args=get_permfunc_args(permute_func,e,m,filtered_args[funcidx])
54             new_m=permute_func(pos_args...;kw_args...)
55
56                 → push!(call_report,(filteridxs[funcidx],time()-start,new_m.log_Li
57                     - e.contour))
58
59         if length(new_m.permute_blacklist) > 0 && new_m.log_Li ==
60             → -Inf #in this case the blacklisted function will not work
61             → on this model at all; it should be removed from the
62             → functions to use
63             vcat(model_blacklist,new_m.permute_blacklist)
64             filteridxs, filtered_funcs, filtered_weights,
65             → filtered_args = filter_permutes(instruction,
66             → model_blacklist)
67         end
68
69             dupecheck(new_m,m) && new_m.log_Li > e.contour &&
70                 → return new_m, call_report
71         end
72     end
73 end

```

```

59         return nothing, call_report
60     end
61
62 function permute_IPM(e::IPM_Ensemble, job_chan::RemoteChannel,
63     ↳ models_chan::RemoteChannel, comms_chan::RemoteChannel)
64     ↳ #ensemble.models is partially updated on the worker to populate
65     ↳ arguments for permute funcs
66         persist=true
67         id=myid()
68         put!(comms_chan,id)
69         while persist
70             wait(job_chan)
71             start=time()
72             e.models, e.contour, instruction = fetch(job_chan)
73             instruction = "stop" && (persist=false; break)
74
75             call_report=Vector{Tuple{Int64,Float64,Float64}}()
76             for model=1:instruction.model_limit
77                 found::Bool=false
78                 m_record = rand(e.models)
79
80                 remotecall_fetch(isfile,1,m_record.path) ? m =
81                 ↳ (remotecall_fetch(deserialize,1,m_record.path)) : (break)
82
83                 model_blacklist=copy(m.permute_blacklist)
84                 filteridxs, filtered_funcs, filtered_weights, filtered_args =
85                 ↳ filter_permutes(instruction, model_blacklist)
86
86                 for call in 1:instruction.func_limit
87                     start=time()
88                     funcidx=rand(filtered_weights)
89                     permute_func=filtered_funcs[funcidx]
90
91                     ↳ pos_args,kw_args=get_permfunc_args(permute_func,e,m,filtered_args)
92                     new_m=permute_func(pos_args ... ;kw_args ... )
93
94                     ↳ push!(call_report,(filteridxs[funcidx],time()-start,new_m.log_Li
95                     ↳ - e.contour))
96
97                     if length(new_m.permute_blacklist) > 0 && new_m.log_Li =
98                     ↳ -Inf #in this case the blacklisted function will not
99                     ↳ work on this model at all; it should be removed from
100                     ↳ the functions to use

```

```

91         vcat(model_blacklist,new_m.permute_blacklist)
92         filteridxs, filtered_funcs, filtered_weights,
93         ↳ filtered_args = filter_permutes(instruction,
94         ↳ model_blacklist)
95     end
96
97     dupecheck(new_m,m) && new_m.log_Li > e.contour &&
98     ↳ ((put!(models_chan, (new_m ,id, call_report)));
99     ↳ found=true; break)
100
101         end
102     found=true && break;
103     wait(job_chan)
104     fetch(job_chan)!=e.models && (break) #if the ensemble has
105     ↳ changed during the search, update it
106     model=instruction.model_limit &&
107     ↳ (put!(models_chan, ("quit", id,
108     ↳ call_report));persist=false)#worker to put
109     ↳ "quit" on channel if it fails to find a model
110     ↳ more likely than contour
111 end
112
113 function filter_permutes(instruction, model_blacklist)
114
115     ↳ filteridxs=findall(p→!in(p,model_blacklist),instruction.funcs)
116     filtered_funcs=instruction.funcs[filteridxs]
117     filtered_weights=filter_weights(instruction.weights,
118     ↳ filteridxs)
119     filtered_args=instruction.args[filteridxs]
120     return filteridxs, filtered_funcs, filtered_weights,
121     ↳ filtered_args
122 end
123
124 function filter_weights(weights, idxs)
125     deplete=0.
126     for (i, weight) in enumerate(weights)
127         !in(i, idxs) && (deplete+=weight)
128     end
129     weights[idxs].+=deplete/length(idxs)
130     new_weights=weights[idxs]./sum(weights[idxs])
131     return Categorical(new_weights)
132 end

```

```

122
123     function
124         ← get_permfunc_args(func::Function,e::IPM_Ensemble,
125         ← m::ICA_PWM_Model, args::Vector{Tuple{Symbol,Any}})
126         pos_args=[]
127         argparts=Base.arg_decl_parts(methods(func).ms[1])
128         argnames=[Symbol(argparts[2][n][1]) for n in
129             ← 2:length(argparts[2])]
130         for argname in argnames #assemble basic positional
131             ← arguments from ensemble and model fields
132             if argname == Symbol('m')
133                 push!(pos_args,m)
134             elseif argname in fieldnames(IPM_Ensemble)
135                 push!(pos_args,getfield(e,argname))
136             elseif argname in fieldnames(ICA_PWM_Model)
137                 push!(pos_args,getfield(m,argname))
138             else
139                 throw(ArgumentError("Positional argument
140                     ← $argname of $func not available in the
141                     ← ensemble or model!"))
142             end
143         end
144
145         kw_args = NamedTuple()
146         if length(args) > 0 #if there are any keyword
147             ← arguments to pass
148             sym=Vector{Symbol}()
149             val=Vector{Any}()
150             for arg in args
151                 push!(sym, arg[1]); push!(val, arg[2])
152             end
153             kw_args = (;zip(sym,val)...)
154         end
155
156         return pos_args, kw_args
157     end
158
159     function dupecheck(new_model, model)
160         (new_model.sources==model.sources &&
161         ← new_model.mix_matrix==model.mix_matrix) ? (return
162             ← false) : (return true)
163     end

```

---

### 17.6.12 /src/permuation/permute\_functions.jl

---

```

1 #DECORRELATION SEARCH PATTERNS
2 function permute_source(m::ICA_PWM_Model, models::Vector{Model_Record},
3   ↪ obs_array::AbstractMatrix{<:Integer},
3   ↪ obs_lengths::AbstractVector{<:Integer},
3   ↪ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat,
3   ↪ source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFlo
3   ↪ iterates::Integer=length(m.sources)*2,
3   ↪ weight_shift_freq::AbstractFloat=PWM_SHIFT_FREQ,
3   ↪ weight_shift_dist::Distributions.ContinuousUnivariateDistribution=PWM_SHIFT_DIST,
3   ↪ length_change_freq::AbstractFloat=PWM_LENGTHPERM_FREQ,
3   ↪ length_perm_range::UnitRange{<:Integer}=LENGTHPERM_RANGE,
3   ↪ remote=false)
3 #weight_shift_dist is given in decimal probability values- converted to
3   ↪ log space in permute_source_lengths!
4   new_log_Li=-Inf; iterate = 1
5   0 = size(obs_array,2);S = length(m.sources)
6   new_sources=deepcopy(m.sources);
7
8   a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
9     ↪ bg_scores, m.mix_matrix, true, true)
10
10  while new_log_Li ≤ contour && iterate ≤ iterates #until we produce
11    ↪ a model more likely than the lh contour or exceed iterates
12    new_sources=deepcopy(m.sources);
13    s = rand(1:S)
14    clean=Vector{Bool}(trues(0))
14    clean[m.mix_matrix[:,s]]=false #all obs with source are dirty
15
16    weight_shift_freq > 0 &&
17      ↪ (new_sources[s]=permute_source_weights(new_sources[s],
17      ↪ weight_shift_freq, weight_shift_dist))
17    rand() < length_change_freq &&
18      ↪ (new_sources[s]=permute_source_length(new_sources[s],
18      ↪ source_priors[s], m.source_length_limits, length_perm_range))
19
19    new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
20      ↪ obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
20      ↪ clean)
20    iterate += 1
21 end
22

```

```

23     cons_check, cons_idxs = consolidate_check(new_sources)
24     cons_check ? (return ICA_PWM_Model("candidate", "PS from $(m.name)",
25         ← new_sources, m.source_length_limits, m.mix_matrix, new_log_Li)) :
26     ← (return consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate",
27         ← "PS from $(m.name)", new_sources, m.source_length_limits,
28         ← m.mix_matrix, new_log_Li), obs_array, obs_lengths, bg_scores,
29         ← contour, models; remote=remote))
30 end
31
32 function permute_mix(m::ICA_PWM_Model,
33     ← obs_array::AbstractMatrix{<:Integer},
34     ← obs_lengths::AbstractVector{<:Integer},
35     ← bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
36     ← iterates::Integer=10,
37     ← mix_move_range::UnitRange=1:length(m.mix_matrix), remote=false)
38     new_log_Li=-Inf; iterate = 1; 0 = size(obs_array,2);
39     new_mix=falses(size(m.mix_matrix))
40
41     a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
42         ← bg_scores, m.mix_matrix, true, true)
43     dirty=false
44
45     while (new_log_Li ≤ contour || !dirty) && iterate ≤ iterates #until
46         ← we produce a model more likely than the lh contour or exceed
47         ← iterates
48         mix_moves=rand(mix_move_range)
49         mix_moves > length(m.mix_matrix) && (mix_moves =
50             ← length(m.mix_matrix))
51
52         new_mix, clean = mix_matrix_decorrelate(m.mix_matrix, mix_moves)
53             ← #generate a decorrelated candidate mix
54         c_log_li, c_cache = IPM_likelihood(m.sources, obs_array,
55             ← obs_lengths, bg_scores, new_mix, true, true, cache, clean)
56             ← #calculate the model with the candidate mix
57         positive_indices=c_cache.>(cache) #obtain any obs indices that
58             ← have greater probability than we started with
59         clean=Vector{Bool}(trues(0)-positive_indices)
60         if any(positive_indices) #if there are any such indices
61             new_log_Li, cache = IPM_likelihood(m.sources, obs_array,
62                 ← obs_lengths, bg_scores, new_mix, true, true, cache,
63                 ← clean) #calculate the new model
64             dirty=true
65     end

```

```

46     iterate += 1
47   end
48
49   return ICA_PWM_Model("candidate", "PM from $(m.name)", m.sources,
50                         ← m.source_length_limits, new_mix, new_log_Li, Vector{Function}())
51                         ← #no consolidate check is necessary as sources havent changed
52 end
53
54 function perm_src_fit_mix(m::ICA_PWM_Model,
55                           models::Vector{Model_Record}, obs_array::AbstractMatrix{<:Integer},
56                           obs_lengths::AbstractVector{<:Integer},
57                           bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat,
58                           source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat}}, <:AbstractMatrix{<:AbstractFloat}}}, iterates::Integer=length(m.sources)*2,
59                           weight_shift_freq::AbstractFloat=PWM_SHIFT_FREQ,
60                           length_change_freq::AbstractFloat=PWM_LENGTHPERM_FREQ,
61                           length_perm_range::UnitRange{<:Integer}=LENGTHPERM_RANGE, weight_shift_dist::Distr,
62                           remote=false)
63   new_log_Li=-Inf; iterate = 1
64   O = size(obs_array,2); S = length(m.sources)
65   new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
66   fit_mix in m.permute_blacklist ? (new_bl=m.permute_blacklist) :
67     (new_bl=Vector{Function}())
68
69   while new_log_Li ≤ contour && iterate ≤ iterates #until we produce
70     ← a model more likely than the lh contour or exceed iterates
71     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix);
72     ← tm_one=deepcopy(m.mix_matrix);
73     clean=Vector{Bool}(trues(O))
74     s = rand(1:S);
75     clean[new_mix[:,s]]=false #all obs starting with source are
76     ← dirty
77
78     new_mix[:,s]=false;tm_one[:,s]=true
79
80     weight_shift_freq > 0 &&
81       (new_sources[s]=permute_source_weights(new_sources[s],
82         weight_shift_freq, weight_shift_dist))
83     rand() < length_change_freq &&
84       (new_sources[s]=permute_source_length(new_sources[s],
85         source_priors[s], m.source_length_limits, length_perm_range))
86
87
88

```

```

69      l,zero_cache=IPM_likelihood(new_sources, obs_array, obs_lengths,
70          ↵ bg_scores, new_mix, true, true)
71      l,one_cache=IPM_likelihood(new_sources, obs_array, obs_lengths,
72          ↵ bg_scores, tm_one, true, true)
73      fit_mix=one_cache.≥ zero_cache

74      if REVCOMP #if we're looking at reverse strands, we want to see
75          ↵ if sources fit better on the reverse strand, and if so switch
76          ↵ over to the revcomp source and use its fitted mix
77          revsrc=(revcomp_pwm(new_sources[s][1]),new_sources[s][2])
78          revsrcs=deepcopy(new_sources)
79          revsrcs[s]=revsrc
80          l, revsrc_cache=IPM_likelihood(revsrcs, obs_array,
81              ↵ obs_lengths, bg_scores, tm_one, true, true)
82          rfit_mix=revsrc_cache.≥ zero_cache
83          lps(revsrc_cache[rfit_mix])>lps(one_cache[fit_mix]) &&
84              ↵ (new_sources=revsrcs;fit_mix=rfit_mix) #if the total
85              ↵ likelihood contribution for the revsource fit is greater
86              ↵ than the source fit, take the revsource
87      end

88      new_mix[:,s]=fit_mix

89      clean[fit_mix]::=false #all obs ending with source are dirty

90      new_log_Li = IPM_likelihood(new_sources, obs_array, obs_lengths,
91          ↵ bg_scores, new_mix, true, false, zero_cache, clean)
92      iterate += 1
93  end

94  function fit_mix(m::ICA_PWM_Model, models::Vector{Model_Record},
95      ↵ obs_array::AbstractMatrix{<:Integer},
96      ↵ obs_lengths::AbstractVector{<:Integer},
97      ↵ bg_scores::AbstractMatrix{<:AbstractFloat}; remote=false)

```



```

122 function random_decorrelate(m::ICA_PWM_Model,
123   → models::Vector{Model_Record}, obs_array::AbstractMatrix{<:Integer},
124   → obs_lengths::AbstractVector{<:Integer},
125   → bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat,
126   → source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat}},<:Integer}},
127   → iterates::Integer=length(m.sources)*2,
128   → source_permute_freq::AbstractFloat=SRC_PERM_FREQ,
129   → weight_shift_freq::AbstractFloat=PWM_SHIFT_FREQ,
130   → length_change_freq::AbstractFloat=PWM_LENGTHPERM_FREQ,
131   → weight_shift_dist::Distributions.ContinuousUnivariateDistribution=PWM_SHIFT_DIST,
132   → mix_move_range::UnitRange=1:size(m.mix_matrix,1), remote=false)
133   new_log_Li=-Inf; iterate = 1
134   O = size(obs_array,2); S = length(m.sources)
135   new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
136
137   a, cache = IPM_likelihood(new_sources, obs_array, obs_lengths,
138     → bg_scores, new_mix, true, true)
139
140   while new_log_Li ≤ contour && iterate ≤ iterates #until we produce
141     → a model more likely than the lh contour or exceed iterates
142     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
143     clean=Vector{Bool}(trues(O))
144     s = rand(1:S)
145     clean[new_mix[:,s]]*=false #all obs starting with source are
146     → dirty
147     rand() < source_permute_freq &&
148       → (new_sources[s]=permute_source_weights(new_sources[s],
149           → weight_shift_freq, weight_shift_dist))
150     rand() < length_change_freq &&
151       → (new_sources[s]=permute_source_length(new_sources[s],
152           → source_priors[s], m.source_length_limits))
153
154     → new_mix[:,s]=mixvec_decorrelate(new_mix[:,s],rand(mix_move_range))
155     clean[new_mix[:,s]]*=false #all obs ending with source are dirty
156     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
157       → obs_lengths, bg_scores, new_mix, true, true, cache, clean)
158     iterate += 1
159
160 end
161
162 cons_check, cons_idxs = consolidate_check(new_sources)

```

```

143   cons_check ? (return ICA_PWM_Model("candidate", "RD from $(m.name)",
144     ↵ new_sources, m.source_length_limits, new_mix, new_log_Li)) :
145   (return consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate", "RD
146     ↵ from $(m.name)", new_sources, m.source_length_limits, new_mix,
147     ↵ new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;
148     ↵ remote=remote))
149 end
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164

```

`cons_check ? (return ICA_PWM_Model("candidate", "RD from $(m.name)",  
 ↵ new_sources, m.source_length_limits, new_mix, new_log_Li)) :  
 (return consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate", "RD  
 ↵ from $(m.name)", new_sources, m.source_length_limits, new_mix,  
 ↵ new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;  
 ↵ remote=remote))  
end  
  
function shuffle_sources(m::ICA_PWM_Model,  
 ↵ models::AbstractVector{<:Model_Record},  
 ↵ obs_array::AbstractMatrix{<:Integer},  
 ↵ obs_lengths::AbstractVector{<:Integer},  
 ↵ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;  
 ↵ remote=false)  
 new_log_Li=-Inf; O = size(obs_array,2); S = length(m.sources)  
 new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)  
  
 a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,  
 ↵ bg_scores, m.mix_matrix, true, true)  
  
 remote ? (merger_m = deserialize(rand(models).path)) : (merger_m =  
 ↵ remotecall_fetch(deserialize, 1, rand(models).path))#randomly  
 ↵ select a model to merge  
  
 svec=[1:S ... ]  
  
 while new_log_Li <= contour && length(svec)>0 #until we produce a  
 ↵ model more likely than the lh contour or no more sources to  
 ↵ attempt merger  
 new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)  
 clean=Vector{Bool}(trues(0))  
  
 s = popat!(svec,rand(1:length(svec))) #randomly select a source  
 ↵ to merge  
  
 new_sources[s]=merger_m.sources[s];  
 ↵ new_mix[:,s] *= merger_m.mix_matrix[:,s] #shuffle in the merger  
 ↵ model source from this index  
  
 clean[m.mix_matrix[:,s]]*=false #mark dirty any obs that have the  
 ↵ source`

```

165     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
166     ↵ obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
167     ↵ clean) #assess likelihood
168 end
169
170 cons_check, cons_idxs = consolidate_check(new_sources)
171 cons_check ? (return ICA_PWM_Model("candidate","SS from
172   ↵ $(m.name)",new_sources, m.source_length_limits, new_mix,
173   ↵ new_log_Li,Vector{Function}())) : (return
174   ↵ consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate","SS from
175   ↵ $(m.name)",new_sources, m.source_length_limits, new_mix,
176   ↵ new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;
177   ↵ remote=remote))
178 end
179
180 function accumulate_mix(m::ICA_PWM_Model,
181   ↵ models :: AbstractVector{<:Model_Record},
182   ↵ obs_array :: AbstractMatrix{<:Integer},
183   ↵ obs_lengths :: AbstractVector{<:Integer},
184   ↵ bg_scores :: AbstractMatrix{<:AbstractFloat}, contour :: AbstractFloat;
185   ↵ remote=false)
186     new_log_Li=-Inf; O = size(obs_array,2); S = length(m.sources)
187     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
188     new_bl=Vector{Function}()
189
190     a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
191     ↵ bg_scores, m.mix_matrix, true, true)
192
193     remote ? (merger_m = deserialize(rand(models).path)) : (merger_m =
194       ↵ remotecall_fetch(deserialize, 1, rand(models).path))#randomly
195       ↵ select a model to merge
196
197     svec=[1:S ... ]
198
199     while new_log_Li ≤ contour && length(svec)>0 #until we produce a
200       ↵ model more likely than the lh contour or no more sources to
201       ↵ attempt merger
202       ↵ new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
203       ↵ clean=Vector{Bool}(trues(O))
204
205       ↵ s = popat!(svec,rand(1:length(svec))) #randomly select a source
206       ↵ to merge

```

```

188     distvec=[pwm_distance(src[1],m_src[1]) for src in m.sources,
189     ↵   m_src in merger_m.sources]
190     S > 1 ? merge_s=findmin(distvec)[2][2] :
191     ↵   merge_s=findmin(distvec)[2]
192
193     new_mix[:,s]=[new_mix[o,s] || merger_m.mix_matrix[o,merge_s] for
194     ↵   o in 1:size(new_mix,1)] #accumulate the mix vector for this
195     ↵   source
196
197     clean[m.mix_matrix[:,s]]=false #mark dirty any obs that have the
198     ↵   source
199     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
200     ↵   obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
201     ↵   clean) #assess likelihood
202
203     if new_log_Li <= contour #if accumulating on the host source
204     ↵   doesnt work, try copying over the merger source
205     new_sources[s]=merger_m.sources[merge_s]
206     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
207     ↵   obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
208     ↵   clean) #assess likelihood
209   end
210 end
211
212 (new_log_Li == m.log_Li || new_log_Li in [model.log_Li for model in
213   ↵   models]) && (new_log_Li=-Inf) #accumulate can sometimes duplicate
214   ↵   models if source mix is identical
215
216 cons_check, cons_idxs = consolidate_check(new_sources)
217 cons_check ? (return ICA_PWM_Model("candidate","AM from
218   ↵   $(m.name)",new_sources, m.source_length_limits, new_mix,
219   ↵   new_log_Li)) : (return consolidate_srcs(cons_idxs,
220   ↵   ICA_PWM_Model("candidate","AM from $(m.name)",new_sources,
221   ↵   m.source_length_limits, new_mix, new_log_Li), obs_array,
222   ↵   obs_lengths, bg_scores, contour, models; remote=remote))
223 end
224
225 function distance_merge(m::ICA_PWM_Model,
226   ↵   models::AbstractVector{<:Model_Record},
227   ↵   obs_array::AbstractMatrix{<:Integer},
228   ↵   obs_lengths::AbstractVector{<:Integer},
229   ↵   bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
230   ↵   remote=false)

```

```

209     new_log_Li=-Inf; O = size(obs_array,2); S = length(m.sources)
210     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
211
212     a, cache = IPM_likelihood(new_sources, obs_array, obs_lengths,
213     ↪ bg_scores, new_mix, true, true)
214
215     remote ? (merger_m = deserialize(rand(models).path)) : (merger_m =
216     ↪ remotecall_fetch(deserialize, 1, rand(models).path)) #randomly
217     ↪ select a model to merge
218
219     svec=[1:S ... ]
220
221
222     while new_log_Li <= contour && length(svec)>0 #until we produce a
223     ↪ model more likely than the lh contour or no more sources to
224     ↪ attempt merger
225     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
226     clean=Vector{Bool}(trues(0))
227
228     s = popat!(svec,rand(1:length(svec))) #randomly select a source
229     ↪ to merge
230     merge_s=most_dissimilar(new_mix[:,s],merger_m.mix_matrix) #find
231     ↪ the source in the merger model whose mixvec is most
232     ↪ dissimilar to the one selected
233
234     clean[new_mix[:,s]]-=false #mark dirty any obs that start with
235     ↪ the source
236     new_sources[s] = merger_m.sources[merge_s] #copy the source
237     new_mix[:,s] = merger_m.mix_matrix[:,merge_s] #copy the mixvector
238     ↪ (without which the source will likely be highly improbable)
239     clean[new_mix[:,s]]-=false #mark dirty any obs that end with the
240     ↪ source
241
242     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
243     ↪ obs_lengths, bg_scores, new_mix, true, true, cache, clean)
244     ↪ #assess likelihood
245
246   end
247
248   cons_check, cons_idxs = consolidate_check(new_sources)

```

```

234     cons_check ? (return ICA_PWM_Model("candidate","DM from
235         $(m.name)",new_sources, m.source_length_limits, new_mix,
236         new_log_Li,Vector{Function}())) : (return
237         consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate","DM from
238         $(m.name)",new_sources, m.source_length_limits, new_mix,
239         new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;
240         remote=remote))
241     end
242
243     function similarity_merge(m::ICA_PWM_Model,
244         models::AbstractVector{<:Model_Record},
245         obs_array::AbstractMatrix{<:Integer},
246         obs_lengths::AbstractVector{<:Integer},
247         bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
248         remote=false)
249         new_log_Li=-Inf; O = size(obs_array,2); S = length(m.sources)
250         new_sources=deepcopy(m.sources)
251
252         a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
253             bg_scores, m.mix_matrix, true, true)
254
255         remote ? (merger_m = deserialize(rand(models).path)) : (merger_m =
256             remotecall_fetch(deserialize, 1, rand(models).path))#randomly
257             select a model to merge
258
259         svec=[1:S ... ]
260
261         while new_log_Li ≤ contour && length(svec)>0 #until we produce a
262             model more likely than the lh contour or no more sources to
263             attempt merger
264             new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
265             clean=Vector{Bool}(trues(O))
266
267             s = popat!(svec,rand(1:length(svec))) #randomly select a source
268                 to merge
269             merge_s=most_similar(m.mix_matrix[:,s],merger_m.mix_matrix)
270                 #obtain the source in the merger model whose mixvec is most
271                 similar to the one in the original
272
273             clean[m.mix_matrix[:,s]]*=false #mark dirty any obs that have the
274                 source
275             new_sources[s] = merger_m.sources[merge_s] #copy the source, but
276                 dont copy the mixvector

```

```

256     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
257         ↵ obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
258         ↵ clean) #assess likelihood
259     end
260
261     cons_check, cons_idxs = consolidate_check(new_sources)
262     cons_check ? (return ICA_PWM_Model("candidate", "SM from
263         ↵ $(m.name)", new_sources, m.source_length_limits, m.mix_matrix,
264         ↵ new_log_Li, Vector{Function}())) : (return
265         consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate", "SM from
266         ↵ $(m.name)", new_sources, m.source_length_limits, m.mix_matrix,
267         ↵ new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;
268         ↵ remote=remote))
269   end
270
271   function reinit_src(m::ICA_PWM_Model,
272     ↵ models::AbstractVector{<:Model_Record},
273     ↵ obs_array::AbstractMatrix{<:Integer},
274     ↵ obs_lengths::AbstractVector{<:Integer},
275     ↵ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat,
276     ↵ source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat}},<:Vector{Bool}}}},
277     ↵ iterates::Integer=length(m.sources)*2, remote=false)
278     new_log_Li=-Inf; iterate = 1
279     0 = size(obs_array,2); S = length(m.sources)
280     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
281
282     while new_log_Li <= contour && iterate <= iterates #until we produce
283         ↵ a model more likely than the lh contour or exceed iterates
284         new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix);
285         ↵ tm_one=deepcopy(m.mix_matrix);
286         clean=Vector{Bool}(trues(0))
287         s = rand(1:S);
288         clean[new_mix[:,s]] = false #all obs starting with source are
289         ↵ dirty
290
291         new_mix[:,s] = false; tm_one[:,s] = true
292
293         new_sources[s] = init_logPWM_sources([source_priors[s]],
294             ↵ m.source_length_limits)[1] #reinitialize the source
295
296         l, zero_cache=IPM_likelihood(new_sources, obs_array, obs_lengths,
297             ↵ bg_scores, new_mix, true, true)

```

```

279     l,one_cache=IPM_likelihood(new_sources, obs_array, obs_lengths,
280     ↵ bg_scores, tm_one, true, true)
281
282     fit_mix=one_cache.≥zero_cache
283
284     new_mix[:,s]=fit_mix
285
286     clean[fit_mix]=false #all obs ending with source are dirty
287
288     new_log_Li = IPM_likelihood(new_sources, obs_array, obs_lengths,
289     ↵ bg_scores, new_mix, true, false, zero_cache, clean)
290     iterate += 1
291   end
292
293   cons_check, cons_idxs = consolidate_check(new_sources)
294   cons_check ? (return ICA_PWM_Model("candidate","RS from $(m.name)",
295     ↵ new_sources, m.source_length_limits, new_mix,
296     ↵ new_log_Li,Vector{Function}())) : (return
297     ↵ consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate","RS from
298     ↵ $(m.name)",new_sources, m.source_length_limits, new_mix,
299     ↵ new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;
300     ↵ remote=remote))
301 end
302
303 function erode_model(m::ICA_PWM_Model,
304   ↵ models::AbstractVector{<:Model_Record},
305   ↵ obs_array::AbstractMatrix{<:Integer},
306   ↵ obs_lengths::AbstractVector{<:Integer},
307   ↵ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
308   ↵ info_thresh::AbstractFloat=EROSION_INFO_THRESH, remote=false)
309   new_log_Li=-Inf; 0 = size(obs_array,2);
310   new_sources=deepcopy(m.sources)
311
312   erosion_sources=Vector{Integer}()
313   for (s,src) in enumerate(m.sources)
314     pwm,pi=src
315     if size(pwm,1)>m.source_length_limits[1] #do not consider
316       ↵ eroding srcs at min length limit
317       infovec=get_pwm_info(pwm)
318       any(info→<(info, info_thresh),infovec) &&
319       ↵ push!(erosion_sources,s)
320   end
321 end
322
323 end

```

```

307
308 length(erosion_sources)==0 && return ICA_PWM_Model("candidate", "EM
309   ← from $(m.name)", new_sources, m.source_length_limits,
310   ← m.mix_matrix, new_log_Li,[erode_model])#if we got a model we cant
311   ← erode bail out with a model marked -Inf lh and with EM
312   ← blacklisted
313
314 a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
315   ← bg_scores, m.mix_matrix, true, true)
316
317 while new_log_Li ≤ contour && length(erosion_sources) > 0 #until we
318   ← produce a model more likely than the lh contour or there are no
319   ← more sources to erode
320   clean=Vector{Bool}(trues(0))
321   s=popat!(erosion_sources,rand(1:length(erosion_sources)))
322
323   new_sources[s]=erode_source(new_sources[s],
324     ← m.source_length_limits, info_thresh)
325   clean[m.mix_matrix[:,s]]-=false
326
327   new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
328     ← obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
329     ← clean) #assess likelihood
330
331 end
332
333 new_log_Li ≤ contour ? (blacklist=[erode_model]) :
334   ← (blacklist=Vector{Function}())
335
336 cons_check, cons_idxs = consolidate_check(new_sources)
337 cons_check ? (return ICA_PWM_Model("candidate", "EM from
338   ← $(m.name)", new_sources, m.source_length_limits, m.mix_matrix,
339   ← new_log_Li, blacklist)) : (return consolidate_srcs(cons_idxs,
340   ← ICA_PWM_Model("candidate", "EM from $(m.name)", new_sources,
341   ← m.source_length_limits, m.mix_matrix, new_log_Li, blacklist),
342   ← obs_array, obs_lengths, bg_scores, contour, models;
343   ← remote=remote))
344
345 end
346
347

```

```

328 function info_fill(m::ICA_PWM_Model,
329   → models :: AbstractVector{<:Model_Record},
330   → obs_array :: AbstractMatrix{<:Integer},
331   → obs_lengths :: AbstractVector{<:Integer},
332   → bg_scores :: AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
333   → remote=false)
334   new_log_Li=-Inf; O = size(obs_array,2); S = length(m.sources);
335   → iterate=1
336   new_sources=deepcopy(m.sources)
337
338   a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
339   → bg_scores, m.mix_matrix, true, true)
340
341   svec=[1:S ... ]
342
343   while new_log_Li ≤ contour && length(svec)>0 #until we produce a
344   → model more likely than the lh contour or no more sources to
345   → attempt infofill
346   new_sources=deepcopy(m.sources)
347   clean=Vector{Bool}(trues(0))
348   s = popat!(svec,rand(1:length(svec)))
349   pwm=new_sources[s][1]
350   fill_idx=findmin(get_pwm_info(pwm))[2]
351   fill_bases=[1,2,3,4]
352   fill_candidate=findmax(pwm[fill_idx,:])[2]
353   deleteat!(fill_bases, findfirst(b→b==fill_candidate,fill_bases))
354   new_sources[s][1][fill_idx,:]= -Inf;
355   → new_sources[s][1][fill_idx,fill_candidate]=0.
356   clean[m.mix_matrix[:,s]]=false
357   new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
358   → obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
359   → clean) #assess likelihood
360
361   while new_log_Li ≤ contour && length(fill_bases)>0
362     fill_candidate=popat!(fill_bases,rand(1:length(fill_bases)))
363     new_sources[s][1][fill_idx,:]= -Inf;
364     → new_sources[s][1][fill_idx,fill_candidate]=0.
365     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
366     → obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
367     → clean) #assess likelihood
368
369   end
370 end
371
372

```

```

356     new_log_Li ≤ contour ? (blacklist=[info_fill]) :
  ↵   (blacklist=Vector{Function}())
357
358     cons_check, cons_idxs = consolidate_check(new_sources)
359     cons_check ? (return ICA_PWM_Model("candidate","IF from
  ↵   $(m.name)",new_sources, m.source_length_limits, m.mix_matrix,
  ↵   new_log_Li, blacklist)) : (return consolidate_srcs(cons_idxs,
  ↵   ICA_PWM_Model("candidate","IF from $(m.name)",new_sources,
  ↵   m.source_length_limits, m.mix_matrix, new_log_Li, blacklist),
  ↵   obs_array, obs_lengths, bg_scores, contour, models;
  ↵   remote=remote))
360 end
361
362 full_perm_funcvec=[permute_source, permute_mix, perm_src_fit_mix,
  ↵   fit_mix, random_decorrelate, shuffle_sources, accumulate_mix,
  ↵   distance_merge, similarity_merge, reinit_src, erode_model, info_fill]

```

---

### 17.6.13 /src/permuation/permute\_utilities.jl

```

1 ##BASIC UTILITY FUNCTIONS
2 #SOURCE PERMUTATION
3 function
  ↵   permute_source_weights(source::Tuple{<:AbstractMatrix{<:AbstractFloat},<:Integer})
  ↵   shift_freq::AbstractFloat,
  ↵   PWM_shift_dist::Distribution{Univariate,Continuous})
4     dirty=false; source_length=size(source[1],1)
5     new_source=deepcopy(source)
6
7     for source_pos in 1:source_length
8       if rand() ≤ shift_freq
9         pos_WM = exp.(source[1][source_pos,:]) #leaving logspace, get
  ↵   the wm at that position
10        new_source[1][source_pos,:] = log.(wm_shift(pos_WM,
  ↵   PWM_shift_dist)) #accumulate probabiltiy at a randomly
  ↵   selected base, reassign in logspace and carry on
11        !dirty && (dirty=true)
12      end
13    end
14
15    if !dirty #if no positions were shifted, pick one and shift
16      rand_pos=rand(1:source_length)
17      pos_WM = exp.(source[1][rand_pos,:])

```

```

18     new_source[1][rand_pos,:] = log.(wm_shift(pos_WM, PWM_shift_dist))
19   end
20
21   return new_source
22 end
23
24   function
25     ↳  wm_shift(pos_WM::AbstractVector{<:AbstractFloat},
26     ↳  PWM_shift_dist::Distribution{Univariate,Continuous})
27     base_to_shift = rand(1:4) #pick a base to accumulate
28     ↳  probability
29     permute_sign = rand(-1:2:1)
30     shift_size = rand(PWM_shift_dist)
31     new_wm=zeros(4)
32
33     for base in 1:4 #ACGT
34       if base == base_to_shift
35         new_wm[base] =
36           clamp(0, #no lower than 0 prob
37                 (pos_WM[base] #selected PWM posn
38                  + permute_sign * shift_size), #randomly
39                  ↳  permuted by size param
40                  1) #no higher than prob 1
41       else
42         size_frac = shift_size / 3 #other bases
43         ↳  shifted in the opposite direction by 1/3
44         ↳  the shift accumulated at the base to
45         ↳  permute
46         new_wm[base] =
47           clamp(0,
48                 (pos_WM[base]
49                  - permute_sign * size_frac),
50                  1)
51     end
52   end
53   new_wm = new_wm ./ sum(new_wm) #renormalise to sum 1
54   ↳  - necessary in case of clamping at 0 or 1
55   !isprobvec(new_wm) && throw(DomainError(new_wm, "Bad
56   ↳  weight vector generated in wm_shift!")) #throw
57   ↳  assertion exception if the position WM is invalid
58   return new_wm
59 end
60

```

```

51
52
53 function
  ↳ permute_source_length(source::Tuple{<:AbstractMatrix{<:AbstractFloat},<:Integer},
  ↳ prior::Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat}},<:Bool},
  ↳ length_limits::UnitRange{<:Integer},
  ↳ permute_range::UnitRange{<:Integer}=LENGTHPERM_RANGE,
  ↳ uninformative::Dirichlet=Dirichlet([.25,.25,.25,.25]))
  source_PWM, prior_idx = source
  source_length = size(source_PWM,1)

56
57 permute_sign, permute_length = get_length_params(source_length,
  ↳ length_limits, permute_range)

58
59 permute_sign==1 ? permute_pos = rand(1:source_length+1) :
60           permute_pos=rand(1:source_length-permute_length)

61
62 if permute_sign == 1 #if we're to add positions to the PWM
  ins_WM=zeros(permute_length,4)
  if prior=false
    for pos in 1:permute_length
      ins_WM[pos,:] = log.(transpose(rand(uninformative)))
    end
  else
    for pos in 1:permute_length
      prior_position=permute_pos+prior_idx-1
      prior_position<1 || prior_position>length(prior) ?
        ins_WM[pos,:] = log.(transpose(rand(uninformative)))
        ↳ :
      ins_WM[pos,:] =
        ↳ log.(transpose(rand(prior[prior_position])))
    end
  end
  upstream_source=source_PWM[1:permute_pos-1,:]
  downstream_source=source_PWM[permute_pos:end,:]
  source_PWM=vcat(upstream_source,ins_WM,downstream_source)
  permute_pos==1 && (prior_idx-=permute_length)
else #if we're to remove positions
  upstream_source=source_PWM[1:permute_pos-1,:]
  downstream_source=source_PWM[permute_pos+permute_length:end,:]
  source_PWM=vcat(upstream_source,downstream_source)
  permute_pos==1 && (prior_idx+=permute_length)
end

```

```

86
87     return (source_PWM, prior_idx) #return a new source
88 end

89
90     function get_length_params(source_length::Integer,
91         → length_limits::UnitRange{<:Integer},
92         → permute_range::UnitRange{<:Integer})
93         extendable = length_limits[end]-source_length
94         contractable = source_length-length_limits[1]

95         if extendable == 0 && contractable > 0
96             permute_sign=-1
97         elseif contractable == 0 && extendable > 0
98             permute_sign=1
99         else
100             permute_sign = rand(-1:2:1)
101         end

102         permute_sign==1 && extendable<permute_range[end] &&
103             → (permute_range=permute_range[1]:extendable)
104         permute_sign==-1 && contractable<permute_range[end]
105             → && (permute_range=permute_range[1]:contractable)
106         permute_length = rand(permute_range)

107         return permute_sign, permute_length
108     end

109 function
110     → erode_source(source :: Tuple{<:AbstractMatrix{<:AbstractFloat}, <:Integer}, length_li
111     pwm,prior_idx=source
112     infovec=get_pwm_info(pwm)
113     start_idx,end_idx=get_erosion_idxs(infovec, info_thresh,
114         → length_limits)

115     return new_source=(pwm[start_idx:end_idx,:], prior_idx+start_idx-1)
116 end

117     function get_pwm_info(pwm :: AbstractMatrix{<:AbstractFloat};
118         → logsw::Bool=true)
119         wml=size(pwm,1)
120         infovec=zeros(wml)
121         for pos in 1:wml

```

```

121         logsw ? wvec=deepcopy(exp.(pwm[pos,:])) :
122             ↪ wvec=deepcopy(pwm[pos,:])
123             !isprobvec(wvec) && throw(DomainError(wvec, "Bad wvec in
124             ↪ get_pwm_info -Original sources must be in logspace!!"))
125             wvec.+=10^-99
126             infscore = (2.0 + sum([x*log(2,x) for x in wvec]))
127             infovec[pos]=infscore
128         end
129     return infovec
130 end
131
132 function get_erosion_idxs(infovec::AbstractVector{<:AbstractFloat},
133     ↪ info_thresh::AbstractFloat, length_limits::UnitRange{<:Integer})
134     srcl=length(infovec)
135     contractable = srcl-length_limits[1]
136     contractable ≤ 0 && throw(DomainError(contractable, "erode_source
137     ↪ passed a source at its lower length limit!"))
138     centeridx=findmax(infovec)[2]
139
140
141     start_idx=findprev(info→<(info,info_thresh),infovec,centeridx)
142     start_idx==nothing ? (start_idx=1) : (start_idx+=1)
143     end_idx=findnext(info→<(info, info_thresh),infovec,centeridx)
144     end_idx==nothing ? (end_idx=srcl) : (end_idx-=1)
145
146     pos_to_eroде=srcl-(end_idx-start_idx)
147     if pos_to_eroде > contractable
148         pos_to_restore = pos_to_eroде-contractable
149         while pos_to_restore>0
150             end_die=rand()
151             if end_die ≤ .5
152                 start_idx>1 && (pos_to_restore-=1; start_idx-=1)
153             else
154                 end_idx<srcl && (pos_to_restore-=1; end_idx+=1)
155             end
156         end
157     end
158
159     return start_idx, end_idx
160 end
161
162 #MIX MATRIX FUNCTIONS
163 function mixvec_decorrelate(mix::BitVector, moves::Integer)
164     new_mix=deepcopy(mix)

```

```

160     idxs_to_flip=rand(1:length(mix), moves)
161     new_mix[idxs_to_flip] *= .!mix[idxs_to_flip]
162     return new_mix
163 end
164
165 function mix_matrix_decorrelate(mix::BitMatrix, moves::Integer)
166     clean=Vector{Bool}(trues(size(mix,1)))
167     new_mix=deepcopy(mix)
168     indices_to_flip = rand(CartesianIndices(mix), moves)
169     new_mix[indices_to_flip] *= .!mix[indices_to_flip]
170     clean[unique([idx[1] for idx in indices_to_flip])] *= false #mark all
171     → obs that had flipped indices dirty
172     return new_mix, clean
173 end
174
175 # function most_dissimilar(mix1, mix2)
176 #     S1=size(mix1,2);S2=size(mix2,2)
177 #     dist_mat=zeros(S1,S2)
178 #     for s1 in 1:S1, s2 in 1:S2
179 #         dist_mat[s1,s2]=sum(mix1[:,s1].==mix2[:,s2])
180 #     end
181 #     scores=vec(sum(dist_mat,dims=1))
182 #     return findmin(scores)[2]
183 # end
184
185
186 function most_dissimilar(src_mixvec, target_mixmat)
187     src_sim = [sum(src_mixvec.==target_mixmat[:,s]) for s in
188     → 1:size(target_mixmat,2)] #compose array of elementwise equality
189     → comparisons between mixvectors and sum to score
190     merge_s=findmin(src_sim)[2] #source from merger model will be the one
191     → with the highest equality comparison score
192 end
193
194 function most_similar(src_mixvec, target_mixmat)
195     src_sim = [sum(src_mixvec.==target_mixmat[:,s]) for s in
196     → 1:size(target_mixmat,2)] #compose array of elementwise equality
197     → comparisons between mixvectors and sum to score
198     merge_s=findmax(src_sim)[2] #source from merger model will be the one
199     → with the highest equality comparison score
200 end

```

---

### 17.6.14 /src/utilities/model\_display.jl

---

```

1 #THIS CODE BASED ON N. REDDY'S THICWEED.JL DISPLAY SCRIPT
2
3 #CONSTANTS
4 # each tuple specifies the x, y, fontsize, xscale to print the character
5 # in a 100×100 box at position 100,100.
6 charvals_dict = Dict{Char,Tuple}('A'⇒(88.5,199.,135.,1.09),
7                                'C'⇒(79.,196.,129.,1.19),
8                                'G'⇒(83.,196.,129.,1.14),
9                                'T'⇒(80.,199.,135.,1.24))
10
11 colour_dict = Dict{Char,String}('A'⇒"(0,128,0)", #green
12                                'C'⇒"(0,0,128)", #blue
13                                'G'⇒"(255,69,0)", #yellow-brown
14                                'T'⇒"(150,0,0)") #red
15
16 #char output params for string display of PWMs in nested sampler
   ↪ instrumentation
17 lwcs_vec=['a','c','g','t']
18 upcs_vec=['A','C','G','T']
19 cs_vec[:green,:blue,:yellow,:red]
20 thresh_dict=Dict([(0.,'_'),(.25,'.'),(.5,"lwcs"),(1,"upcs")])
21
22 source_left_x = 10
23 xlen = 20
24 yscale = 250
25 scale_factor = 0.9
26 ypixels_per_source = 250
27 ypad = 10
28 xpad = 20
29 xpixels_per_position = 40
30 fontsize1=60
31 fontsize2=40
32
33 #function to convert ICA_PWM_Model sources lists to PWM sequence logo
   ↪ diagrams
34 function
   ↪ logo_from_model(model::ICA_PWM_Model,svg_output::String;freq_sort::Bool=false)
35     source_tups = Vector{Tuple{AbstractFloat, Integer,
   ↪ Matrix{AbstractFloat}}}() #(%of sequences w/ source, prior index,
   ↪ weight matrix)
36     mix = model.mix_matrix #o x s

```

```

37   for (prior, source) in enumerate(model.sources)
38     push!(source_tups,
39       (sum(model.mix_matrix[:,prior])/size(model.mix_matrix)[1],
40        prior, exp.(source[1])))
41   end
42
43 freq_sort && sort(source_tups)
44
45 file = open(svg_output, "w")
46 write(file,
47   ← svg_header(xpad+xpixels_per_position*maximum([length(source[1])
48   ← for source in
49   ← model.sources]),ypixels_per_source*length(model.sources)+ypad))
50
51 curry = 0
52 for (frequency, index, source) in source_tups
53   curry += yscale
54   font1y = curry-190
55   ndig = ndigits(index+1)
56   write(file,
57     ← pwm_to_logo(source,source_left_x,curry,xlen,yscale*scale_factor))
58   write(file, "<text x=\"$10\" y=\"$font1y\""
59     ← font-family=\"Helvetica\" font-size=\"$fontsize1\""
60     ← font-weight=\"bold\" >$index</text>")
61   write(file, "<text x=\"$((20+$fontsize1*0.6*ndig))\""
62     ← y=\"$($font1y-$fontsize1+1.5*$fontsize2)\""
63     ← font-family=\"Helvetica\" font-size=\"$fontsize2\""
64     ← font-weight=\"bold\" >$frequency*100) % of
65     ← sequences</text>\n")
66 end
67
68 write(file, svg_footer())
69 @info "Logo written."
70 close(file)
71 end
72
73 function svg_header(canvas_size_x, canvas_size_y)
74   return """
75   <?xml version="1.0" standalone="no"?>
76   <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
77   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
78   <svg width="$canvas_size_x" height="$canvas_size_y"
79   version="1.1" xmlns="http://www.w3.org/2000/svg">
```

```

68     """
69 end
70
71 function pwm_to_logo(source,xpos,ypos,xlen,yscale)
72     outstr = ""
73     for n in 1:size(source,1)
74         outstr *=
75             → print_weightvec_at_x_y(source[n,:],xpos+xlen*n,ypos,xlen,yscale)
76     end
77     outstr *= "\n"
78     return outstr
79 end
80
81 function pwmstr_to_io(io,source;log=true)
82     log && (source=exp.(source))
83     for pos in 1:size(source,1)
84         char,color=uni_wvec_params(source[pos,:])
85         printstyled(io, char; color=color)
86     end
87 end
88 function uni_wvec_params(wvec) #assign a single unicode char and color
89     ← symbol for the most informational position in a position weight
90     ← vector
91     wvec.+=10^-99
92     infoscore=(2.0 + sum([x*log(2,x) for x in wvec]))
93     infovec = [x*infoscore for x in wvec]
94     val,idx=findmax(infovec)
95     return char,color=get_char_by_thresh(idx,val)
96 end
97
98 function get_char_by_thresh(idx,val)
99     char = '?'; seen_thresh=0.
100    for (thresh,threshchar) in thresh_dict
101        val ≥ thresh && thresh ≥ seen_thresh &&
102            → (char=threshchar; seen_thresh=thresh)
103    end
104    char=='?' && println(val)
105    char=="lwcs" && (char=lwcs_vec[idx])
106    char=="upcs" && (char=upcs_vec[idx])
107    color=cs_vec[idx]
108
109    return char,color
110 end

```

```

107
108 function print_weightvec_at_x_y(wvec, xpos, ypos, xlen, yscale)
109     # xlen is the length occupied by that column
110     # yscale is the total height for 2 information bits i.e. the
111     # maximum height available for a "perfect" nucleotide
112     outstr = ""
113     basestr = "ACGT"
114     wvec.+=10^-99 #prevent log(0) = -Inf
115     wvec = [x/sum(wvec) for x in wvec] #renorm
116     infscore = (2.0 + sum([x*log(2,x) for x in wvec]))
117     if infscore==0.0
118         return ""
119     end
120     wvec = [x*infscore*yscale/2.0 for x in wvec]
121     # at this point, the sum of all wvec is a maximum of yscale
122     wveclist = [(wvec[n],basestr[n]) for n in 1:4]
123     wveclist = sort(wveclist)
124     curr_ypos = ypos
125     for n in 1:4
126         curr_ypos -= wveclist[n][1]
127         outstr *=
128             → print_char_in_rect(wveclist[n][2],xpos,curr_ypos,xlen,wveclist[n][1])
129     end
130     return outstr
131 end
132 blo = 1
133
134 function print_char_in_rect(c,x,y,width,height)
135     raw_x, raw_y, raw_fontsize, raw_xsclae = charvals_dict[c]
136     raw_x = (raw_x*raw_xsclae-100)/raw_xsclae
137     raw_y = raw_y-100
138
139     xscale = width/100.0 * raw_xsclae
140     yscale = height/100.0
141
142     scaled_x = x/xscale + raw_x
143     scaled_y = y/yscale + raw_y
144
145     return "<text x=\"$scaled_x\" y=\"$scaled_y\""
146         → font-size=\"$raw_fontsize\" font-weight=\"$bold\""
147         → font-family=\"$Helvetica\" fill=\"$rgb*colour_dict[c]*\""
148         → transform=\"scale($xscale,$yscale)>$c</text>\n"

```

```

146 end

147

148 function svg_footer()
149     return "</svg>"
150 end

```

---

### 17.6.15 /src/utilities/ns\_progressmeter.jl

```

1 #UTILITY REPORTS WORKER NUMBER AND CURRENT ITERATE
2 mutable struct ProgressNS{T<:Real} <: AbstractProgress
3     interval::T
4     dt::AbstractFloat
5     start_it::Integer
6     counter::Integer
7     triggered::Bool
8     tfirst::AbstractFloat
9     tlast::AbstractFloat
10    tstop::AbstractFloat
11    printed::Bool      # true if we have issued at least one status
12    ↵ update
13    desc::AbstractString # prefix to the percentage, e.g. "Computing ... "
14    color::Symbol        # default to green
15    output::IO           # output stream into which the progress is
16    ↵ written
17    numprintedvalues::Integer # num values printed below progress in
18    ↵ last iteration
19    offset::Integer       # position offset of progress bar
20    ↵ (default is 0)
21
22    e::IPM_E ensemble
23    wm::Worker_Monitor
24    tuner::Permute_Tuner
25    top_m::ICA_PWM_Model
26
27    mean_stp_time::AbstractFloat
28
29    wk_disp::Bool
30    tuning_disp::Bool
31    conv_plot::Bool
32    ens_disp::Bool
33    lh_disp::Bool
34    liwi_disp::Bool

```



```

72     0,
73     offset,
74     e,
75     wm,
76     tuner,
77     top_m,
78     0.,
79     wk_disp,
80     tuning_disp,
81     conv_plot,
82     ens_disp,
83     lh_disp,
84     liwi_disp,
85     src_disp,
86     nsrcts,
87     disp_rotate_inst,
88     zeros(CONVERGENCE_MEMORY))
89   end
90 end
91
92 function ProgressNS(e::IPM_Ensemble, wm::Worker_Monitor,
93   → tuner::Permute_Tuner, interval::Real; dt::Real=0.1,
94   → desc::AbstractString="Nested Sampling:: ", color::Symbol=:green,
95   → output::IO=stderr, offset::Integer=0, start_it::Integer=1,
96   → wk_disp::Bool=false, tuning_disp::Bool=false, conv_plot::Bool=false,
97   → lh_disp::Bool=false, liwi_disp::Bool=false, ens_disp::Bool=false,
98   → src_disp::Bool=false, nsrcts=0,
99   → disp_rotate_inst::Vector{Any}=[false,0,0,Vector{Vector{Symbol}}()])
100  top_m = deserialize(e.models[findmax([model.log_Li for model in
101    → e.models])[2]].path)

102  return ProgressNS{typeof(interval)}(e, top_m, wm, tuner, interval,
103    → dt=dt, desc=desc, color=color, output=output, offset=offset,
104    → start_it=start_it, wk_disp=wk_disp, tuning_disp=tuning_disp,
105    → conv_plot=conv_plot, lh_disp=lh_disp, liwi_disp=liwi_disp,
106    → ens_disp=ens_disp, src_disp=src_disp, nsrcts=nsrcts,
107    → disp_rotate_inst=disp_rotate_inst)
108 end
109
110
111 function update!(p::ProgressNS, val, thresh; options ... )
112   p.counter += 1

```

```

102     p.tstp=time()-p.tlast
103     popfirst!(p.tuner.time_history)
104     push!(p.tuner.time_history, p.tstp)
105
106     p.interval = val - thresh
107     popfirst!(p.convergence_history)
108     push!(p.convergence_history, p.interval)
109
110     p.top_m.name != basename(p.e.models[findmax([model.log_Li for model
111         in p.e.models])[2]].path) &&
112         (p.top_m=deserialize(p.e.models[findmax([model.log_Li for model
113         in p.e.models])[2]].path))
114
114
115     function updateProgress!(p::ProgressNS; offset::Integer = p.offset, keep
116         = (offset == 0))
117         p.offset = offset
118         t = time()
119         p.disp_rotate_inst[1] && p.counter%p.disp_rotate_inst[2] == 0 &&
120             rotate_displays(p)
121
121
122         if p.interval ≤ 0 && !p.triggered
123             p.triggered = true
124             if p.printed
125                 p.triggered = true
126                 dur = durationstring(t-p.tfirst)
127                 msg = @sprintf "%s Converged. Time: %s (%d iterations). logZ:
128                     %s\n" p.desc dur p.counter p.e.log_Zi[end]
129
130                 print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
131                 move_cursor_up_while_clearing_lines(p.output,
132                     p.numprintedvalues)
133                 upper_lines=display_upper_dash(p)
134                 printover(p.output, msg, :magenta)
135                 lower_lines=display_lower_dash(p)
136
137                 p.numprintedvalues=upper_lines + lower_lines + 1
138
139             if keep
140                 println(p.output)
141             else

```

```

138         print(p.output, "\r\u1b[A" ^ (p.offset +
139             ↵ p.numprintedvalues))
140     end
141   return
142 end
143
144 if t > p.tlast+p.dt && !p.triggered
145   p.counter < CONVERGENCE_MEMORY ?
146     ↵ mean_step_time=mean(p.tuner.time_history[end-(p.counter-1):end])
147     ↵ :
148       ↵ mean_step_time=mean(p.tuner.time_hist
149 msg = @sprintf "%s Iterate: %s Recent step time μ: %s Convergence
150   ↵ Interval: %g\n" p.desc p.counter hmss(mean_step_time)
151   ↵ p.interval
152
153 print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
154 move_cursor_up_while_clearing_lines(p.output, p.numprintedvalues)
155 upper_lines=display_upper_dash(p)
156 printover(p.output, msg, p.color)
157 lower_lines=display_lower_dash(p)
158
159 p.numprintedvalues=upper_lines + lower_lines + 1
160 print(p.output, "\r\u1b[A" ^ (p.offset + p.numprintedvalues))
161
162 # Compensate for any overhead of printing. This can be
163 # especially important if you're running over a slow network
164 # connection.
165 p.tlast = t + 2*(time()-t)
166 p.printed = true
167
168 end
169 end
170
171 function rotate_displays(p)
172   curr_inst=p.disp_rotate_inst[3]
173   curr_inst+1>length(p.disp_rotate_inst[4]) ?
174     ↵ next_inst=1 : next_inst=curr_inst+1
175   next_disps=p.disp_rotate_inst[4][next_inst]
176   for disp in
177     ↵ [wk_disp,:tuning_disp,:conv_plot,:ens_disp,:lh_disp,:liwi_di
178       disp in next_disps ? setproperty!(p, disp, true)
179         ↵ : setproperty!(p, disp, false)
180   end

```

```

172         p.disp_rotate_inst[3]=next_inst
173     end
174
175     function hmss(dt)
176         dt<0 ? (dt=-dt; prfx="-") : (prfx="")
177         isnan(dt) && return "NaN"
178         (h,r) = divrem(dt,60*60)
179         (m,r) = divrem(r, 60)
180         (isnan(h)||isnan(m)||isnan(r)) && return "NaN"
181         string(prfx,Int(h),":",Int(m),":",Int(ceil(r)))
182     end
183
184     function display_upper_dash(p::ProgressNS)
185         wklines = tunelines = cilines = 0
186         p.wk_disp && (wklines=show(p.output, p.wm,
187             ↳ progress=true);println())
188         p.tuning_disp && (tunelines=show(p.output, p.tuner,
189             ↳ progress=true);println())
190         if p.conv_plot
191             ↳ ciplot=lineplot([p.counter-(length(p.convergence_history)
192             ↳ p.convergence_history, title="Convergence
193             ↳ Interval Recent History",
194             ↳ xlabel="Iterate",ylabel="CI", color=:yellow)
195             cilines=nrows(ciplot.graphics)+5
196             show(p.output, ciplot); println()
197         end
198         return wklines + tunelines + cilines
199     end
200
201     function display_lower_dash(p::ProgressNS)
202         lhlines = liwilines = ensemblelines = srclines = 0
203         if p.lh_disp
204             lhplot=lineplot(p.e.log_Li[2:end], title="Contour
205             ↳ History", xlabel="Iterate", color=:magenta,
206             ↳ name="Ensemble logLH")
207             lineplot!(lhplot, [p.e.naive_lh for it in
208                 ↳ 1:length(p.e.log_Li[2:end])], name="Naive
209                 ↳ logLH")
210             lhlines=nrows(lhplot.graphics)+5
211             show(p.output, lhplot); println()
212         end
213         if p.liwi_disp && p.counter>2

```

```

205             ↵ liwiplot=lineplot([max(2,p.counter-(CONVERGENCE_MEMORY-1)
206             ↵ title="Recent iterate evidentiary weight",
207             ↵ xlabel="Iterate", name="Ensemble log Liwi",
208             ↵ color=:cyan)
209             liwilines=nrows(liwiplot.graphics)+5
210             show(p.output, liwiplot); println()
211             end
212             p.ens_disp && (ensemblelines=show(p.output, p.e,
213             ↵ progress=true))
214             p.src_disp && (println("MLE Model
215             ↵ Sources:");srclines=show(p.output, p.top_m,
216             ↵ nsrcc=p.no_displayed_srcs, progress=true))
217             return lhlines + liwilines + ensemblelines + srclines
218         end

```

---

### 17.6.16 /src/utilities/synthetic\_genome.jl

```

1 function synthetic_sample(no_obs::Integer, obsl,
2   ↵ bhmm_vec::AbstractVector{<:BHMM}, bhmm_dist::Categorical,
2   ↵ spikes::AbstractVector{<:AbstractMatrix{<:AbstractFloat}},
2   ↵ spike_instruct::AbstractVector{<:Tuple{<:Bool, <:Tuple}})
3   !(typeof(obsl) <:UnitRange || typeof(obsl) <:Integer) &&
4     ↵ throw(ArgumentError("obsl must be Integer or UnitRange"))
5   length(spike_instruct)!=length(spikes) &&
6     ↵ throw(ArgumentError("spike_instruct must be as long as spikes"))
7
8   obs, hmm_truth=obs_array_from_bhmms(no_obs, obsl, bhmm_vec, bhmm_dist)
9
10  spike_truth = spike_obs!(obs, spikes, spike_instruct)
11
12  bg_scores = score_synthetic(obs, bhmm_vec, hmm_truth)
13
14  return obs, bg_scores, hmm_truth, spike_truth
15 end
16
17 function obs_array_from_bhmms(no_obs, obsl, bhmm_vec, bhmm_dist)
18   obs=zeros(UInt8,max(obsl...)+1, no_obs)
19   truthvec=zeros(UInt8,no_obs)
20   for o in 1:no_obs
21     hmm_idx=rand(bhmm_dist)
22     truthvec[o]=hmm_idx
23   end
24 end

```

```

20     bhmm=bhmm_vec[hmm_idx]
21     typeof(obs1)<:UnitRange ? (l=rand(obs1)) : (l=obs1)
22     obs[1:l,o]=rand(bhmm, l)
23   end
24   return obs, truthvec
25 end
26
27 function spike_obs!(obs, spikes, spike_instruct)
28   truthmat=false.(size(obs,2),length(spikes))
29   for (s,spike) in enumerate(spikes)
30     structural,ins=spike_instruct[s]
31     structural ? (truthmat[:,s] = spike_struc!(obs, spike, ins...)) :
32       (truthmat[:,s] = spike_irreg!(obs, spike, ins...))
33   end
34   return truthmat
35 end
36 function spike_struc!(obs, spike, frac_obs, periodicity)
37   truth=false.(size(obs,2))
38   for o in 1:size(obs,2)
39     if rand() < frac_obs
40       truth[o]=true
41       rand()<.5 ? (source=revcomp_pwm(spike)) : (source=spike)
42       pos=rand(1:periodicity)
43       oidx=findfirst(iszero,obs[:,o])
44       while pos<oidx
45         pos_ctr=pos
46         pwm_ctr=1
47         while pos_ctr<oidx&&pwm_ctr≤size(source,1)
48           obs[pos_ctr,o]=rand(Categorical(source[pwm_ctr,:]))
49           pos_ctr+=1
50           pwm_ctr+=1
51         end
52         pos+=periodicity+pwm_ctr
53       end
54     end
55   end
56   return truth
57 end
58
59 function spike_irreg!(obs, spike, frac_obs, recur)
60   truth=false.(size(obs,2))
61   for o in 1:size(obs,2)

```

```

62     oidx=findfirst(iszero,obs[:,o])
63     if rand()<frac_obs
64         truth[o]=true
65         for r in 1:rand(recur)
66             rand()<.5 ? (source=revcomp_pwm(spike)) : source=spike
67             pos=rand(1:oidx-1)
68             pwm_ctr=1
69             while pos<oidx && pwm_ctr≤size(source,1)
70                 obs[pos,o]=rand(Categorical(source[pwm_ctr,:]))
71                 pos+=1
72                 pwm_ctr+=1
73             end
74         end
75     end
76     return truth
78 end
79
80 function score_synthetic(obs, bhmm_vec, hmm_truth)
81     lh_mat=zeros(size(obs,1)-1, size(obs,2))
82     for o in 1:size(obs,2)
83         oidx=findfirst(iszero,obs[:,o])
84
85         → lh_mat[1:oidx-1,o]=BioBackgroundModels.get_BGHMM_symbol_lh(transpose(obs[1:oidx-1,:]),bhmm_vec,hmm_truth)
86     end
87     return lh_mat
87 end

```

---

### 17.6.17 /src/utilities/worker\_diagnostics.jl

```

1 struct Worker_Monitor
2     idx::Dict{Integer, Integer}
3     persist::BitMatrix
4     last_seen::Matrix{Float64}
5 end
6
7 function Worker_Monitor(wk_pool::Vector{<:Integer})
8     idx=Dict{Integer, Integer}()
9     for (n,wk) in enumerate(wk_pool)
10        idx[wk]=n
11    end
12    persist=true(1,length(wk_pool))

```

```

13     last_seen=[time() for x in 1:1, y in 1:length(wk_pool)]
14     return Worker_Monitor(idx, persist, last_seen)
15 end
16
17 function update_worker_monitor!(mon,wk,persist)
18     mon.persist[1,mon.idx[wk]]=persist
19     mon.last_seen[1,mon.idx[wk]]=time()
20 end
21
22 function Base.show(io::IO, mon::Worker_Monitor; progress=false)
23     printstyled("Worker Diagnostics\n", bold=true)
24     pers=heatmap(float.(mon.persist), colormap=persistcolor,
25         → title="Persistence",labels=false)
26     ls=heatmap([time()-ls for ls in mon.last_seen], title="Last
27         → Seen",labels=false)
28     show(pers)
29     println()
30     show(ls)
31     progress && return nrows(pers.graphics)+nrows(ls.graphics)+7
32 end
33
34 function persistcolor(z, zmin, zmax)
35     z==1. && return 154
36     z==0. && return 160
37 end

```

---

### 17.6.18 /src/utilities/worker\_sequencer.jl

```

1 #waits for one worker to begin before calling the next so that network is
2     → not slammed trying to send huge arrays to many workers simultaneously
3
4 function sequence_workers(wk_pool, func, args...)
5     comms_chan = RemoteChannel(()→Channel{Integer}(length(wk_pool)))
6
7     for worker in wk_pool
8         remote_do(func, worker, args..., comms_chan)
9         wait(comms_chan); report=take!(comms_chan)
10        @assert worker==report
11    end
12 end

```

---

### 17.6.19 /test/consolidate\_unit\_tests.jl

---

```

1 @testset "Orthogonality helper" begin
2     bg_scores = log.(fill(.25, (17,3)))
3     obs=[BioSequences.LongSequence{DNAAlphabet{2}}]("ATGATTACGATGATGCA")
4     BioSequences.LongSequence{DNAAlphabet{2}}("TCAGTTACGATGATCAG")
5     BioSequences.LongSequence{DNAAlphabet{2}}("TTACGCACAGATTTAC")]
6     order_seqs = BioBackgroundModels.get_order_n_seqs(obs, 0)
7     coded_seqs = BioBackgroundModels.code_seqs(order_seqs)
8     obs=Array(transpose(coded_seqs))
9     obsl=[findfirst(iszero,obs[:,o])-1 for o in 1:size(obs)[2]]
10
11     src_ATG = [.7 .1 .1 .1
12                 .1 .1 .1 .7
13                 .1 .1 .7 .1]
14
15     src_CAG = [.1 .7 .1 .1
16                 .7 .1 .1 .1
17                 .1 .1 .7 .1]
18
19     src_TTAC = [.1 .1 .1 .7
20                 .1 .1 .1 .7
21                 .7 .1 .1 .1
22                 .1 .7 .1 .1]
23
24     src_GCA = [.1 .1 .7 .1
25                 .1 .7 .1 .1
26                 .7 .1 .1 .1]
27
28     consolidate_one =
29         [(log.(src_ATG),0),(log.(src_TTAC),0),(log.(src_ATG),0)]
30     cons_one_mix = BitMatrix([true true false
31                             true false true
32                             false true true])
33
34     consolidate_two =
35         [(log.(src_ATG),0),(log.(src_ATG),0),(log.(src_ATG),0)]
36     cons_two_mix = BitMatrix([true false false
37                             false true false
38                             false false true])
39
40     distance_model =
41         [(log.(src_TTAC),0),(log.(src_CAG),0),(log.(src_GCA),0)]

```

```

39
40    c1model = ICA_PWM_Model("c1", "c1", consolidate_one, 3:4,
41        ↵  cons_one_mix, IPM_likelihood(consolidate_one, obs, obsl,
42        ↵  bg_scores, cons_one_mix))
43    c2model = ICA_PWM_Model("c2", "c2", consolidate_two, 3:4,
44        ↵  cons_two_mix, IPM_likelihood(consolidate_two, obs, obsl,
45        ↵  bg_scores, cons_two_mix))
46    dmodel = ICA_PWM_Model("d", "d", distance_model, 3:4, trues(3,3),
47        ↵  IPM_likelihood(distance_model, obs, obsl, bg_scores, trues(3,3)))
48
49    dpath=randstring()
50    serialize(dpath, dmodel)
51    drec=Model_Record(dpath,dmodel.log_Li)
52
53    #check distance calculation
54    pwmtest_1=zeros(1,4);pwmtest_1[1]=1.
55    pwmtest_2=zeros(1,4);pwmtest_2[2]=1.
56    @test pwm_distance(log.(pwmtest_1),log.(pwmtest_2)) =
57        ↵  euclidean(pwmtest_1,pwmtest_2) = 1.4142135623730951
58
59    #test consolidate check
60    @test consolidate_check(distance_model) =
61        ↵  (true,Dict{Integer,Vector{Integer}}){}
62    @test consolidate_check(consolidate_one) = (false,
63        ↵  Dict{Integer,Vector{Integer}}{}(1⇒[3]))
64    @test consolidate_check(consolidate_two) = (false,
65        ↵  Dict{Integer,Vector{Integer}}{}(1⇒[2,3], 2⇒[3]))
66
67    #test overall consolidate function
68    _,con_idxs=consolidate_check(consolidate_one)
69    c1consmod=consolidate_srcs(con_idxs, c1model, obs, obsl, bg_scores,
70        ↵  drec.log_Li, [drec])
71
72    @test consolidate_check(c1consmod.sources)[1]
73    @test c1consmod.log_Li > dmodel.log_Li
74    @test all(c1consmod.mix_matrix[:,1])
75    @test c1consmod.sources[1][1]=log.(src_ATG)
76    @test c1consmod.sources[3][1]≠log.(src_ATG)
77    @test c1consmod.origin=="consolidated c1"
78
79    _,con_idxs=consolidate_check(consolidate_two)
80    c2consmod=consolidate_srcs(con_idxs, c2model, obs, obsl, bg_scores,
81        ↵  drec.log_Li, [drec])

```

```

71
72     @test consolidate_check(c2consmod.sources)[1]
73     @test c2consmod.log_Li > dmodel.log_Li
74     @test all(c2consmod.mix_matrix[:,1])
75     @test c2consmod.sources[1][1]==log.(src_ATG)
76     @test c2consmod.sources[2][1]≠log.(src_ATG)
77     @test c2consmod.sources[3][1]≠log.(src_ATG)
78     @test c2consmod.origin=="consolidated c2"
79
80     rm(dpath)
81 end

```

---

### 17.6.20 /test/ensemble\_tests.jl

```

1 @testset "Ensemble assembly and nested sampling functions" begin
2     ensembledir = randstring()
3     spensembledir = randstring()
4     distdir = randstring()
5
6     source_pwm = [.7 .1 .1 .1
7                   .1 .1 .1 .7
8                   .1 .1 .7 .1]
9
10    source_pwm_2 = [.6 .1 .1 .2
11                      .2 .1 .1 .6
12                      .1 .2 .6 .1]
13
14    src_length_limits=2:12
15    no_sources=3
16
17    source_priors = assemble_source_priors(no_sources, [source_pwm,
18                           ↳ source_pwm_2])
19    mix_prior=.5
20
21    bg_scores = log.(fill(.1, (30,4)))
22
23    ↳ obs=[BioSequences.LongSequence{DNAAlphabet{2}}("CCGTTGACGATGTGATGAATAATGAAAGA",
24
25    ↳ BioSequences.LongSequence{DNAAlphabet{2}}("CCCCGATGATGACCGTTGACCAGATGGATG")
26
27    ↳ BioSequences.LongSequence{DNAAlphabet{2}}("CCCCGATGATGACCCGATTTGAAAAAAA")]

```

```

24     ↵ BioSequences.LongSequence{DNAAlphabet{2}}( "TCATCATGCTGATGATGAATCAGATGAAAG" )
25   ]
26
27   order_seqs = BioBackgroundModels.get_order_n_seqs(obs, 0)
28   coded_seqs = BioBackgroundModels.code_seqs(order_seqs)
29   obs=Array(transpose(coded_seqs))
30
31   ensemble = IPM_Elbow(ensemblendir, 150, source_priors,
32     ↵ (falses(0,0),mix_prior), bg_scores, obs, src_length_limits)
33   ensemble = IPM_Elbow(ensemblendir, 200, source_priors,
34     ↵ (falses(0,0),mix_prior), bg_scores, obs, src_length_limits) #test
35     ↵ resumption
36
37   sp_ensemble = IPM_Elbow(spensemblendir, 200, source_priors,
38     ↵ (falses(0,0),mix_prior), bg_scores, obs, src_length_limits,
39     ↵ posterior_switch=true)
40
41   @test length(ensemble.models) == 200
42   for model in ensemble.models
43     @test -350 < model.log_Li < -150
44   end
45
46   assembler=addprocs(1)
47
48   @everywhere using BioMotifInference
49
50   dist_ensemble=IPM_Elbow(assembler, distdir, 150, source_priors,
51     ↵ (falses(0,0),mix_prior), bg_scores, obs, src_length_limits)
52   dist_ensemble=IPM_Elbow(assembler, distdir, 200, source_priors,
53     ↵ (falses(0,0),mix_prior), bg_scores, obs, src_length_limits) #test
54     ↵ resumption
55
56   @test length(dist_ensemble.models) == 200
57   for model in ensemble.models
58     @test -350 < model.log_Li < -150
59   end

```

```

58     rmprocs(assembler)
59     rm(distdir, recursive=true)

60
61     models_to_permute = 600
62     funclimit=200
63     funcvec=full_perm_funcvec

64
65     instruct = Permute_Instruct(funcvec,
66         → ones(length(funcvec))./length(funcvec),models_to_permute,200,
67         → min_clmps=fill(.02,length(funcvec)))

68     @info "Testing convergence displays..."
69     sp_logZ = converge_ensemble!(sp_ensemble, instruct,
70         → converge_factor=500., wk_disp=true, tuning_disp=true,
71         → ens_disp=true, conv_plot=true, src_disp=true, lh_disp=true,
72         → liwi_disp=true, max_iterates=50)

73     sp_ensemble=reset_ensemble!(sp_ensemble)

74     @info "Testing threaded convergence..."
75     sp_logZ = converge_ensemble!(sp_ensemble, instruct,
76         → converge_factor=500.,wk_disp=false, tuning_disp=false,
77         → ens_disp=false, conv_plot=false, src_disp=false, lh_disp=false,
78         → liwi_disp=false, backup=(true, 150), max_iterates=300)

79     @info "Testing resumption..."
80     sp_logZ = converge_ensemble!(sp_ensemble, instruct,
81         → converge_factor=500.,wk_disp=false, tuning_disp=false,
82         → ens_disp=false, conv_plot=false, src_disp=false, lh_disp=false,
83         → liwi_disp=false, backup=(true, 150))
84     @test length(sp_ensemble.models) == 200
85     @test length(sp_ensemble.log_Li) == length(sp_ensemble.log_Xi) ==
86         length(sp_ensemble.log_wi) == length(sp_ensemble.log_Liwi) ==
87         length(sp_ensemble.log_Zi) == length(sp_ensemble.Hi) ==
88         sp_ensemble.model_counter-200
89     for i in 1:length(sp_ensemble.log_Li)-1
90         @test sp_ensemble.log_Li[i] ≤ sp_ensemble.log_Li[i+1]
91     end
92     for i in 1:length(sp_ensemble.log_Zi)-1
93         @test sp_ensemble.log_Zi[i] ≤ sp_ensemble.log_Zi[i+1]
94     end
95     @test sp_logZ > -1500.0
96

```

```

87  @info "Testing multiprocess convergence ... "
88  @info "Spawning worker pool ... "
89  worker_pool=addprocs(2, topology=:master_worker)
90  @everywhere using BioMotifInference

91
92  #####CONVERGE#####
93  final_logZ = converge_ensemble!(ensemble, instruct, worker_pool,
94      → converge_factor=500., wk_disp=false, tuning_disp=false,
95      → ens_disp=false, conv_plot=false, src_disp=false, lh_disp=false,
96      → liwi_disp=false, backup=(true,500), clean=(true, 500, 1000))

97  #test converging already converged, wih different converge criterion
98  final_logZ = converge_ensemble!(ensemble, instruct, worker_pool,
99      → converge_factor=150., converge_criterion="compression",
100     → wk_disp=false, tuning_disp=false, ens_disp=false,
101     → conv_plot=false, src_disp=false, lh_disp=false, liwi_disp=false,
102     → backup=(true,500), clean=(true, 500, 1000))

103
104  length(ensemble.log_Li)==convits
105
106  rmprocs(worker_pool)

107
108
109
110
111
112
113
114
115
116
117 end

```

---

### 17.6.21 /test/lh\_perf.jl

---

```

1 using BenchmarkTools, BioMotifInference, BioBackgroundModels,
   ↵ BioSequences
2 println(Threads.nthreads())
3 source_pwm = [.7 .1 .1 .1
4 .1 .1 .1 .7
5 .1 .1 .7 .1]
6 source_pwm_2 = [.6 .1 .1 .2
7 .2 .1 .1 .6
8 .1 .2 .6 .1]
9 sources = [(log.(source_pwm), 1),(log.(source_pwm_2), 1)]
10
11 bg_scores = log.(fill(.1, (150,10000)))
12
13 obs=zeros(UInt8,151,10000)
14 obs[1:150,1:end] .= 1
15 obsl=[findfirst(iszero,obs[:,o])-1 for o in 1:size(obs)[2]]
16 mix=trues(10000,2)
17
18 println(median(@benchmark (BioMotifInference.IPM_likelihood($sources,
   ↵ $obs, $obsl, $bg_scores, $mix)) samples=5 evals=20))
19
20 println(median(@benchmark (BioMotifInference.dev_likelihood($sources,
   ↵ $obs, $obsl, $bg_scores, $BitMatrix(transpose(mix)))) samples=5
   ↵ evals=20))

```

---

### 17.6.22 /test/likelihood\_unit\_tests.jl

---

```

1 @testset "Model scoring and likelihood functions" begin
2     #test a trivial scoring example
3     obs=[BioSequences.LongSequence{DNAAlphabet{2}}("AAAAAA")]
4     order_seqs = BioBackgroundModels.get_order_n_seqs(obs, 0)
5     coded_seqs = BioBackgroundModels.code_seqs(order_seqs)
6     obs=Array(transpose(coded_seqs))
7
8     source_pwm = zeros(1,4)
9     source_pwm[1,1] = 1
10    log_pwm = log.(source_pwm)
11
12    source_stop=5
13

```

---

```

14 #score preallocates
15 ss_srcs = [revcomp_pwm(log_pwm)]
16 ds_srcs=[cat(log_pwm,revcomp_pwm(log_pwm),dims=3)]
17 score_mat_ds=zeros(source_stop,2)
18 score_mat_ss=zeros(source_stop,1)
19 score_matrices_ds=Vector{Matrix{Float64}}(undef, 2)
20 score_matrices_ss=Vector{Vector{Float64}}(undef, 2)

21
22 score_sources_ds!(score_mat_ds, score_matrices_ds, ds_srcs, obs[:,1],
23   ↪ source_stop)
23 @test score_matrices_ds[1] = [0.0 -Inf; 0.0 -Inf; 0.0 -Inf; 0.0
24   ↪ -Inf; 0.0 -Inf]

25 score_sources_ss!(score_mat_ss, score_matrices_ss, ss_srcs, obs[:,1],
26   ↪ source_stop)

27 @test score_matrices_ss[1] = [-Inf for i in 1:5]

28
29 #test a more complicated scoring example with two sources
30 obs=[BioSequences.LongSequence{DNAAlphabet{2}}("ATGATGATGATG")]
31 order_seqs = BioBackgroundModels.get_order_n_seqs(obs, 0)
32 coded_seqs = BioBackgroundModels.code_seqs(order_seqs)
33 obs=Array(transpose(coded_seqs))

34
35 source_pwm = [.7 .1 .1 .1
36           .1 .1 .1 .7
37           .1 .1 .7 .1]

38
39 source_pwm_2 = [.6 .1 .1 .2
40           .2 .1 .1 .6
41           .1 .2 .6 .1]

42
43 target_s1 = [.7^3 .1^3; .1^3 .1^3; .1^3 (.1*.7^2); .7^3 .1^3; .1^3
44   ↪ .1^3; .1^3 (.1*.7^2); .7^3 .1^3; .1^3 .1^3; .1^3 (.1*.7^2); .7^3
45   ↪ .1^3]
44 target_s2 = [.6^3 (.2*.1^2); (.2*.1^2) (.2*.1^2); (.2*.1^2)
45   ↪ (.2*.6^2); .6^3 (.2*.1^2); (.2*.1^2) (.2*.1^2); (.2*.1^2)
46   ↪ (.2*.6^2); .6^3 (.2*.1^2)]
46 log_pwms = [log.(source_pwm), log.(source_pwm_2)]

47 source_start = 1
48 source_stop = 10

```

```

49
50 #score preallocates
51 ss_srcs = log_pwms
52 ds_srcs=[cat(pwm,revcomp_pwm(pwm),dims=3) for pwm in log_pwms]
53 score_mat_ds=zeros(source_stop,2)
54 score_mat_ss=zeros(source_stop,1)

55
56 score_sources_ds!(score_mat_ds, score_matrices_ds, ds_srcs, obs[:,1],
57   ↳ [source_stop, source_stop])
58 @test isapprox(score_matrices_ds[1], log.(target_s1))
59 @test isapprox(score_matrices_ds[2], log.(target_s2))

60 score_sources_ss!(score_mat_ss, score_matrices_ss, ss_srcs, obs[:,1],
61   ↳ [source_stop, source_stop])
62 @test isapprox(score_matrices_ss[1], log.(target_s1[:,1]))
63 @test isapprox(score_matrices_ss[2], log.(target_s2[:,1]))

64 #test score weaving and IPM likelihood calculations
65 #trivial example using fake cardinality_penalty
66 target=logaddexp(log(.25), 0.)
67 lh_vec=zeros(2)
68 obsl=1
69 bg_scores=view([log(.25)],:,:)
70 score_mat=[[0.]]
71 osi=[1]
72 source_wmls=[1]
73 lme=0.
74 cardinality_penalty=0. #not correct but for test purposes
75 osi_emit=Vector{Int64}()

76
77 @test weave_scores_ss!(lh_vec, obsl, bg_scores, score_mat, osi,
78   ↳ source_wmls, lme, cardinality_penalty, osi_emit) = target

79 target2=logsumexp([log(.25), log(.5), log(.5)])
80 lh_vec=zeros(2)
81 weavevec=zeros(3)
82 score_mat=[zeros(1,2)]
83 empty!(osi_emit)
84 lme=log(.5)

85
86 @test weave_scores_ds!(weavevec, lh_vec, obsl, bg_scores, score_mat,
87   ↳ osi, source_wmls, lme, cardinality_penalty, osi_emit) = target2

```

```

88 #test more complicated weaves
89 obs=[BioSequences.LongSequence{DNAAlphabet{2}}("ATGATGATGATG")
90 BioSequences.LongSequence{DNAAlphabet{2}}("TGATGATGATGA")]
91 order_seqs = BioBackgroundModels.get_order_n_seqs(obs, 0)
92 coded_seqs = BioBackgroundModels.code_seqs(order_seqs)
93 obs=Array(transpose(coded_seqs))

94
95 sources = [(log.(source_pwm), 1),(log.(source_pwm_2), 1)]
96 dbl1_sources=[(log.(source_pwm), 1),(log.(source_pwm), 1)]

97
98 ds_srcs=[cat(pwm[1],revcomp_pwm(pwm[1])),dims=3) for pwm in sources]
99 dbl1_srcs=[cat(pwm[1],revcomp_pwm(pwm[1])),dims=3) for pwm in
   → dbl1_sources]

100
101 source_stops=[10,10]

102
103 target_o2_s1 = [.1^3 .1^3; .1^3 (.1*.7^2); .7^3 .1^3; .1^3 .1^3
   → (.1*.7^2); .7^3 .1^3; .1^3 .1^3; .1^3 (.1*.7^2); .7^3 .1^3; .1^3
   → .1^3]
104 target_o2_s2 = [(.(2*.1^2) (.(2*.1^2)); (.(2*.1^2) (.(2*.6^2); .6^3
   → (.(2*.1^2); (.(2*.1^2) (.(2*.1^2); (.(2*.1^2) (.(2*.6^2); .6^3
   → (.(2*.1^2); (.(2*.1^2) (.(2*.1^2); (.(2*.1^2) (.(2*.6^2); .6^3
   → (.(2*.1^2); (.(2*.1^2) (.(2*.1^2))]

105
106 score_mat_ds=zeros(source_stop,2)
107 score_matrices_ds=Vector{Matrix{Float64}}(undef, 2)

108
109 score_sources_ds!(score_mat_ds, score_matrices_ds, ds_srcs, obs[:,1],
   → source_stops)

110
111 o1_mats = copy(score_matrices_ds)

112
113 @test isapprox(o1_mats, [log.(target_s1), log.(target_s2)])
114
115 score_sources_ds!(score_mat_ds, score_matrices_ds, ds_srcs, obs[:,2],
   → source_stops)

116
117 o2_mats = copy(score_matrices_ds)

118
119 @test isapprox(o2_mats, [log.(target_o2_s1), log.(target_o2_s2)])
120
121 bg_scores = log.(fill(.5, (12,2)))

```

```

123 log_motif_expectation = log(0.5 / size(bg_scores,1))
124 osi = [1,2]
125 obs_cardinality = length(osi)
126 penalty_sum = sum(exp.(fill(log_motif_expectation,obs_cardinality)))
127 cardinality_penalty=log(1.0-penalty_sum)
128 empty!(osi_emit)
129 source_wmls=[3,3]

130
131 lh_target = -8.87035766177774
132 lh_vec=zeros(13)

133
134 o1_lh = weave_scores_ds!(weavevec, lh_vec, 12, view(bg_scores,:,:1),
135   ↳ o1_mats, osi, source_wmls, log_motif_expectation,
136   ↳ cardinality_penalty, osi_emit)
137 @test isapprox(lh_target,o1_lh)

138
139 o2_lh = weave_scores_ds!(weavevec, lh_vec, 12, view(bg_scores,:,:2),
140   ↳ o2_mats, osi, source_wmls, log_motif_expectation,
141   ↳ cardinality_penalty, osi_emit)

142 lh,cache = IPM_likelihood(sources, obs, [12,12], bg_scores,
143   ↳ trues(2,2), true,true)
144 @test isapprox(o1_lh,cache[1])
145 @test isapprox(o2_lh,cache[2])
146 @test isapprox(lps(o1_lh,o2_lh),lh)

147 #test source penalization
148 score_sources_ds!(score_mat_ds, score_matrices_ds, dbl1_srcs,
149   ↳ obs[:,1], source_stops)

150
151 dbl_score_mat=copy(score_matrices_ds)

152
153 naive=weave_scores_ds!(weavevec, lh_vec, 12,
154   ↳ view(bg_scores,:,:1),Vector{Matrix{Float64}}(), Vector{Int64}(),
155   ↳ Vector{Int64}(), log_motif_expectation, cardinality_penalty,
156   ↳ osi_emit)

157
158 empty!(osi_emit)

```

```

157     single=weave_scores_ds!(weavevec, lh_vec, 12, view(bg_scores,:,1),
158     ↳ [dbl_score_mat[1]], [1], [3], log_motif_expectation,
159     ↳ cardinality_penalty, osi_emit)
160
161     empty!(osi_emit)
162
163     double=weave_scores_ds!(weavevec, lh_vec, 12, view(bg_scores,:,1),
164     ↳ dbl_score_mat, [1,2], [3,3], log_motif_expectation,
165     ↳ cardinality_penalty, osi_emit)
166
167     empty!(osi_emit)
168
169     triple=weave_scores_ds!(weavevec, lh_vec, 12, view(bg_scores,:,1),
170     ↳ [dbl_score_mat[1] for i in 1:3], [1,2,3], [3,3,3],
171     ↳ log_motif_expectation, cardinality_penalty, osi_emit)
172
173     @test (single-naive) > (double-single) > (triple-double)
174
175     naive_target=-16.635532333438686
176     naive = IPM_likelihood(sources, obs, [findfirst(iszero,obs[:,o])-1
177     ↳ for o in 1:size(obs)[2]],
178     ↳ bg_scores,falses(size(obs)[2],length(sources)))
179     @test naive==naive_target
180
181     #test IPM_likelihood clean vector and cache calculations
182     mix_matrix=true(2,2)
183     baselh,basecache = IPM_likelihood(sources, obs, [12,12], bg_scores,
184     ↳ mix_matrix, true,true)
185
186     clean=[true,false]
187
188     unchangedlh,unchangedcache=IPM_likelihood(sources, obs, [12,12],
189     ↳ bg_scores, mix_matrix, true, true, basecache, clean)
190
191     changed_lh,changedcache=IPM_likelihood(sources, obs, [12,12],
192     ↳ bg_scores, BitMatrix([true true; false false]), true, true,
193     ↳ basecache, clean)
194
195     indep_lh, indepcache=IPM_likelihood(sources, obs,[12,12], bg_scores,
196     ↳ BitMatrix([true true; false false]), true, true)
197
198     @test baselh==unchangedlh!=changed_lh==indep_lh
199     @test basecache==unchangedcache!=changedcache==indepcache

```

```

187
188     #check that IPM_likelihood works in single stranded operation
189     IPM_likelihood(sources, obs,[12,12], bg_scores, BitMatrix([true true;
190         ↳ false false]), false)
190 end

```

---

### 17.6.23 /test/mix\_matrix\_unit\_tests.jl

```

1 @testset "Mix matrix initialisation and manipulation functions" begin
2     #test mix matrix init
3     prior_mix_test=init_mix_matrix((trues(2,10),0.0),2, 20)
4     @test all(prior_mix_test[:,1:10])
5     @test !any(prior_mix_test[:,11:20])
6
7     @test sum(init_mix_matrix((falses(0,0),1.0), 0, S)) = 0*S
8     @test sum(init_mix_matrix((falses(0,0),0.0), 0, S)) = 0
9     @test 0 < sum(init_mix_matrix((falses(0,0),0.5), 0, S)) < 0*S
10
11    #test mix matrix decorrelation
12    empty_mixvec=falses(0)
13    one_mix=mixvec_decorrelate(empty_mixvec,1)
14    @test sum(one_mix)==1
15
16    empty_mix = falses(0,S)
17    new_mix,clean=mix_matrix_decorrelate(empty_mix, 500)
18    @test 0 < sum(new_mix) ≤ 500
19    @test !all(clean)
20
21    full_mix = trues(0,S)
22    less_full_mix,clean=mix_matrix_decorrelate(full_mix, 500)
23
24    @test 0*S-sum(less_full_mix) ≤ 500
25    @test !all(clean)
26
27    #test matrix similarity and dissimilarity functions
28    test_mix=falses(0,S)
29    test_mix[1:Int(floor(0/2)),:] *= true
30    test_idx=3
31    compare_mix=deepcopy(test_mix)
32    compare_mix[:,test_idx] *= .!compare_mix[:,test_idx]
33    @test most_dissimilar(test_mix,compare_mix)=test_idx
34

```

```

35     src_mixvec=false(0)
36     src_mixvec[Int(ceil(0/2)):end]:=true
37     @test most_similar(src_mixvec,compare_mix)=test_idx
38 end

```

---

### 17.6.24 /test/permute\_func\_tests.jl

```

1 @testset "Model permutation functions" begin
2     source_pwm = [.7 .1 .1 .1
3                     .1 .1 .1 .7
4                     .1 .1 .7 .1]
5
6     source_pwm_2 = [.6 .1 .1 .2
7                     .2 .1 .1 .6
8                     .1 .2 .6 .1]
9
10    pwm_to_erode = [.25 .25 .25 .25
11                      .97 .01 .01 .01
12                      .01 .01 .01 .97
13                      .01 .01 .97 .01
14                      .25 .25 .25 .25]
15
16    src_length_limits=2:5
17
18    source_priors = assemble_source_priors(3, [source_pwm, source_pwm_2])
19    mix_prior=.5
20
21    bg_scores = log.(fill(.25, (12,2)))
22    obs=[BioSequences.LongSequence{DNAAlphabet{2}}]("ATGATGATGATG")
23    BioSequences.LongSequence{DNAAlphabet{2}}("TGATGATGATGA")]
24    order_seqs = BioBackgroundModels.get_order_n_seqs(obs, 0)
25    coded_seqs = BioBackgroundModels.code_seqs(order_seqs)
26    obs=Array(transpose(coded_seqs))
27    obsl=[findfirst(iszero,obs[:,o])-1 for o in 1:size(obs)[2]]
28
29    test_model = ICA_PWM_Model("test", source_priors,
30                                (false(0,0),mix_prior), bg_scores, obs, src_length_limits)
31
32    ps_model= permute_source(test_model, Vector{Model_Record}(), obs,
33                             obsl, bg_scores, test_model.log_Li, source_priors,
34                             iterates=1000, weight_shift_freq=.2,length_change_freq=.2)
35    @test ps_model.log_Li > test_model.log_Li

```

```

33     @test ps_model.sources ≠ test_model.sources
34     @test ps_model.mix_matrix == test_model.mix_matrix
35     @test ps_model.origin == "PS from test"
36
37     pm_model= permute_mix(test_model, obs, obsl, bg_scores,
38     ↪ test_model.log_Li, iterates=1000)
39     @test pm_model.log_Li > test_model.log_Li
40     @test pm_model.sources == test_model.sources
41     @test pm_model.mix_matrix ≠ test_model.mix_matrix
42     @test "PM from test" == pm_model.origin
43
44     psfm_model=perm_src_fit_mix(test_model, Vector{Model_Record}(),obs,
45     ↪ obsl, bg_scores, test_model.log_Li, source_priors,
46     ↪ iterates=1000)
47     @test psfm_model.log_Li > test_model.log_Li
48     @test psfm_model.sources ≠ test_model.sources
49     @test psfm_model.mix_matrix ≠ test_model.mix_matrix
50     @test "PSFM from test" == psfm_model.origin
51
52     fm_model=fit_mix(test_model, Vector{Model_Record}(),
53     ↪ obs,obsl,bg_scores)
54     @test fm_model.log_Li > test_model.log_Li
55     @test fm_model.sources == test_model.sources
56     @test fm_model.mix_matrix ≠ test_model.mix_matrix
57     @test "FM from test" == fm_model.origin
58     @test fit_mix in fm_model.permute_blacklist
59
60     post_fm_psfm=perm_src_fit_mix(fm_model, Vector{Model_Record}(),obs,
61     ↪ obsl, bg_scores, test_model.log_Li, source_priors, iterates=1000)
62     @test "PSFM from candidate" == post_fm_psfm.origin
63     @test fit_mix in post_fm_psfm.permute_blacklist
64
65     rd_model=random_decorrelate(test_model, Vector{Model_Record}(),obs,
66     ↪ obsl, bg_scores, test_model.log_Li, source_priors, iterates=1000)
67     @test rd_model.log_Li > test_model.log_Li
68     @test rd_model.sources ≠ test_model.sources
69     @test rd_model.mix_matrix ≠ test_model.mix_matrix
70     @test "RD from test" == rd_model.origin
71
72     rs_model=reinit_src(test_model, Vector{Model_Record}(),obs, obsl,
73     ↪ bg_scores, test_model.log_Li, source_priors, iterates=1000)
74     @test rs_model.log_Li > test_model.log_Li
75     @test rs_model.sources ≠ test_model.sources

```

```

69  @test rs_model.mix_matrix ≠ test_model.mix_matrix
70  @test "RS from test" = rs_model.origin
71
72  ↪ erosion_sources=[(log.(source_pwm),1),(log.(source_pwm_2),1),(log.(pwm_to_ero
73
74  eroded_mix=true(2,3)
75
76  erosion_lh=IPM_likelihood(erosion_sources,obs,obsl, bg_scores,
77  ↪ eroded_mix)
78
78  erosion_model=ICA_PWM_Model("erode", "", erosion_sources,
79  ↪ test_model.source_length_limits,eroded_mix, erosion_lh)
80
80  eroded_model=erode_model(erosion_model, Vector{Model_Record}(), obs,
81  ↪ obsl, bg_scores, erosion_model.log_Li)
81  @test eroded_model.log_Li > erosion_model.log_Li
82  @test eroded_model.sources ≠ erosion_model.sources
83  @test eroded_model.mix_matrix = erosion_model.mix_matrix
84  @test "EM from erode" = eroded_model.origin
85  @test eroded_model.sources[1]=erosion_model.sources[1]
86  @test eroded_model.sources[2]=erosion_model.sources[2]
87  @test eroded_model.sources[3]≠erosion_model.sources[3]
88  @test eroded_model.sources[3][1]=erosion_model.sources[3][1][2:4,:]
89
90  dbl_eroded=erode_model(eroded_model, Vector{Model_Record}(), obs,
91  ↪ obsl, bg_scores, eroded_model.log_Li)
91  @test dbl_eroded.permute_blacklist = [erode_model]
92
93
94  merger_srcs=  [(log.([.1 .7 .1 .1
95  .1 .7 .1 .1
96  .15 .35 .35 .15]),
97  1
98  ),
99  (log.([.1 .7 .1 .1
100  .1 .65 .15 .1
101  .1 .7 .1 .1]),
102  1
103  ),
104  (log.([.1 .7 .1 .1
105  .1 .7 .1 .1
106  .1 .7 .1 .1

```

```

107     .7 .1 .1 .1]),
108     1
109 )
110
111 accurate_srcs=[ (log.([0.90 0.06 0.02 0.02; 0.02 0.02 0.02 0.94; 0.02
112   ↪ 0.02 0.94 0.02]), 1),
113 (log.([0.94 0.02 0.02 0.02; 0.02 0.07 0.07 0.84; 0.02 0.02 0.94
114   ↪ 0.02]), 1),
115 (log.([0.95 0.01 0.02 0.02; 0.02 0.02 0.02 0.94; 0.02 0.02 0.94
116   ↪ 0.02]), 1)]
117
118 merger_mix = BitMatrix([true false false
119   true true false])
120
121 merger_base=ICA_PWM_Model("merge", "", merger_srcs,src_length_limits,
122   ↪ merger_mix, IPM_likelihood(merger_srcs,obs,obsl, bg_scores,
123   ↪ merger_mix))
124
125 merger_target=ICA_PWM_Model("target", "", accurate_srcs,
126   ↪ src_length_limits, accurate_mix,
127   ↪ IPM_likelihood(accurate_srcs,obs,obsl, bg_scores, accurate_mix))
128
129 path=randstring()
130 test_record = Model_Record(path, merger_target.log_Li)
131 serialize(path, merger_target)
132
133 dm_model=distance_merge(merger_base, [test_record], obs, obsl,
134   ↪ bg_scores, merger_base.log_Li)
135 @test dm_model.log_Li > merger_base.log_Li
136 @test dm_model.sources ≠ merger_base.sources
137 distance_dict=Dict(1⇒3,2⇒1,3⇒1)
138 @test all([src=merger_base.sources[n] ||
139   ↪ src=merger_target.sources[distance_dict[n]] for (n,src) in
140   ↪ enumerate(dm_model.sources)])
141 @test "DM from merge" == dm_model.origin
142
143 sm_model=similarity_merge(merger_base, [test_record], obs, obsl,
144   ↪ bg_scores, merger_base.log_Li)
145 @test sm_model.log_Li > merger_base.log_Li
146 @test sm_model.sources ≠ merger_base.sources

```

```

139  @test all([src=merger_base.sources[n] ||
140    ↪ src=merger_target.sources[n] for (n,src) in
141    ↪ enumerate(sm_model.sources)])
142  @test "SM from merge" == sm_model.origin
143
144  shuffled_model=shuffle_sources(merger_base, [test_record], obs, obsl,
145    ↪ bg_scores, merger_base.log_Li)
146  @test
147    ↪ sum(shuffled_model.sources.=merger_base.sources)=length(shuffled_model.sour
148  for (s,src) in enumerate(shuffled_model.sources)
149    @test src=merger_base.sources[s] ||
150      ↪ src=merger_target.sources[s]
151  if src == merger_base.sources[s]
152    @test
153      ↪ shuffled_model.mix_matrix[:,s]=merger_base.mix_matrix[:,s]
154  else
155    @test
156      ↪ shuffled_model.mix_matrix[:,s]=merger_target.mix_matrix[:,s]
157  end
158  end
159  @test "SS from merge" == shuffled_model.origin
160
161  acc_base_mix=false(2,1);acc_base_mix[1,1]=true
162  acc_merge_mix=false(2,1);acc_merge_mix[2,1]=true
163
164  acc_base=ICA_PWM_Model("accbase", "",
165    ↪ [merger_srcs[1]],src_length_limits, acc_base_mix,
166    ↪ IPM_likelihood([merger_srcs[1]],obs,obsl, bg_scores,
167    ↪ acc_base_mix))
168  acc_merge=ICA_PWM_Model("accmerge", "",
169    ↪ [accurate_srcs[1]],src_length_limits, acc_merge_mix,
170    ↪ IPM_likelihood([accurate_srcs[1]],obs,obsl, bg_scores,
171    ↪ acc_merge_mix))
172
173  accpath=randstring()
174  acc_record = Model_Record(accpath, acc_merge.log_Li)
175  serialize(accpath, acc_merge)
176
177  acc_model=accumulate_mix(acc_base, [acc_record], obs, obsl,
178    ↪ bg_scores, acc_merge.log_Li)
179  @test acc_model.mix_matrix=true(2,1)
180  @test acc_model.sources == acc_merge.sources
181  @test "AM from accbase" == acc_model.origin

```

```

168
169     testwk=addprocs(1)[1]
170     @everywhere import BioMotifInference
171
172     ddm_model=remotecall_fetch(distance_merge, testwk, merger_base,
173         → [test_record], obs, obsl, bg_scores, merger_base.log_Li,
174         → remote=true)
175     @test ddm_model.log_Li > merger_base.log_Li
176     @test ddm_model.sources ≠ merger_base.sources
177     @test all([src=merger_base.sources[n] ||
178         → src=merger_target.sources[distance_dict[n]] for (n,src) in
179         → enumerate(ddm_model.sources)])
180     @test "DM from merge" == ddm_model.origin
181
182
183     dsm_model=remotecall_fetch(similarity_merge, testwk, merger_base,
184         → [test_record], obs, obsl, bg_scores, merger_base.log_Li,
185         → remote=true)
186     @test dsm_model.log_Li > merger_base.log_Li
187     @test dsm_model.sources ≠ merger_base.sources
188     @test all([src=merger_base.sources[n] ||
189         → src=merger_target.sources[n] for (n,src) in
190         → enumerate(dsm_model.sources)])
191     @test "SM from merge" == dsm_model.origin
192
193
194     dshuffled_model=remotecall_fetch(shuffle_sources, testwk,
195         → merger_base, [test_record], obs, obsl, bg_scores,
196         → merger_base.log_Li, remote=true)
197     @test
198         → sum(dshuffled_model.sources.==merger_base.sources)=length(dshuffled_model.sou
199     for (s,src) in enumerate(dshuffled_model.sources)
200         @test src=merger_base.sources[s] ||
201             → src=merger_target.sources[s]
202         if src == merger_base.sources[s]
203             @test
204                 → dshuffled_model.mix_matrix[:,s]=merger_base.mix_matrix[:,s]
205         else
206             @test
207                 → dshuffled_model.mix_matrix[:,s]=merger_target.mix_matrix[:,s]
208         end
209     end
210     @test "SS from merge" == dshuffled_model.origin
211
212

```

```

196 dacc_model=remotecall_fetch(accumulate_mix, testwk, acc_base,
197   ↳ [acc_record], obs, obsl, bg_scores, acc_merge.log_Li)
198 @test dacc_model.mix_matrix=trues(2,1)
199 @test dacc_model.sources = acc_merge.sources
200 @test "AM from accbase" = dacc_model.origin
201
202 info_base=ICA_PWM_Model("info_base", "", [accurate_srcs[1]],
203   ↳ src_length_limits, accurate_mix[:,1:1],
204   ↳ IPM_likelihood([accurate_srcs[1]],obs,obsl,bg_scores,accurate_mix[:,1:1]))
205
206   ↳ info_model=info_fill(info_base,Vector{Model_Record}(),obs,obsl,bg_scores,info_
207 @test info_model.sources[1][1][1,:] = [0., -Inf, -Inf, -Inf]
208 @test info_model.log_Li > info_base.log_Li
209 @test "IF from info_base" = info_model.origin
210
211 rmprocs(testwk)
212 rm(path)
213 rm(accpath)
214 end

```

---

### 17.6.25 /test/permute\_tuner\_tests.jl

```

1 @testset "Permute Tuner" begin
2   funcvec=[random_decorrelate,fit_mix]
3   instruct=Permute_Instruct(funcvec, [.5,.5],100,100)
4   tuner=Permute_Tuner(instruct)
5   #need to test update_weights functionality
6   #want to supply some fake data to induce a .8 .2 categorical
7   tuner.successes[:,1]=falses(TUNING_MEMORY*instruct.func_limit)
8   tuner.successes[:,2]=falses(TUNING_MEMORY*instruct.func_limit)
9
10  ↳ tuner.successes[1:Int(floor(TUNING_MEMORY*instruct.func_limit*.8)),1]=true
11
12  ↳ tuner.successes[1:Int(floor(TUNING_MEMORY*instruct.func_limit*.2)),2]=true
13  update_weights!(tuner)
14  @test tuner.inst.weights=[.8,.2]    #check clamping
15  tuner.successes[:,1]=falses(TUNING_MEMORY*instruct.func_limit)
16  @test tuner.inst.min_clmps=[.01,.01]
17  @test tuner.inst.max_clmps=[1.,1.]
18  update_weights!(tuner)
19  @test tuner.inst.weights=[.01,1-.01]

```

```

19   #test override
20   instruct=Permute_Instruct(funcvec,
21     → [.5,.5],100,100,override_time=2.,override_weights=[.75,.25])
22   tuner=Permute_Tuner(instruct)
23   tune_weights!(tuner, [(1,1.,1.)])
24   @test tuner.inst.weights=[.5,.5] #no override
25   tuner.time_history=fill(3.,CONVERGENCE_MEMORY)
26   tune_weights!(tuner, [(1,1.,1.)])
27   @test tuner.inst.weights=[.75,.25]

28   #more clamping tests
29   testvec=[0.02693244970132842, 0.031512823560878485,
30     → 0.050111382705776294, 0.6893314065796903, 0.04206537140157707,
31     → 0.02013161919125878, 0.026535815205008566, 0.051758654139053166,
32     → 0.03497967993105292, 0.013999262917669668, 0.01264153466670629]
33   target=[0.02527900179828276, 0.029859375657832827,
34     → 0.04845793480273063, 0.6876779586766446, 0.040411923498531406,
35     → 0.02, 0.02488236730196291, 0.050105206236007505,
36     → 0.03332623202800726, 0.02, 0.02]

37   #high clamp
38   testvec=[.9,.08,.02]
39   clamp_pvec!(testvec, fill(.05,length(testvec)), [.45,.45,.45])
40 end

```

---

### 17.6.26 /test/pwm\_unit\_tests.jl

```

1 @testset "PWM source prior setup, PWM source initialisation and
2   manipulation functions" begin
3   #test dirichlet prior estimation from wm inputs
4   wm_input = [.0 .2 .3 .5; .0 .2 .3 .5]
5   est_dirichlet_vec = estimate_dirichlet_prior_on_wm(wm_input)
6   @test typeof(est_dirichlet_vec) = Vector{Dirichlet{Float64}}
7   for pos in 1:length(est_dirichlet_vec)
8     @test isapprox(est_dirichlet_vec[pos].alpha,
9       → wm_input[pos,:].*PRIOR_WT)

```

```

8      end
9
10     bad_input = wm_input .* 2
11     @test_throws DomainError estimate_dirichlet_prior_on_wm(bad_input)
12
13     wm_input = [.1 .2 .3 .4; .1 .2 .3 .4]
14     est_dirichlet_vec = estimate_dirichlet_prior_on_wm(wm_input)
15     @test typeof(est_dirichlet_vec) == Vector{Dirichlet{Float64}}
16     for pos in 1:length(est_dirichlet_vec)
17         @test est_dirichlet_vec[pos].alpha == wm_input[pos,:].*PRIOR_WT
18     end
19
20     length_range = 2:2
21
22     #test informative/uninformative source prior vector assembly
23     test_priors = assemble_source_priors(2, [wm_input])
24     @test length(test_priors) == 2
25     for pos in 1:length(test_priors[1])
26         @test test_priors[1][pos].alpha == wm_input[pos,:].*PRIOR_WT
27     end
28     @test test_priors[2] == false
29
30     #test source wm initialisation from priors
31     test_sources = init_logPWM_sources(test_priors, length_range)
32     for source in test_sources
33         for pos in 1:size(source[1])[1]
34             @test isprobvec(exp.(source[1][pos,:]))
35         end
36     end
37
38     #test that wm_shift is returning good shifted probvecs
39     rando_dist=Dirichlet([.25,.25,.25,.25])
40     for i in 1:1000
41         wm=rand(rando_dist)
42         new_wm=wm_shift(wm,Weibull(1.5,.1))
43         @test isprobvec(new_wm)
44         @test wm!=new_wm
45     end
46
47     #test that legal new sources are generated by permute_source_weights
48     permuted_weight_sources=deepcopy(test_sources)
49
→     permuted_weight_sources[1]=permute_source_weights(permuted_weight_sources[1],

```

```

50      → permuted_weight_sources[2]=permute_source_weights(permuted_weight_sources[2],
51      @test permuted_weight_sources ≠ test_sources
52      for (s,source) in enumerate(permuted_weight_sources)
53          for pos in 1:size(source[1],1)
54              @test isprobvec(exp.(source[1][pos,:]))
55              @test source[1][pos,:] ≠ test_sources[s][1][pos,:]
56      end
57  end
58
59  #test that get_length_params returns legal length shifts
60  lls=1:10
61  pr=1:5
62  for i in 1:1000
63      srcl=rand(1:10)
64      sign, permute_length=get_length_params(srcl, lls, pr)
65      @test pr[1] ≤ permute_length ≤ pr[end]
66      @test sign=-1 || sign=1
67      @test lls[1] ≤ (srcl+(sign*permute_length)) ≤ lls[end]
68  end
69
70  permuted_length_sources=deepcopy(test_sources)
71
72      → permuted_length_sources[1]=permute_source_length(permuted_length_sources[1],t
73
74      → permuted_length_sources[2]=permute_source_length(permuted_length_sources[2],t
75      for (s, source) in enumerate(permuted_length_sources)
76          @test size(source[1],1) ≠ size(test_sources[1][1],1)
77          @test 1 ≤ size(source[1],1) ≤ 3
78      end
79
80      info_test_wm=[1. 0. 0. 0.
81                  .94 .02 .02 .02
82                  .82 .06 .06 .06
83                  .7 .1 .1 .1
84                  .67 .11 .11 .11
85                  .52 .16 .16 .16
86                  .4 .2 .2 .2
87                  .25 .25 .25 .25]
88
88  #test eroding sources by finding most informational position and
     → cutting off when information drops below threshold
89  infovec=get_pwm_info(log.(info_test_wm))

```

```

89     @test infovec=[2.0, 1.5774573308022544, 1.0346297041419121,
    ↵   0.6432203505529603, 0.5620360019822908, 0.2403724636586433,
    ↵   0.07807190511263773, 0.0]
90
91     erosion_test_source=(log.([.25 .25 .25 .25
92                           .2 .4 .2 .2
93                           .7 .1 .1 .1
94                           .06 .06 .06 .82
95                           .7 .1 .1 .1
96                           .25 .25 .25 .25]),1)
97
98     infovec=get_pwm_info(erosion_test_source[1])
99     start_idx, end_idx = get_erosion_idxs(infovec, .25, 2:8)
100    @test start_idx==3
101    @test end_idx==5
102
103    eroded_pwm,eroded_prior_idx=erode_source(erosion_test_source,2:8,.25)
104    for pos in 1:size(eroded_pwm,1)
105        @test isprobvec(exp.(eroded_pwm[pos,:]))
106    end
107    @test eroded_prior_idx==3
108    @test isapprox(exp.(eroded_pwm), [.7 .1 .1 .1
109                      .06 .06 .06 .82
110                      .7 .1 .1 .1])
111
112
113     #make sure revcomp_pwm is reversing pwms across both dimensions
114     revcomp_test_pwm = zeros(2,4)
115     revcomp_test_pwm[1,1] = 1
116     revcomp_test_pwm[2,3] = 1
117     log_revcomp_test_pwm = log.(revcomp_test_pwm)
118     @test revcomp_pwm(log_revcomp_test_pwm) == [-Inf 0. -Inf -Inf
119                               -Inf -Inf -Inf
120                               ↵   0.]
120 end

```

---

### 17.6.27 /test/runtests.jl

---

```

1 @info "Loading test packages ... "
2
3 using BioMotifInference, BioBackgroundModels, BioSequences,
   ↵ Distributions, Distributed, Random, Serialization, Test

```

```

4 import StatsFuns: logsumexp, logaddexp
5 import BioMotifInference:estimate_dirichlet_prior_on_wm,
→ assemble_source_priors, init_logPWM_sources, wm_shift,
→ permute_source_weights, get_length_params, permute_source_length,
→ get_pwm_info, get_erosion_idxs, erode_source, init_mix_matrix,
→ mixvec_decorrelate, mix_matrix_decorrelate, most_dissimilar,
→ most_similar, revcomp_pwm, score_sources_ds!, score_sources_ss!,
→ weave_scores_ss!, weave_scores_ds!, IPM_likelihood,
→ consolidate_check, consolidate_srcs, pwm_distance, permute_source,
→ permute_mix, perm_src_fit_mix, fit_mix, random_decorrelate,
→ reinit_src, erode_model, reinit_src, shuffle_sources, accumulate_mix,
→ distance_merge, similarity_merge, info_fill, converge_ensemble!,
→ reset_ensemble!, Permute_Tuner, PRIOR_WT, TUNING_MEMORY,
→ CONVERGENCE_MEMORY, tune_weights!, update_weights!, clamp_pvec!
6 import Distances: euclidean
7
8 @info "Beginning tests ... "
9 using Random
10 Random.seed!(786)
11 O=1000;S=50
12
13 include("pwm_unit_tests.jl")
14 include("mix_matrix_unit_tests.jl")
15 include("likelihood_unit_tests.jl")
16 include("consolidate_unit_tests.jl")
17 include("permute_func_tests.jl")
18 include("permute_tuner_tests.jl")
19 include("ensemble_tests.jl")
20 @info "Tests complete!"

```

---

### 17.6.28 /test/spike\_recovery.jl

---

```

1 #Testbed for synthetic spike recovery from example background
2 using BioBackgroundModels, BioMotifInference, Random, Distributed,
→ Distributions, Serialization
3 Random.seed!(786)
4 #CONSTANTS
5 no_obs=500
6 obsl=100:200
7
8 const folder_path="/bench/PhD/NGS_binaries/BBM/refined_folders"
9

```

```

10 report_folders=deserialize(folder_path)

11
12 bhmm_vec=[BHMM([0.40027008799648645, 0.1997894691941005,
    ↵ 0.006556173444055311, 0.3381222117369811, 0.053302496489949405,
    ↵ 0.0019595611384158307], [0.9910385961237116 0.001055084535731191
    ↵ 0.0030117366072612726 0.0025443692116936516 1.875366316590218e-83
    ↵ 0.002350213521605879; 0.003592562092379169 0.6212087435679351
    ↵ 1.4862557155404555e-91 0.3738436265294334 0.001355067810251799
    ↵ 5.674610926019336e-174; 0.03314627345971247 0.00026886477654034587
    ↵ 0.9628685354878737 3.677787936178394e-5 0.00367954839651186 5.0e-324;
    ↵ 0.0002903769304330447 0.3737391779744011 1.584645416437449e-35
    ↵ 0.6225645652084915 0.002839228767872827 0.0005666511188010834;
    ↵ 0.0005590565306708014 0.0064195824547637215 0.004061509889975232
    ↵ 0.01962586251822172 0.9687912312979926 0.0005427573083774576;
    ↵ 0.0174697691356862 2.0648278441425858e-20 8.008498087945918e-218
    ↵ 0.004151185943888868 0.0013251598095032858 0.9770538851109234],
    ↵ [Categorical([0.2565257692018922, 0.24506145230281004,
    ↵ 0.2649485969649606, 0.23346418153033835]),
    ↵ Categorical([0.052491168994042534, 0.0987036876775239,
    ↵ 0.2383241923003866, 0.610480951028046]),
    ↵ Categorical([0.08371833024203595, 0.07204591736665274,
    ↵ 0.3854013598343639, 0.4588343925569475]),
    ↵ Categorical([0.5954709485303519, 0.2497223085607694,
    ↵ 0.09528006039357192, 0.0595266825153071]),
    ↵ Categorical([0.4744103622376032, 0.04038264172760258,
    ↵ 0.04588736323302444, 0.4393196328017719]),
    ↵ Categorical([0.41173384083242826, 0.44270375576246973,
    ↵ 0.030612125665857307, 0.11495027773924743])])

13
14 bhmm_dist=Categorical([1.])

15
16 struc_sig_1=[.1 .7 .1 .1
    ↵ .1 .1 .1 .7
    ↵ .1 .7 .1 .1]
17
18
19 periodicity=8
20 struc_frac_obs=.75
21
22
23 tata_box=[.05 .05 .05 .85
    ↵ .85 .05 .05 .05
    ↵ .05 .05 .05 .85
    ↵ .85 .05 .05 .05
    ↵ .425 .075 .075 .425

```

```

28          .85 .05 .05 .05
29          .425 .075 .075 .425]
30 motif_frac_obs=.7
31 motif_recur_range=1:4
32
33 @info "Constructing synthetic sample set ... "
34 obs1, bg_scores1, hmm_truth1, spike_truth1 =
    → synthetic_sample(no_obs,obs1,bhmm_vec,bhmm_dist,[struc_sig_1,tata_box],[(true,(st
35
36 e1 = "BMISpikeTest"
37
38 #JOB CONSTANTS
39 const ensemble_size = 750
40 const no_sources = 4
41 const source_min_bases = 3
42 const source_max_bases = 12
43 const source_length_range= source_min_bases:source_max_bases
44 const mixing_prior = .5
45 const models_to_permute = ensemble_size * 3
46 funcvec=full_perm_funcvec
47 push!(funcvec, BioMotifInference.permute_source)
48 push!(funcvec, BioMotifInference.permute_source)
49 args=[Vector{Tuple{Symbol,Any}}() for i in 1:length(funcvec)]
50 args[end-1]=[:weight_shift_freq,0.],(:length_change_freq,1.),(:length_perm_range,1:1
51 args[end]=[:weight_shift_freq,.1],(:length_change_freq,0.),(:weight_shift_dist,Unifo
52
53
54 instruct = Permute_Instruct(funcvec,
    → ones(length(funcvec))/length(funcvec),models_to_permute,100;
    → args=args)
55
56 @info "Assembling source priors ... "
57 prior_array= Vector{Matrix{Float64}}()
58 source_priors = BioMotifInference.assemble_source_priors(no_sources,
    → prior_array)
59
60 @info "Assembling ensemble ... "
61 isfile(e1*/ens") ? (ens1 = deserialize(e1*/ens")) :
62     (ens1 = IPM_Eensemle(e1, ensemble_size, source_priors, (falses(0,0),
    → mixing_prior), bg_scores1, obs1, source_length_range))
63
64 @info "Converging ensemble ... "

```

---

```

65 logZ1 = BioMotifInference.converge_ensemble!(ens1, instruct,
→   backup=true,250), wk_disp=false, tuning_disp=false, ens_disp=true,
→   conv_plot=false, src_disp=true, lh_disp=false, liwi_disp=false)

```

---

## 17.7 AWSWrangler

Github repository: <https://github.com/mmattocks/AWSWrangler.jl>

### 17.7.1 /src/AWSWrangler.jl

---

```

1 module AWSWrangler
2
3 using AWSSDK, Dates, Sockets
4
5 function spot_wrangle(no_instances::Integer, spot_price::AbstractFloat,
→   sgdp_name::String, sgdp_desc::String, skeys::String, zone::String,
→   ami::String, instance_type::String)
6   #ssh permissions
7   WANip=chomp(read(pipeline(`dig +short myip.opendns.com
→   @resolver1.opendns.com`), String))
8   permissions = [
9     [
10       "FromPort" => 22,
11       "IpProtocol" => "tcp",
12       "IpRanges" => [
13         [
14           "CidrIp" => WANip*"/32",
15           "Description" => "SSH access"
16         ]
17       ],
18       "ToPort" => 22
19     ]
20   ]
21
22   @info "Configuring AWS credentials ... "
23   aws=aws_config()
24
25   @info "Getting security group info ... "
26   sgdp=Dict()
27
28   try

```

```

29      ↵ sgrp=AWSSDK.EC2.create_security_group(aws,GroupDescription=sgrp_desc,Grou
30      inret=AWSSDK.EC2.authorize_security_group_ingress(aws,
31          ↵ GroupId=sgrp["groupId"], IpPermissions=permissions)
32      outret=AWSSDK.EC2.authorize_security_group_egress(aws,
33          ↵ GroupId=sgrp["groupId"], IpPermissions=permissions)
34  catch error
35      if error.code == "InvalidGroup.Duplicate"
36
37      ↵ sgrp=AWSSDK.EC2.describe_security_groups(aws,GroupName=sgrp_name)[ "se
38  end
39
40  #Launch specs
41 LaunchSpec = [
42      "ImageId" => ami,
43      "InstanceType" => instance_type,
44      "SecurityGroupId" => [
45          sgrp["groupId"]
46      ],
47      "Placement" => [
48          "AvailabilityZone" => zone
49      ],
50      "KeyName" => skeys
51  ]
52
53  @info "Requesting spot instances ... "
54
55  spot_reqs=AWSSDK.EC2.request_spot_instances(aws,
56      ↵ LaunchSpecification=LaunchSpec, InstanceCount=no_instances,
57      ↵ SpotPrice=spot_price, Type="one-time")
58
59  instance_public_ips=Vector()
60
61  @info "Waiting for instances to become active ... "
62
63  if no_instances==1
64
65      ↵ spot_req_Id=spot_reqs["spotInstanceRequestSet"]["item"]["spotInstanceReq
66      active=false
67      while !active

```

```

65         AWSSDK.EC2.describe_spot_instance_requests(aws,
66             → SpotInstanceRequestId=spot_req_Id)[ "spotInstanceRequestSet" ][ "item" ][
67                 → = "active" ? active=true : sleep(10)
68     end
69     instance_Id=AWSSDK.EC2.describe_spot_instance_requests(aws,
70         → SpotInstanceRequestId=spot_req_Id)[ "spotInstanceRequestSet" ][ "item" ][ "ins
71     address=AWSSDK.EC2.describe_instances(aws,
72         → InstanceId=instance_Id)[ "reservationSet" ][ "item" ][ "instancesSet" ][ "item"
73     push!(instance_public_ips,address)
74   else
75     for spot_req in spot_reqs[ "spotInstanceRequestSet" ][ "item" ]
76       spot_req_Id = spot_req[ "spotInstanceRequestId" ]
77       active=false
78       while !active
79         AWSSDK.EC2.describe_spot_instance_requests(aws,
80             → SpotInstanceRequestId=spot_req_Id)[ "spotInstanceRequestSet" ][ "item" ][
81                 → = "active" ? active=true : sleep(10)
82     end
83     instance_Id = AWSSDK.EC2.describe_spot_instance_requests(aws,
84         → SpotInstanceRequestId=spot_req_Id)[ "spotInstanceRequestSet" ][ "item" ][
85     address=AWSSDK.EC2.describe_instances(aws,
86         → InstanceId=instance_Id)[ "reservationSet" ][ "item" ][ "instancesSet" ][ "item"
87     push!(instance_public_ips,address)
88   end
89 end
90
91     return instance_public_ips
92 end
93
94 function get_cheapest_zone(instance_type::String)
95   arguments=[  

96     "ProductDescription"⇒["Linux/UNIX"]  

97     "InstanceType"⇒[instance_type]  

98     "StartTime"⇒Dates.now(Dates.UTC)  

99   ]
100   history=AWSSDK.EC2.describe_spot_price_history(aws_config(),
101     → arguments)
102
103   price_dict=Dict{String,Float64}()  

104   for record in history[ "spotPriceHistorySet" ][ "item" ]
105     price_dict[record[ "availabilityZone" ]]=parse(Float64,
106           → record[ "spotPrice" ])
107 end

```

```

98
99     return sort(collect(price_dict), by=x→x[2])[1]
100 end
101
102 export spot_wrangle,get_cheapest_zone
103
104 end # module

```

---

## 17.8 Analyses

Github repository: <https://github.com/mmattocks/Thesis/Analyses>

### 17.8.1 /cmz/a10correlation.jl

```

1 using
   ↳ BayesianLinearRegression,Csv,DataFrames,Distributions,NGRefTools,StatsBase,Plots,
   ↳ Measurements
2 import Measurements:value,uncertainty
3
4 default(legendfont = (8), guidefont = (10,), tickfont = (8), guide = "x")
5
6 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
7
8 a10df=DataFrame(Csv.read(a10pth))
9 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
   ↳ eachrow(a10df)]
10
11 X=Vector{Float64}()
12 measure_dict=Dict{String,Vector{Vector{Float64}}}()
13
14 measure_dict["PopEst"]=Vector{Vector{Float64}}()
15 measure_dict["VolEst"]=Vector{Vector{Float64}}()
16
17 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
18 for t_df in groupby(a10df, "Time point (d)")
19     push!(X,Float64(unique(t_df."Time point (d)")[1]))
20     pevec,tvec,dvec,vvec,vevec,rtvec,rplvec,rptvec,ldvec,ondvec,
   ↳ onlvec,oplvec,inlvec,iplvec,gclvec=[zeros(0) for i in 1:15]
21
22     for i_df in groupby(t_df,"BSR")
23         length([skipmissing(i_df."Lens diameter")...])>0 &&
   ↳ push!(ldvec,mean(skipmissing(i_df."Lens diameter")))

```

```

24     end
25
26     for i_df in groupby(t_df, "BSR")
27         if length([skipmissing(i_df."CMZ Sum") ... ]) > 0
28             push!(tvec, mean(skipmissing(i_df."CMZ Sum")))
29             if length([skipmissing(i_df."Lens diameter") ... ]) > 0
30                 push!(pevec, mean(skipmissing(i_df."CMZ
31                         → Sum")) * mean(skipmissing(i_df."Lens diameter")) / 14.)
32             else
33                 push!(pevec, mean(skipmissing(i_df."CMZ
34                         → Sum")) * mean(ldvec) / 14.)
35             end
36         end
37         if length([skipmissing(i_df."Retina thickness") ... ]) > 0 &&
38             length([skipmissing(i_df."IPL") ... ]) > 0 &&
39             length([skipmissing(i_df."OPL") ... ]) > 0 &&
40             length([skipmissing(i_df."RPE length") ... ]) > 0
41
42             rthi = mean(skipmissing(i_df."Retina
43                         → thickness")) - mean(skipmissing(i_df."OPL")) - mean(skipmissing(i_df."IPL"))
44             rpel = mean(skipmissing(i_df."RPE length"))
45             if length([skipmissing(i_df."Lens diameter") ... ]) > 0
46                 rcirc = rpel + mean(skipmissing(i_df."Lens diameter"))
47             else
48                 rcirc = rpel + mean(ldvec)
49             end
50
51             or = rcirc / 2π
52             ir = or - .5 * rthi
53
54             ov = (4/3) * π * (or^3)
55             iv = (4/3) * π * (ir^3)
56
57             push!(vevec, (ov - iv) * (4/5))
58         end
59     end
60 end
61
62 x = measure_dict["PopEst"][1]
63 uncorrX3d = ones(length(x), 1)

```

```

64 corrX3d=hcat(ones(length(x)),x)
65 y=measure_dict["VolEst"][1]
66 uncorr=BayesianLinearRegression.fit!(BayesianLinReg(uncorrX3d,y))
67 corr=BayesianLinearRegression.fit!(BayesianLinReg(corrX3d,y))

68
69 function blr_plt(x,y,Xs,blrs,colors,labels,q=.975)
70
    → plt=scatter(x,y,marker=:diamond,markersize=8,markerstrokestyle=:bold,markerco
    → "retinal volume", xlabel="Estimated CMZ Annulus
    → Population", legend=:bottomleft)
71 for (X,blr,color,label) in zip(Xs, blrs,colors,labels)
72     line=zeros(size(X,1))
73     ribbon=zeros(size(X,1))
74     predictions=BayesianLinearRegression.predict(blr,X)
75     for (n,p) in enumerate(predictions)
76         normal=Normal(value(p),uncertainty(p))
77         ribbon[n]=quantile(normal,q)-mean(normal)
78         line[n]=mean((value(p)))
79     end
80     plot!(plt,x,line,ribbon=ribbon,color=color,label=label)
81 end
82 return plt
83 end

84
85 p=blr_plt(x,y,[corrX3d,uncorrX3d],[corr,uncorr],[:darkmagenta,:green],[ "Correlated
    → Model", "Uncorrelated Model"])
86
87 savefig(p, "/bench/PhD/Thesis/images/cmz/3dcorr.png")
88 @info "Saved fig."
89
90 println("\begin{tabular}{l|l|l|l}")
91 println("\hline")
92 println("{\bf Age (dpf)} & {\bf Uncorrelated logZ} & {\bf Correlated
    → logZ} & {\bf logZR}\hline")
93
94 for (x,xs,ys) in zip(X,measure_dict["PopEst"],measure_dict["VolEst"])
95     uncorrX=ones(length(xs),1)
96     corrX=hcat(ones(length(xs)),xs)
97     uncorr=BayesianLinearRegression.fit!(BayesianLinReg(uncorrX,ys))
98     corr=BayesianLinearRegression.fit!(BayesianLinReg(corrX,ys))

```

```

99    println("$x & {\bf $(round(logEvidence(uncorr),digits=3))} &
  ↳ $(round(logEvidence(corr),digits=3)) &
  ↳ $(round(logEvidence(uncorr)-logEvidence(corr),digits=3))\\\ \
  ↳ \\\hline")
100 end
101
102 println("\\end{tabular}")
103
104
105

```

---

### 17.8.2 /cmz/a10decayslicediagnostic.jl

```

1 using
  ↳ BayesianLinearRegression, CSV, DataFrames, Distributions, CMZNicheSims, GMC_NS,
  ↳ Serialization, Plots
2
3 emultipath="/bench/PhD/NGS_binaries/CNS/A10/emulti_decay"
4
5 default(legendfont = (10), guidefont = (12), tickfont = (10))
6
7 base_scale=[144.,5.]
8 base_min=[10.,0.]
9 time_scale=90
10 time_min=4.
11
12 prior=[LogNormal(log(20),log(2)),Uniform(eps(),.03),LogNormal(log(.9),log(1.6))]
13
14
15 em=deserialize(emultipath*"/ens")
16
17 models=["132.93","674.18","575.46","745.16"]
18 MLE=deserialize(em.path*"/132.93").log_Li
19 X=em.constants[1]
20 plotticks=[10*i for i in 1:9]
21
22 mapplots=Vector{Plots.Plot}()
23 mapxs=Vector{Float64}()
24
25 for model in models
26     mapm=deserialize(em.path*"/$model")
27

```

```

28 catdobs=vcat([em.obs[1][t] for t in 1:length(X)]...)
29 catvobs=vcat([em.obs[2][t] for t in 1:length(X)]...)

30
31 dXs=vcat([[X[n] for i in 1:length(em.obs[1][n])] for n in
32             ↳ 1:length(X)]...)
33 vXs=vcat([[X[n] for i in 1:length(em.obs[2][n])] for n in
34             ↳ 1:length(X)]...)

35
36 md_mean=mapm.slices[1].disp_mat[:,2]
37 md_upper=mapm.slices[1].disp_mat[:,3]-mapm.slices[1].disp_mat[:,2]
38 md_lower=mapm.slices[1].disp_mat[:,2]-mapm.slices[1].disp_mat[:,1]

39
40 mv_mean=mapm.slices[2].disp_mat[:,2]
41 mv_upper=mapm.slices[2].disp_mat[:,3]-mapm.slices[2].disp_mat[:,2]
42 mv_lower=mapm.slices[2].disp_mat[:,2]-mapm.slices[2].disp_mat[:,1]

43
44 mapmd=plt=scatter(dXs,catdobs, marker=:cross, color=:black,
45                     ↳ markersize=3, showaxis=:y, xformatter=_→" ", xticks=X,
46                     ↳ legend=nothing, title=model, ylabel="Population")
47 plot!(mapmd=plt, X, md_mean, ribbon=(md_lower,md_upper),
48       ↳ color=:green)

49
50 mapmv=plt=scatter(vXs,catvobs, marker=:cross, color=:black,
51                     ↳ markersize=3, xlabel="Population", xticks=X, xlabel="Age (dpf)",
52                     ↳ legend=nothing)
53 plot!(mapmv=plt, X, mv_mean, ribbon=(mv_lower,mv_upper),
54       ↳ color=:darkmagenta)

55
56 combined_map=Plots.plot(mapmd=plt, mapmv=plt, layout=grid(2,1),
57                           ↳ size=(300,600), link=:xy)

58
59 push!(mapplots, combined_map)
60 push!(mapxs, mapm.θ[1])
61
62 end

63
64 mkdes=posterior_kde(em, bivar=[1⇒2,1⇒3,2⇒3])

65
66 ctkmarg=contour(mkdes[4].x,mkdes[4].y, transpose(mkdes[4].density),
67                   ↳ c=:viridis, xlabel="iCT (hr)", ylabel="K", colorbar=:false,
68                   ↳ xlims=[0,144], ylims=[0,.03], levels=200, xticks=24.:24.:144.)

```

```

58 ctemarg=contour(mkdes[5].x,mkdes[5].y,transpose(mkdes[5].density),
  ↵ c=:viridis, xlabel="ICT (hr)", ylabel="ε", colorbar=:false,
  ↵ xlims=[0,144], ylims=[0,5.], levels=200, xticks=24.:24.:144.)
59
60 kemarg=contour(mkdes[6].x,mkdes[6].y,transpose(mkdes[6].density),
  ↵ c=:viridis, xlabel="κ", ylabel="ε", colorbar=:false, xlims=[0,.03],
  ↵ ylims=[0,5.], levels=200, xticks=24.:24.:144.)
61
62 # clim=(0,.5), legend=(.8,.3)
63
64 ymarg=map(x→pdf(prior[1][1],x),mkdes[1].x)
65
66 ctmarg=plot(mkdes[1].x, ymarg, color=:darkmagenta, fill=true,
  ↵ fillalpha=.5,xlims=[0,144], xlabel="Initial Cycle Time (hr)",
  ↵ ylabel="Probability density", label="Prior")
67 plot!(mkdes[1].x,mkdes[1].density,color=:green, fill=true, fillalpha=.5,
  ↵ label="Combined Posterior")
68
69 colors=[:darkgreen,:darkred,:darkblue,:darkmagenta]
70 for (n, model) in enumerate(models)
71
  ↵ plot!([mapxs[n],mapxs[n]],[0,maximum(mkdes[1].density)],color=colors[n],
  ↵ label=model)
72 end
73
74 combined_layout=@layout[ctmarg{.3h}
  ↵ grid(1,4)
  ↵ ]
75
76
77
78 pmpplot=Plots.plot(ctmarg,mapplots...,layout=combined_layout,size=(1200,900))
79
80 savefig(pmpplot, "/bench/PhD/Defense/images/polymodality.png")

```

---

### 17.8.3 /cmz/a10dvdecayslice.jl

---

```

1 using
  ↵ BayesianLinearRegression,CSV,DataFrames,Distributions,CMZNicheSims,GMC_NS,
  ↵ Serialization, Plots
2 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
3
4 edpth="/bench/PhD/NGS_binaries/CNS/A10/ed_decay"
5 evpth="/bench/PhD/NGS_binaries/CNS/A10/ev_decay"

```

```

6 empth="/bench/PhD/NGS_binaries/CNS/A10/emulti_decay"
7 paths=[edpth,evpth,emph]
8
9 default(legendfont = (10), guidefont = (12), tickfont = (10))
10
11 a10df=DataFrame(CSV.read(a10pth))
12 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
13   eachrow(a10df)]
14 X=Vector{Float64}()
15 measure_dict=Dict{String,Vector{Vector{Float64}}}()
16
17 measure_dict["Dorsal CMZ (#)"]=Vector{Vector{Float64}}()
18 measure_dict["Ventral CMZ (#)"]=Vector{Vector{Float64}}()
19 measure_dict["Lens circumference"]=Vector{Vector{Float64}}()
20
21 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
22 for t_df in groupby(a10df, "Time point (d)")
23     if !(Float64(unique(t_df."Time point (d)")[1]) in [180., 360.])
24
25         push!(X,Float64(unique(t_df."Time point (d)")[1]))
26         dvec,vvec,lcirc = [zeros(0) for i in 1:4]
27
28         for i_df in groupby(t_df,"BSR")
29             length([skipmissing(i_df."Lens diameter") ... ]) > 0 &&
29                 push!(lcirc,mean(skipmissing(i_df."Lens diameter"))*π)
30         end
31
32         for i_df in groupby(t_df,"BSR")
33
34             length([skipmissing(i_df."Dorsal CMZ (#)") ... ]) > 0 &&
34                 mean(skipmissing(i_df."Dorsal CMZ (#)")) > 0. &&
34                 push!(dvec,mean(skipmissing(i_df."Dorsal CMZ (#)")))
35             length([skipmissing(i_df."Ventral CMZ (#)") ... ]) > 0 &&
35                 mean(skipmissing(i_df."Ventral CMZ (#)")) > 0. &&
35                 push!(vvec,mean(skipmissing(i_df."Ventral CMZ (#)")))
36         end
37
38         push!(measure_dict["Dorsal CMZ (#)"],dvec)
39         push!(measure_dict["Ventral CMZ (#)"],vvec)
40         push!(measure_dict["Lens circumference"],lcirc)
41     end
42 end

```

```

43
44 dobs=measure_dict["Dorsal CMZ (#)"]
45 vobs=measure_dict["Ventral CMZ (#)"]
46 mobs=[measure_dict["Dorsal CMZ (#)"],measure_dict["Ventral CMZ (#)"]]
47
48 obsset=[dobs,vobs,mobs]
49
50 dpopdist=fit(LogNormal,measure_dict["Dorsal CMZ (#)"][[1]])
51 vpopdist=fit(LogNormal,measure_dict["Ventral CMZ (#)"][[1]])
52
53 logLC=vcat([log.(measure_dict["Lens circumference"][[n]]) for n in
54   ↪ 1:length(X)] ... )
55 logX=vcat([[log.(X[n]) for ms in measure_dict["Lens circumference"][[n]]]
56   ↪ for n in 1:length(X)] ... )
57 logX=hcat(ones(length(logX)),logX)
58
59 lm=BayesianLinearRegression.fit!(BayesianLinReg(logX,logLC))
60 w1,w2=posteriorWeights(lm)
61 factor=10^w1.val
62 power=w2.val
63
64 mc_its=Int64(1e6)
65
66 ed_constants=[X,dpopdist,lm,mc_its]
67 ev_constants=[X,vpopdist,lm,mc_its]
68 em_constants=[X,[dpopdist,vpopdist],lm,mc_its]
69 constants=[ed_constants,ev_constants,em_constants]
70
71 priors=[LogNormal(log(20),log(2)),Uniform(eps(),.03),LogNormal(log(.9),log(1.6))]
72
73 box=[10. 144.
74       eps() .03
75       0. 5.]
76 box=GMC_NS.to_unit_ball.(box,priors)
77
78 uds=Vector{Vector{Function}}({{[],[liwi_display],[convergence_display]}])
79 lds=Vector{Vector{Function}}({{[model_obs_display],[ensemble_display],[ensemble_displa
80
81 gmc_settings=GMC_DEFAULTS
82 gmc_settings[end]=100
83

```

```

84 for (pth,constants,obs) in zip(paths,constants,obsset)
85     if isfile(pth*"/ens")
86         e=deserialize(pth*"/ens")
87     else
88         if !occursin("emulti",pth)
89             e=Decay_Ensemble(pth,1000,obs, priors, constants, box,
90                               ↪ gmc_settings)
91         else
92             e=MultiSlice_Decay_Ensemble(pth,1000,obs, priors, constants,
93                               ↪ box, gmc_settings)
94         end
95     end
96 end
97
98 ed=deserialize(edpth*"/ens")
99 ev=deserialize(evpth*"/ens")
100 em=deserialize(empth*"/ens")
101
102 edev=measure_evidence(ed)
103 evev=measure_evidence(ev)
104 emev=measure_evidence(em)
105
106 evidence_ratio=emev-(edev+evev)
107
108 println("\\"begin{tabular}{|l|l|l|l|l|l|}")
109 println("\\"hline")
110 println("{\\bf Combined D/V model logZ} & {\\bf Separate D/V model logZ}
111   & {\\bf logZR} & {$\\sigma$ Significance}\\\\ \\hline")
112 println("\\textbf{${(round(emev,digits=3))} & ${round(edev+evev,digits=3)}
113   & ${round(evidence_ratio,digits=3)} &
114   & ${round(evidence_ratio.val/evidence_ratio.err,digits=3)}\\\\
115 \\hline")
116 println("\\"end{tabular}")
117
118 mapd=deserialize(ed.models[findmax([m.log_Li for m in
119   ↪ ed.models])[2]].path)
120 mapv=deserialize(ev.models[findmax([m.log_Li for m in
121   ↪ ev.models])[2]].path)

```

```

116 mapm=deserialize(em.models[findmax([m.log_Li for m in
117   ↳ em.models])[2]].path)
118
119 println("\begin{tabular}{l|l|l|l}")
120 println("{\bf Parameter} & {\bf Combined MAP} & {\bf Dorsal MAP} &
121   {\bf Ventral MAP}\hline")
122 println("$CT_{i}$(h) & $(round(mapm.θ[1], digits=1)) &
123   $(round(mapd.θ[1], digits=1)) & $(round(mapv.θ[1], digits=1))\\\\
124   \hline")
125 println("$\\kappa$ decay& $(round(mapm.θ[2], digits=4)) &
126   $(round(mapd.θ[2], digits=4)) & $(round(mapv.θ[2], digits=4))\\\\
127   \hline")
128
129 X=ed.constants[1]
130
131 plotticks=[10*i for i in 1:9]
132
133 catdobs=vcat([ed.obs[t] for t in 1:length(X)] ... )
134 catvobs=vcat([ev.obs[t] for t in 1:length(X)] ... )
135
136 dXs=vcat([[X[n] for i in 1:length(ed.obs[n])] for n in 1:length(X)] ... )
137 vXs=vcat([[X[n] for i in 1:length(ev.obs[n])] for n in 1:length(X)] ... )
138 dvXs=vcat([[X[n] for i in 1:length(em.obs[1][n])] for n in
139   1:length(X)] ... )
140
141 d_mean=mapd.disp_mat[:,2]
142 d_upper=mapd.disp_mat[:,3]-mapd.disp_mat[:,2]
143 d_lower=mapd.disp_mat[:,2]-mapd.disp_mat[:,1]
144
145 v_mean=mapv.disp_mat[:,2]
146 v_upper=mapv.disp_mat[:,3]-mapv.disp_mat[:,2]
147 v_lower=mapv.disp_mat[:,2]-mapv.disp_mat[:,1]
148
149 md_mean=mapm.slices[1].disp_mat[:,2]
150 md_upper=mapm.slices[1].disp_mat[:,3]-mapm.slices[1].disp_mat[:,2]
151 md_lower=mapm.slices[1].disp_mat[:,2]-mapm.slices[1].disp_mat[:,1]
152
153 mv_mean=mapm.slices[2].disp_mat[:,2]
154 mv_upper=mapm.slices[2].disp_mat[:,3]-mapm.slices[2].disp_mat[:,2]
155 mv_lower=mapm.slices[2].disp_mat[:,2]-mapm.slices[2].disp_mat[:,1]

```

```

150 mv_lower=mapm.slices[2].disp_mat[:,2]--mapm.slices[2].disp_mat[:,1]
151
152 mapmd=plt.scatter(dXs,catdobs, marker=:cross, color=:black, markersize=3,
    ↵ label="Dorsal CMZ population data", showaxis=:y, xformatter=_→"",
    ↵ xticks=X, ylabel="Population")
153 plot!(mapmd=plt, X, md_mean, ribbon=(md_lower,md_upper), color=:green,
    ↵ label="Combined model D population")
154 annotate!([(8,140,Plots.text("A",18))])
155
156 mapmv=plt.scatter(vXs,catvobs, marker=:cross, color=:black, markersize=3,
    ↵ label="Ventral CMZ population data", showaxis=:y,
    ↵ ylabel="Population", xformatter=_→"",xticks=X)
157 plot!(mapmv=plt, X, mv_mean, ribbon=(mv_lower,mv_upper),
    ↵ color=:darkmagenta, label="Combined model V population")
158 annotate!([(8,120,Plots.text("B",18))])
159
160 mapd=plt.scatter(dXs,catdobs, marker=:cross, color=:black, markersize=3,
    ↵ label="Dorsal CMZ population data", xlabel="Age (dpf)", xticks=X,
    ↵ ylabel="Population")
161 plot!(mapd=plt, X, d_mean, ribbon=(d_lower,d_upper), color=:green,
    ↵ label="Separate model D population")
162 annotate!([(8,125,Plots.text("C",18))])
163
164 mapv=plt.scatter(vXs,catvobs, marker=:cross, color=:black, markersize=3,
    ↵ label="Ventral CMZ population data", xlabel="Age (dpf)",
    ↵ ylabel="Population", xticks=X)
165 plot!(mapv=plt, X, v_mean, ribbon=(v_lower,v_upper), color=:darkmagenta,
    ↵ label="Separate model V population")
166 annotate!([(8,140,Plots.text("D",18))])
167
168 combined_map=Plots.plot(mapmd=plt, mapmv=plt, mapd=plt, mapv=plt, layout=grid(2,2),
    ↵ size=(1200,900), link=:x)
169
170 savefig(combined_map, "/bench/PhD/Thesis/images/cmz/a10dvdecayMAP.png")
171
172 dkdes=posterior_kde(ed)
173 vkdes=posterior_kde(ev)
174 mkdes=posterior_kde(em)
175
176 ymarg=map(x→pdf(priors[1],x),mkdes[1].x)
177

```

```

178 ctmarg=plot(mkdes[1].x, ymarg, color=:darkmagenta, fill=true,
    ↵ fillalpha=.5,xlims=[0,144], xlabel="Initial Cycle Time (hr)",
    ↵ ylabel="Probability density", label="Prior")
179 plot!(mkdes[1].x,mkdes[1].density,color=:green, fill=true, fillalpha=.5,
    ↵ label="Combined Posterior")
180 plot!(dkdes[1].x,dkdes[1].density,color=:orange, fill=true,
    ↵ fillalpha=.25, label="Dorsal Posterior")
181 plot!(vkdes[1].x,vkdes[1].density,color=:blue, fill=true, fillalpha=.25,
    ↵ label="Ventral Posterior")
182 plot!([mapm.θ[1],mapm.θ[1]],[0,maximum(mkdes[1].density)],color=:darkgreen,
    ↵ label="Combined MAP")
183 plot!([mapd.θ[1],mapd.θ[1]],[0,maximum(dkdes[1].density)],color=:darkred,
    ↵ label="Dorsal MAP")
184 plot!([mapv.θ[1],mapv.θ[1]],[0,maximum(vkdes[1].density)],color=:darkblue,
    ↵ label="Ventral MAP")
185 annotate!([(0,.06,Plots.text("A",18))])

186
187 ymarg=map(x→pdf(priors[2],x),mkdes[2].x)
188 kmarg=plot(mkdes[2].x, ymarg, color=:darkmagenta, fill=true,
    ↵ fillalpha=.5, xlims=[0,.03], xlabel="Decay constant κ",
    ↵ ylabel="Probability density", label="Prior")
189 plot!(mkdes[2].x,mkdes[2].density,color=:green, fill=true, fillalpha=.5,
    ↵ label="Combined Posterior")
190 plot!(dkdes[2].x,dkdes[2].density,color=:orange, fill=true,
    ↵ fillalpha=.25, label="Dorsal Posterior")
191 plot!(vkdes[2].x,vkdes[2].density,color=:blue, fill=true, fillalpha=.25,
    ↵ label="Ventral Posterior")
192 plot!([mapm.θ[2],mapm.θ[2]],[0,maximum(mkdes[2].density)],color=:darkgreen,
    ↵ label="Combined MAP")
193 plot!([mapd.θ[2],mapd.θ[2]],[0,maximum(dkdes[2].density)],color=:darkred,
    ↵ label="Dorsal MAP")
194 plot!([mapv.θ[2],mapv.θ[2]],[0,maximum(vkdes[2].density)],color=:darkblue,
    ↵ label="Ventral MAP")
195 annotate!([(0,225,Plots.text("B",18))])

196
197 ymarg=map(x→pdf(priors[3],x),mkdes[3].x)
198 ermarg=plot(mkdes[3].x, ymarg, color=:darkmagenta, fill=true,
    ↵ fillalpha=.5, xlims=[0,4], xlabel="Niche exit rate ε",
    ↵ ylabel="Probability density", label="Prior")
199 plot!(mkdes[3].x,mkdes[3].density,color=:green, fill=true, fillalpha=.5,
    ↵ label="Combined Posterior")
200 plot!(dkdes[3].x,dkdes[3].density,color=:orange, fill=true,
    ↵ fillalpha=.25, label="Dorsal Posterior")

```

```

201 plot!(vkdes[3].x,vkdes[3].density,color=:blue, fill=true, fillalpha=.25,
    ↵ label="Ventral Posterior")
202 plot!([mapm.θ[3],mapm.θ[3]],[0,maximum(mkdes[3].density)],color=:darkgreen,
    ↵ label="Combined MAP")
203 plot!([mapd.θ[3],mapd.θ[3]],[0,maximum(dkdes[3].density)],color=:darkred,
    ↵ label="Dorsal MAP")
204 plot!([mapv.θ[3],mapv.θ[3]],[0,maximum(vkdes[3].density)],color=:darkblue,
    ↵ label="Ventral MAP")
205 annotate!([(0,1.2,Plots.text("C",18))])
206
207
208 ph1c=plot(ctmarg,kmarg,ermarg, layout=grid(3,1),size=(1000,1200))
209 savefig(ph1c,"/bench/PhD/Thesis/images/cmz/a10decaymarg.png")

```

---

#### 17.8.4 /cmz/a10dvratio.jl

```

1 using BayesianLinearRegression,CSV,DataFrames,Distributions, GMC_NS,
    ↵ Serialization, Plots, NGRefTools
2 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
3
4 a10df=DataFrame(CSV.read(a10pth))
5 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
    ↵ eachrow(a10df)]
6
7 X=Vector{Float64}()
8 measure_dict=Dict{String,Vector{Vector{Float64}}}()
9
10 measure_dict["Dorsal CMZ (#)"]=Vector{Vector{Float64}}()
11 measure_dict["Ventral CMZ (#)"]=Vector{Vector{Float64}}()
12 measure_dict["Ratio"]=Vector{Vector{Float64}}()
13
14 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
15 for t_df in groupby(a10df, "Time point (d)")
16     push!(X,Float64(unique(t_df."Time point (d)")[1]))
17     rvec,dvec,vvec = [zeros(0) for i in 1:3]
18
19     for i_df in groupby(t_df,"BSR")
20         length(skipmissing(i_df."Dorsal CMZ (#)")...)]>0 &&
            ↵ mean(skipmissing(i_df."Dorsal CMZ (#)")) >0. &&
            ↵ push!(dvec,mean(skipmissing(i_df."Dorsal CMZ (#)")))

```

```

21      length(skipmissing(i_df."Ventral CMZ (#)")...])>0 &&
22          mean(skipmissing(i_df."Ventral CMZ (#)")) >0. &&
23          push!(vvec,mean(skipmissing(i_df."Ventral CMZ (#)")))
24      length(skipmissing(i_df."Dorsal CMZ (#)")...])>0 &&
25          mean(skipmissing(i_df."Dorsal CMZ (#)")) >0. &&
26          length(skipmissing(i_df."Ventral CMZ (#)")...])>0 &&
27          mean(skipmissing(i_df."Ventral CMZ (#)")) >0. &&
28          push!(rvec,mean(skipmissing(i_df."Dorsal CMZ
29              (#)"))/mean(skipmissing(i_df."Ventral CMZ (#)")))
30      end
31
32      push!(measure_dict["Ratio"],rvec)
33      push!(measure_dict["Dorsal CMZ (#)"],dvec)
34      push!(measure_dict["Ventral CMZ (#)"],vvec)
35  end
36
37  d_logn_mts=[fit(MarginalTDist,log.(measure_dict["Dorsal CMZ (#)"])[n]))
38      for n in 1:length(X)]
39  d_mean=[exp(mean(mt)) for mt in d_logn_mts]
40  d_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in d_logn_mts]
41  d_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in d_logn_mts]
42
43  v_logn_mts=[fit(MarginalTDist,log.(measure_dict["Ventral CMZ (#)"])[n]))
44      for n in 1:length(X)]
45  v_mean=[exp(mean(mt)) for mt in v_logn_mts]
46  v_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in v_logn_mts]
47  v_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in v_logn_mts]
48
49  r_n_mts=[fit(MarginalTDist,measure_dict["Ratio"])[n]) for n in
50      1:length(X)]
51  r_mean=[mean(mt) for mt in r_n_mts]
52  r_lower=[mean(mt)-quantile(mt,.025) for mt in r_n_mts]
53  r_upper=[quantile(mt,.975)-mean(mt) for mt in r_n_mts]
54
55  plotticks=[30*i for i in 1:12]
56
57  dv_chart=Plots.plot(X, d_mean, ribbon=(d_lower,d_upper), color=:green,
58      label="Dorsal mean", ylabel="CMZ sectional population size",
59      showaxis=:y, xticks=plotticks, xformatter=_→"")
60  plot!(X, v_mean, ribbon=(v_lower,v_upper), color=:darkmagenta,
61      label="Ventral mean")

```

```

49 scatter!(vcat([[X[n] for i in 1:length(measure_dict["Dorsal CMZ
    ↵ (#)"])[n]]] for n in 1:length(X)]...), vcat(measure_dict["Dorsal CMZ
    ↵ (#)"]...), marker=:utriangle, color=:green, markersize=3,
    ↵ markerstrokecolor=:green, label="Dorsal data")
50 scatter!(vcat([[X[n] for i in 1:length(measure_dict["Ventral CMZ
    ↵ (#)"])[n]]] for n in 1:length(X)]...), vcat(measure_dict["Ventral CMZ
    ↵ (#)"]...), marker=:dtriangle, color=:darkmagenta, markersize=3,
    ↵ markerstrokecolor=:darkmagenta, label="Ventral data")
51 annotate!([(1.5, 125, Plots.text("A", 18))])
52 lens!([2, 31], [10, 150], inset = (1, bbox(0.3, 0.1, 0.45, 0.6)))
53
54 ratio_chart=scatter(vcat([[X[n] for i in
    ↵ 1:length(measure_dict["Ratio"])[n]]] for n in
    ↵ 1:length(X)]...), vcat(measure_dict["Ratio"]...), marker=:cross,
    ↵ color=:black, markersize=3, markerstrokecolor=:black, label="Ratio
    ↵ data", ylabel="Individual asymmetry ratio", xlabel="Age (dpf)",
    ↵ xticks=plotticks)
55 plot!(X, r_mean, ribbon=(r_lower,r_upper), color=:green, label="Ratio
    ↵ mean")
56 plot!(X, [1. for i in 1:length(X)], style=:dot, color=:black, label="Even
    ↵ ratio")
57 annotate!([(1.5,.5,Plots.text("B",18))])
58
59
60 l=@layout [a{.6h}
    ↵ b]
61
62
63 combined=Plots.plot(dv_chart,ratio_chart,layout=l, link=:x,
    ↵ size=(900,600))
64
65 savefig(combined, "/bench/PhD/Thesis/images/cmz/DVontology.png")

```

---

### 17.8.5 /cmz/a10dvslice.jl

---

```

1 using
    ↵ BayesianLinearRegression, CSV, DataFrames, Distributions, CMZNicheSims, GMC_NS,
    ↵ Serialization, Plots
2 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
3
4 edpath="/bench/PhD/NGS_binaries/CNS/A10/ed"
5 evpath="/bench/PhD/NGS_binaries/CNS/A10/ev"
6 emultipath="/bench/PhD/NGS_binaries/CNS/A10/emulti"

```

```

7 paths=[edpath,evpath,emultipath]
8
9 default(legendfont = (10), guidefont = (12), tickfont = (10))
10
11 a10df=DataFrame(CSV.read(a10pth))
12 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
13   eachrow(a10df)]
14 X=Vector{Float64}()
15 measure_dict=Dict{String,Vector{Vector{Float64}}}()
16
17 measure_dict["Dorsal CMZ (#)"]=Vector{Vector{Float64}}()
18 measure_dict["Ventral CMZ (#)"]=Vector{Vector{Float64}}()
19 measure_dict["Lens circumference"]=Vector{Vector{Float64}}()
20
21 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
22 for t_df in groupby(a10df, "Time point (d)")
23   if !(Float64(unique(t_df."Time point (d)")[1]) in [180., 360.])
24
25     push!(X,Float64(unique(t_df."Time point (d)")[1]))
26     dvec,vvec,lcirc = [zeros(0) for i in 1:4]
27
28     for i_df in groupby(t_df,"BSR")
29       length([skipmissing(i_df."Lens diameter")...])>0 &&
29         → push!(lcirc,mean(skipmissing(i_df."Lens diameter"))*π)
30     end
31
32     for i_df in groupby(t_df,"BSR")
33
34       length([skipmissing(i_df."Dorsal CMZ (#)")...])>0 &&
34         → mean(skipmissing(i_df."Dorsal CMZ (#)")) >0. &&
34         → push!(dvec,mean(skipmissing(i_df."Dorsal CMZ (#)")))
35       length([skipmissing(i_df."Ventral CMZ (#)")...])>0 &&
35         → mean(skipmissing(i_df."Ventral CMZ (#)")) >0. &&
35         → push!(vvec,mean(skipmissing(i_df."Ventral CMZ (#)")))
36     end
37
38     push!(measure_dict["Dorsal CMZ (#)"],dvec)
39     push!(measure_dict["Ventral CMZ (#)"],vvec)
40     push!(measure_dict["Lens circumference"],lcirc)
41   end
42 end
43

```

```

44 obsset=[measure_dict["Dorsal CMZ (#)"],measure_dict["Ventral CMZ
45   (#)"],[measure_dict["Dorsal CMZ (#)"],measure_dict["Ventral CMZ
46   (#)"]]]
47
48
49 logLC=vcat([log.(measure_dict["Lens circumference"])[n]) for n in
50   1:length(X)] ... )
51 logX=vcat([[log.(X[n]) for ms in measure_dict["Lens circumference"]][n]
52   for n in 1:length(X)] ... )
53 logX=hcat(ones(length(logX)),logX)
54
55 lm=BayesianLinearRegression.fit!(BayesianLinReg(logX,logLC))
56 w1,w2=posteriorWeights(lm)
57 factor=10^w1.val
58 power=w2.val
59
60 lm=Lens_Model(factor,power,14.)
61
62 mc_its=Int64(1e6)
63 base_scale=[144.,5.]
64 base_min=[10.,0.]
65 time_scale=90
66 time_min=4.
67
68 cycle_prior=[LogNormal(log(20),log(2)),LogNormal(log(.9),log(1.6))]
69 end_prior=[Uniform(4,90)]
70
71 function compose_priors(phases)
72     prs=Vector{Vector{Distribution}}(){}
73     bxes=Vector{Matrix{Float64}}(){}
74     for p in phases
75         pr=[repeat(cycle_prior,p) ... ,repeat(end_prior,p-1) ... ]
76         push!(prs,pr)
77         bx=hcat([repeat(base_min,p) ... ,fill(time_min,p-1) ... ],
78                 [repeat(base_scale,p) ... ,fill(time_scale,p-1) ... ])
79         push!(bxes,GMC_NS.to_unit_ball.(bx,pr))
80     end
81     return prs,bxes
82 end
83
84 prior,box=compose_priors(2)

```

```

83
84 d_constants=[X,dpopdist,lm,mc_its,2]
85 v_constants=[X,vpopdist,lm,mc_its,2]
86 multi_constants=[X,[dpopdist,vpopdist],lm,mc_its,2]
87 constants=[d_constants,v_constants,multi_constants]
88
89 uds=Vector{Vector{Function}}([[[],[liwi_display],[convergence_display]]])
90 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display],[ensemble_displa
91
92 gmc_settings=GMC_DEFAULTS
93 gmc_settings[end]=100
94
95 for (pth,constants,obs) in zip(paths,constants,obsset)
96     if isfile(pth*"/ens")
97         e=deserialize(pth*"/ens")
98     else
99         if !occursin("emulti",pth)
100            e=Slice_Ensemble(pth,1000,obs, prior..., constants, box..., ,
101                           gmc_settings)
102        else
103            e=MultiSlice_Ensemble(pth,1000,obs, prior..., constants,
104                                   box..., gmc_settings)
105        end
106    end
107
108 converge_ensemble!(e,backup=true,20),upper_displays=uds,
109   ↵ lower_displays=lds, disp_rot_its=50, mc_noise=.3,
110   ↵ converge_factor=1e-6)
111 end
112
113 ed=deserialize(edpath*"/ens")
114 ev=deserialize(epath*"/ens")
115 em=deserialize(emultipath*"/ens")
116
117 edev=measure_evidence(ed)
118 evev=measure_evidence(ev)
119 emev=measure_evidence(em)
120
121 evidence_ratio=emev-(edev+evev)
122
123 println("\\begin{tabular}{l|l|l|l|l|l}")
124 println("\\hline")

```

```

121 println("{\\bf Combined D/V model logZ} & {\\bf Separate D/V model logZ}
    & {\\bf logZR} & {$\\sigma\$ Significance}\\\\ \\hline")
122 println("\\textbf{${(round(emev,digits=3))} & ${round(edev+evev,digits=3)}
    & ${round(evidence_ratio,digits=3)} &
    ${round(evidence_ratio.val/evidence_ratio.err,digits=3)})\\\\
    \\hline")
123 println("\\end{tabular}")
124
125 mapd=deserialize(ed.models[findmax([m.log_Li for m in
    ed.models])[2]].path)
126 mapv=deserialize(ev.models[findmax([m.log_Li for m in
    ev.models])[2]].path)
127 mapm=deserialize(em.models[findmax([m.log_Li for m in
    em.models])[2]].path)
128
129 println("\\begin{tabular}{l|l|l|l|l}")
130 println("\\hline")
131 println("{\\bf Parameter} & {\\bf Total MAP} & {\\bf Dorsal MAP} & {\\bf
    Ventral MAP}\\\\ \\hline")
132 println("Phase 1 $CT\$ (h) & ${round(mapm.θ[1], digits=1)} &
    ${round(mapd.θ[1], digits=1)} & ${round(mapv.θ[1], digits=1)})\\\\
    \\hline")
133 println("Phase 1 $\\epsilon\$ & ${round(mapm.θ[2], digits=2)} &
    ${round(mapd.θ[2], digits=2)} & ${round(mapv.θ[2], digits=2)})\\\\
    \\hline")
134 println("Phase 2 $CT\$ (h) & ${round(mapm.θ[3], digits=1)} &
    ${round(mapd.θ[3], digits=1)} & ${round(mapv.θ[3], digits=1)})\\\\
    \\hline")
135 println("Phase 2 $\\epsilon\$ & ${round(mapm.θ[4], digits=2)} &
    ${round(mapd.θ[4], digits=2)} & ${round(mapv.θ[4], digits=2)})\\\\
    \\hline")
136 println("Transition age & ${round(mapm.θ[5], digits=1)} &
    ${round(mapd.θ[5], digits=1)} & ${round(mapv.θ[5], digits=1)})\\\\
    \\hline")
137 println("\\end{tabular}")
138
139 X=ed.constants[1]
140
141 plotticks=[10*i for i in 1:9]
142
143 catdobs=vcat([ed.obs[t] for t in 1:length(X)]...)
144 catvobs=vcat([ev.obs[t] for t in 1:length(X)]...)
145

```

```

146 dXs=vcat([[X[n] for i in 1:length(ed.obs[n])] for n in 1:length(X)] ...)
147 vXs=vcat([[X[n] for i in 1:length(ev.obs[n])] for n in 1:length(X)] ...)
148 dvXs=vcat([[X[n] for i in 1:length(em.obs[1][n])] for n in
   ↵ 1:length(X)] ...)

149
150 d_mean=mapd.disp_mat[:,2]
151 d_upper=mapd.disp_mat[:,3]-mapd.disp_mat[:,2]
152 d_lower=mapd.disp_mat[:,2]-mapd.disp_mat[:,1]

153
154 v_mean=mapv.disp_mat[:,2]
155 v_upper=mapv.disp_mat[:,3]-mapv.disp_mat[:,2]
156 v_lower=mapv.disp_mat[:,2]-mapv.disp_mat[:,1]

157
158 md_mean=mapm.slices[1].disp_mat[:,2]
159 md_upper=mapm.slices[1].disp_mat[:,3]-mapm.slices[1].disp_mat[:,2]
160 md_lower=mapm.slices[1].disp_mat[:,2]-mapm.slices[1].disp_mat[:,1]

161
162 mv_mean=mapm.slices[2].disp_mat[:,2]
163 mv_upper=mapm.slices[2].disp_mat[:,3]-mapm.slices[2].disp_mat[:,2]
164 mv_lower=mapm.slices[2].disp_mat[:,2]-mapm.slices[2].disp_mat[:,1]

165
166 mapmd_plt=scatter(dXs,catdobs, marker=:cross, color=:black, markersize=3,
   ↵ label="Dorsal CMZ population data", showaxis=:y, xformatter=_→"",
   ↵ xticks=X, ylabel="Population")
167 plot!(mapmd_plt, X, md_mean, ribbon=(md_lower,md_upper), color=:green,
   ↵ label="Combined model D population")
168 annotate!([(8,200,Plots.text("A",18))])

169
170 mapmv_plt=scatter(vXs,catvobs, marker=:cross, color=:black, markersize=3,
   ↵ label="Ventral CMZ population data", showaxis=:y,
   ↵ ylabel="Population", xformatter=_→"",xticks=X)
171 plot!(mapmv_plt, X, mv_mean, ribbon=(mv_lower,mv_upper),
   ↵ color=:darkmagenta, label="Combined model V population")
172 annotate!([(8,145,Plots.text("B",18))])

173
174 mapd_plt=scatter(dXs,catdobs, marker=:cross, color=:black, markersize=3,
   ↵ label="Dorsal CMZ population data", xlabel="Age (dpf)", xticks=X,
   ↵ ylabel="Population")
175 plot!(mapd_plt, X, d_mean, ribbon=(d_lower,d_upper), color=:green,
   ↵ label="Separate model D population")
176 annotate!([(8,150,Plots.text("C",18))])

177

```

```

178 mapv_plt=scatter(vXs,catvobs, marker=:cross, color=:black, markersize=3,
    ↵ label="Ventral CMZ population data", xlabel="Age (dpf)",
    ↵ ylabel="Population", xticks=X)
179 plot!(mapv_plt, X, v_mean, ribbon=(v_lower,v_upper), color=:darkmagenta,
    ↵ label="Separate model V population")
180 annotate!([(8,250,Plots.text("D",18))])

181
182 combined_map=Plots.plot(mapmd_plt, mapmv_plt, mapd_plt, mapv_plt, layout=grid(2,2),
    ↵ size=(1200,900), link=:x)

183
184 savefig(combined_map, "/bench/PhD/Thesis/images/cmz/a10dvMAP.png")
185
186 ###KDE FIG
187 kdes=posterior_kde(em,bivar=[1⇒2,3⇒4])
188
189 balx=10.:1.:144.
190 baly=[2^(24/ct)-1 for ct in balx]
191
192 ph1marg=contour(kdes[6].x,kdes[6].y,transpose(kdes[6].density),
    ↵ c=:viridis, xlabel="Phase 1 CT (hr)", ylabel="Phase 1 ε rate",
    ↵ colorbar=false, xlims=[0,144], ylims=[0,5], levels=200, clim=(0,.5),
    ↵ legend=(.8,.3), xticks=24.:24.:144.)
193 plot!(balx,baly,color=:black,linestyle=:dash,label="Balanced pop.")
194 scatter!([mapm.θ[1]],[mapm.θ[2]], marker=:+, markersize=15,
    ↵ markerstrokestyle=:bold, color=:magenta, label="MAP")
195 for i in 1:5
    scatter!([mapm.θ[1]],[mapm.θ[2]], marker=:+, markersize=15,
        ↵ markerstrokestyle=:bold, color=:magenta, label=:none,
        ↵ framestyle=:origin)
196 end
197 lens!([15, 45], [.35, 1.6], inset = (1, bbox(0.2, 0.05, 0.75, 0.55)),
    ↵ colorbar=false, framestyle=:box, subplot=2, xformatter=_→"",
    ↵ yformatter=_→"")
198
199 annotate!([(4,4.5,Plots.text("A",18))])
200
201
202 ymarg=map(x→pdf(cycle_prior[1],x),kdes[1].x)
203 ph1ctmarg=plot(kdes[1].x, ymarg, color=:darkmagenta, fill=true,
204 fillalpha=.5,xlims=[0,144], ylabel="p",xaxis=false, xformatter=_→"",
    ↵ legend=:none)
205 plot!(kdes[1].x,kdes[1].density,color=:green, fill=true, fillalpha=.5)
206 plot!([mapm.θ[1],mapm.θ[1]],[0,maximum(kdes[1].density)],color=:black,linewidth=1)
207 yflip!(ph1ctmarg)

```

```

208
209 ymarg=map(x→pdf(cycle_prior[2],x),kdes[2].x)
210 ph1ermarg=plot(ymarg,kdes[2].x, color=:darkmagenta, fill=true,
    ↵ fillalpha=.5, ylims=[0,5], legend=:none, yaxis=false,
    ↵ yformatter=_→" ", xlabel="p")
211 plot!(kdes[2].density,kdes[2].x,color=:green, fill=true, fillalpha=.5)
212 plot!([0,maximum(kdes[2].density)],[mapm.θ[2],mapm.θ[2]],color=:black,linewidth=1)
213 xflip!(ph1ermarg)
214
215 marglayout=@layout [ymarg{.1w} bivar{.8h}; empty xmarg]
216
217 cbar=heatmap(transpose(ones(101,1).*(0:0.005:.5)), c=:viridis,
    ↵ legend=:none, yticks=:none, xticks=(1:20:101,
    ↵ string.(0:0.1:.5)), xlabel="Probability density")
218
219 empty=plot([],fill=true,xaxis=false,yaxis=false,grid=false,
    ↵ color=:darkmagenta, xformatter=_→" ", yformatter=_→" ",
    ↵ label="Prior")
220 plot!([],color=:green,fill=true,label="Posterior")
221 plot!([], color=:black, linewidth=2, label="MAP")
222 ph1c=plot(ph1ermarg,ph1marg,empty,ph1ctmarg,layout=marglayout,
    ↵ link=:both)
223
224 ph2marg=contour(kdes[7].x,kdes[7].y,transpose(kdes[7].density),
    ↵ c=:viridis, xlabel="Phase 2 CT (hr)", ylabel="Phase 2 ε rate",
    ↵ colorbar=:false, xlims=[0,144], ylims=[0,5], levels=200, clim=(0,.5),
    ↵ legend=(.8,.3), xticks=24.:24.:144.)
225 plot!(balx,baly,color=:black,linestyle=:dash,label="Balanced pop.")
226 scatter!([mapm.θ[3]],[mapm.θ[4]], marker=:+, markersize=15,
    ↵ color=:magenta, label="MAP")
227 for i in 1:5
228     scatter!([mapm.θ[3]],[mapm.θ[4]], marker=:+, markersize=15,
        ↵ color=:magenta, label=:none, framestyle=:origin)
229 end
230 lens!([15, 45], [.35, 1.6], inset = (1, bbox(0.2, 0.05, 0.75, 0.55)),
    ↵ colorbar=:false, framestyle=:box, subplot=2, xformatter=_→" ",
    ↵ yformatter=_→" ")
231 annotate!(ph2marg,[(4,4.5,Plots.text("B",18))])
232
233
234 ymarg=map(x→pdf(cycle_prior[1],x),kdes[1].x)
235 ph2ctmarg=plot(kdes[1].x, ymarg, color=:darkmagenta, fill=true,

```

```

236 fillalpha=.5, ylabel="p", xlims=[0,144], xaxis=false, xformatter=_→"",
  ↵ legend=:none)
237 plot!(kdes[3].x,kdes[3].density,color=:green, fill=true, fillalpha=.5)
238 plot!([mapm.θ[3],mapm.θ[3]],[0,maximum(kdes[3].density)],color=:black,linewidth=1)
239 yflip!(ph2ctmarg)

240
241 ymarg=map(x→pdf(cycle_prior[2],x),kdes[4].x)
242 ph2ermarg=plot(ymarg,kdes[4].x, color=:darkmagenta, fill=true,
  ↵ fillalpha=.5, ylims=[0,5], legend=:none, yaxis=false,
  ↵ yformatter=_→"", xlabel="p")
243 plot!(kdes[4].density,kdes[4].x,color=:green, fill=true, fillalpha=.5)
244 plot!([0,maximum(kdes[4].density)],[mapm.θ[4],mapm.θ[4]],color=:black,linewidth=1)
245 xflip!(ph2ermarg)

246
247 marglayout2=@layout [ymarg{.1w} bivar{.8h}; empty xmarg]
248
249 ph2c=plot(ph2ermarg,ph2marg,deepcopy(empty),ph2ctmarg,layout=marglayout2,
  ↵ link=:both)

250
251 pend=plot(end_prior[1], color=:darkmagenta, fill=true, fillalpha=.5,
  ↵ xlims=[0,90], ylabel="Density", xlabel="Phase transition age",
  ↵ xticks=X, label="Prior", legend=:none)
252 plot!(kdes[5].x, kdes[5].density, color=:green, fill=true, fillalpha=.5,
  ↵ label="Posterior")
253 plot!([mapm.θ[5],mapm.θ[5]],[0,maximum(kdes[5].density)],color=:black,
  ↵ label="MAP")
254 annotate!([(1,.015,Plots.text("C",18))])

255
256 combined_layout=@layout[ph1{.45h}
  ↵ cbar{.01h}
  ↵ ph2{.45h}
  ↵ pend{.09h}]
257
258
259
260
261 combined_marg=plot(ph1c,cbar,ph2c,pend,layout=combined_layout,size=(1000,1200))
262
263 savefig(combined_marg, "/bench/PhD/Thesis/images/cmz/a10dvmarginals.png")

```

---

### 17.8.6 /cmz/a10morphology.jl

---

```

1 using CSV,DataFrames,Distributions,NGRefTools,StatsBase,Plots
2 gr()
3 default(fontsize = 10, guidefont = 11, tickfont = 10)

```

```

4
5 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
6
7 a10df=DataFrame(CSV.read(a10pth))
8 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
9   eachrow(a10df)]
10 X=Vector{Float64}(){}
11 measure_dict=Dict{String,Vector{Vector{Float64}}}(){}
12
13 measure_dict["Retina thickness"]=Vector{Vector{Float64}}(){}
14 measure_dict["RPE length"]=Vector{Vector{Float64}}(){}
15 measure_dict["RPE thickness"]=Vector{Vector{Float64}}(){}
16 measure_dict["Lens diameter"]=Vector{Vector{Float64}}(){}
17 measure_dict["ON diameter"]=Vector{Vector{Float64}}(){}
18 measure_dict["ONL"]=Vector{Vector{Float64}}(){}
19 measure_dict["OPL"]=Vector{Vector{Float64}}(){}
20 measure_dict["INL"]=Vector{Vector{Float64}}(){}
21 measure_dict["IPL"]=Vector{Vector{Float64}}(){}
22 measure_dict["GCL"]=Vector{Vector{Float64}}(){}
23
24 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
25 for t_df in groupby(a10df, "Time point (d)")
26     push!(X,Float64(unique(t_df."Time point (d)")[1]))
27     rtvec, rplvec, rptvec, ldvec, ondvec, onlvec, oplvec, inlvec, iplvec,
28       → gclvec = [zeros(0) for i in 1:10]
29
30     for i_df in groupby(t_df,"BSR")
31         length([skipmissing(i_df."Lens diameter") ... ])>0 &&
32           → push!(ldvec,mean(skipmissing(i_df."Lens diameter")))
33
34         length([skipmissing(i_df."Retina thickness") ... ])>0 &&
35           → push!(rtvec,mean(skipmissing(i_df."Retina thickness")))
36         length([skipmissing(i_df."RPE length") ... ])>0 &&
37           → push!(rplvec,mean(skipmissing(i_df."RPE length")))
38         length([skipmissing(i_df."RPE thickness") ... ])>0 &&
39           → push!(rptvec,mean(skipmissing(i_df."RPE thickness")))
40         length([skipmissing(i_df."ON diameter") ... ])>0 &&
41           → push!(ondvec,mean(skipmissing(i_df."ON diameter")))
42         length([skipmissing(i_df.ONL) ... ])>0 &&
43           → push!(onlvec,mean(skipmissing(i_df.ONL)))
44         length([skipmissing(i_df.OPL) ... ])>0 &&
45           → push!(oplvec,mean(skipmissing(i_df.OPL)))

```

```

38     length([skipmissing(i_df.INL) ... ])>0 &&
39         ↵ push!(inlvec,mean(skipmissing(i_df.INL)))
40     length([skipmissing(i_df.IPL) ... ])>0 &&
41         ↵ push!(iplvec,mean(skipmissing(i_df.IPL)))
42     length([skipmissing(i_df.GCL) ... ])>0 &&
43         ↵ push!(gclvec,mean(skipmissing(i_df.GCL)))
44 end
45
46 push!(measure_dict["Retina thickness"],rtvec)
47 push!(measure_dict["RPE length"],rplvec)
48 push!(measure_dict["RPE thickness"],rptvec)
49 push!(measure_dict["Lens diameter"],ldvec)
50 push!(measure_dict["ON diameter"],ondvec)
51 push!(measure_dict["ONL"],onlvec)
52 push!(measure_dict["OPL"],oplvec)
53 push!(measure_dict["INL"],inlvec)
54 push!(measure_dict["IPL"],iplvec)
55 push!(measure_dict["GCL"],gclvec)
56 end
57
58 #TEST IF DATA IS BETTER MODELLED NORMALLY OR LOGNORMALLY
59 for ms in ["Retina thickness", "RPE length", "RPE thickness", "Lens
60     ↵ diameter", "ON diameter", "ONL", "OPL", "INL", "IPL", "GCL"]
61     ↵ ms=="ON diameter" ? (idxs=length(X)-1) : (idxs=length(X))
62
63     normal_mts=[fit(Normal,measure_dict[ms][n]) for n in 1:idxs]
64     logn_mts=[fit(LogNormal,filter!(i→!iszero(i),measure_dict[ms][n]))
65     ↵ for n in 1:idxs]
66
67     normal_lh=sum([sum(logpdf(normal_mts[n],measure_dict[ms][n])) for n
68     ↵ in 1:idxs])
69     logn_lh=sum([sum(logpdf(logn_mts[n],measure_dict[ms][n])) for n in
70     ↵ 1:idxs])
71
72     println("$ms & $(round(normal_lh,digits=3)) &
73     ↵ $(round(logn_lh,digits=3)) &
74     ↵ $(round(logn_lh-normal_lh,digits=3))\\\\ \\hline")
75 end
76
77 plotticks=[30*i for i in 1:12]
78
79 rt_n_mts=[fit(MarginalTDist,measure_dict["Retina thickness"])[n] for n in
80     ↵ 1:length(X)]

```

```

71 rtn_mean=[mean(mt) for mt in rt_n_mts]
72 rtn_lower=[mean(mt)-quantile(mt,.025) for mt in rt_n_mts]
73 rtn_upper=[quantile(mt,.975)-mean(mt) for mt in rt_n_mts]
74
75 rtn_chart=scatter(vcat([[X[n] for i in 1:length(measure_dict["Retina
    ↵ thickness"])[n]]) for n in 1:length(X)] ... ), vcat(measure_dict["Retina
    ↵ thickness"] ... ), marker=:cross, color=:black, markersize=3,
    ↵ label="Retinal thic. data", ylabel="Retinal thickness (μm)",
    ↵ showaxis=:y, xticks=plotticks, xformatter=_→ "", legend=:bottomright)
76 plot!(X, rtn_mean, ribbon=(rtn_lower,rtn_upper), color=:darkmagenta,
    ↵ label="Mean thickness")
77 annotate!([(8,200,Plots.text("A",18))])
78
79 rpl_n_mts=[fit(MarginalTDist,measure_dict["RPE length"])[n]) for n in
    ↵ 1:length(X)]
80 rpln_mean=[mean(mt) for mt in rpl_n_mts]
81 rpln_lower=[mean(mt)-quantile(mt,.025) for mt in rpl_n_mts]
82 rpln_upper=[quantile(mt,.975)-mean(mt) for mt in rpl_n_mts]
83
84 rpln_chart=scatter(vcat([[X[n] for i in 1:length(measure_dict["RPE
    ↵ length"])[n]]) for n in 1:length(X)] ... ), vcat(measure_dict["RPE
    ↵ length"] ... ), marker=:cross, color=:black, markersize=3, label="RPE
    ↵ length data", ylabel="RPE length (μm)", showaxis=:y,
    ↵ xticks=plotticks, xformatter=_→ "", legend=:bottomright)
85 plot!(X, rpln_mean, ribbon=(rpln_lower,rpln_upper), color=:grey,
    ↵ label="Mean thickness")
86 annotate!([(8,3000,Plots.text("B",18))])
87
88 ld_n_mts=[fit(MarginalTDist,measure_dict["Lens diameter"])[n]) for n in
    ↵ 1:length(X)]
89 ldn_mean=[mean(mt) for mt in ld_n_mts]
90 ldn_mean[end]=ld_n_mts[end].μ
91 ldn_mean[end-2]=ld_n_mts[end-2].μ
92 ldn_lower=[mean(mt)-quantile(mt,.025) for mt in ld_n_mts]
93 ldn_lower[end]=ld_n_mts[end].μ - (ld_n_mts[end].μ-3*ld_n_mts[end].σ)
94 ldn_lower[end-2]=ld_n_mts[end-2].μ -
    ↵ (ld_n_mts[end-2].μ-3*ld_n_mts[end-2].σ)
95 ldn_upper=[quantile(mt,.975)-mean(mt) for mt in ld_n_mts]
96 ldn_upper[end]=(ld_n_mts[end].μ+3*ld_n_mts[end].σ) - ld_n_mts[end].μ
97 ldn_upper[end-2]=(ld_n_mts[end-2].μ+3*ld_n_mts[end-2].σ) -
    ↵ ld_n_mts[end-2].μ
98

```

```

99 ldn_chart=scatter(vcat([[X[n] for i in 1:length(measure_dict["Lens
→   diameter"])[n]] for n in 1:length(X)] ... ),vcat(measure_dict["Lens
→   diameter"] ... ), marker=:cross, color=:black, markersize=3,
→   label="Lens dia. data", ylabel="Lens diameter ( $\mu\text{m}$ )", showaxis=:y,
→   xticks=plotticks, xformatter=_→"", legend=:topleft)
100 plot!(X, ldn_mean, ribbon=(ldn_lower,ldn_upper), color=:darkgreen,
→   label="Diameter mean")
101 annotate!([(8,600,Plots.text("C",18))])
102
103 on_n_mts=[fit(MarginalTDist,measure_dict["ON diameter"])[n]) for n in
→   1:length(X)-1]
104 onn_mean=[mean(mt) for mt in on_n_mts]
105 onn_mean[end]=on_n_mts[end].μ
106 onn_lower=[mean(mt)-quantile(mt,.025) for mt in on_n_mts]
107 onn_lower[end]=on_n_mts[end].μ - (on_n_mts[end].μ-3*on_n_mts[end].σ)
108 onn_upper=[quantile(mt,.975)-mean(mt) for mt in on_n_mts]
109 onn_upper[end]=(on_n_mts[end].μ+3*on_n_mts[end].σ) - on_n_mts[end].μ
110
111 onn_chart=scatter(vcat([[X[n] for i in 1:length(measure_dict["ON
→   diameter"])[n]] for n in 1:length(X)-1] ... ),vcat(measure_dict["ON
→   diameter"] ... ), marker=:cross, color=:black, markersize=3, label="ON
→   dia. data", ylabel="Optic nerve diameter ( $\mu\text{m}$ )", showaxis=:y,
→   xticks=plotticks, xformatter=_→"", legend=:topleft,
→   ylims=[0,200],xlims=[0,360])
112 plot!(X[1:end-1], onn_mean, ribbon=(onn_lower,onn_upper),
→   color=:darkblue, label="Diameter mean")
113 annotate!([(8,105,Plots.text("D",18))])
114 lens!([2, 31], [0, 30], inset = (1, bbox(0.55, 0.1, .45, .8)))
115
116 stackdata=hcat(mean.(measure_dict["RPE
→   thickness"]),mean.(measure_dict["ONL"]),mean.(measure_dict["OPL"]),mean.(measure_
117
118 layers=areaplot(X, stackdata, xticks=plotticks, legend=:none, xlabel="Age
→   (dpf)", ylabel="Layer thickness ( $\mu\text{m}$ )")
119 plot!([100,100],[220,240], color=:black)
120 annotate!([(100,250,Plots.text("GCL",12))])
121 plot!([130,130],[195,240], color=:black)
122 annotate!([(130,250,Plots.text("IPL",12))])
123 plot!([160,160],[130,240], color=:black)
124 annotate!([(160,250,Plots.text("INL",12))])
125 plot!([190,190],[100,240], color=:black)
126 annotate!([(190,250,Plots.text("OPL",12))])
127 plot!([220,220],[75,240], color=:black)

```

```

128 annotate!([(220,250,Plots.text("ONL",12))])
129 plot!([250,250],[25,240], color=:black)
130 annotate!([(250,250,Plots.text("RPE",12))])
131 annotate!([(8.,175,Plots.text("E",18))])
132
133 combined=plot(rtn_chart, rpln_chart, ldn_chart, onn_chart, layers,
    ↪ layout=grid(5,1), size=(1000,1200), link=:x)
134 savefig(combined, "/bench/PhD/Thesis/images/cmz/morphology.png")

```

---

### 17.8.7 /cmz/a10nvln.jl

```

1 using
    ↪ CSV,DataFrames,Distributions,NGRefTools,StatsBase,Plots,GMC_NS,Serialization,
    ↪ Measurements
2
3 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
4 pne_pth="/bench/PhD/NGS_binaries/GMC_NS/A10/pop_norm"
5 plne_pth="/bench/PhD/NGS_binaries/GMC_NS/A10/pop_lognorm"
6 vne_pth="/bench/PhD/NGS_binaries/GMC_NS/A10/vol_norm"
7 vlne_pth="/bench/PhD/NGS_binaries/GMC_NS/A10/vol_lognorm"
8
9 a10df=DataFrame(CSV.read(a10pth))
10 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
    ↪ eachrow(a10df)]
11
12 X=Vector{Float64}(){}
13 measure_dict=Dict{String,Vector{Vector{Float64}}}(){}
14
15 measure_dict["PopEst"]=Vector{Vector{Float64}}(){}
16 measure_dict["VolEst"]=Vector{Vector{Float64}}(){}
17
18 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
19 for t_df in groupby(a10df, "Time point (d)")
20     push!(X,Float64(unique(t_df."Time point (d)")[1]))
21     pevec,ldvec,spvec = [zeros(0) for i in 1:3]
22
23     for i_df in groupby(t_df,"BSR")
24         length(skipmissing(i_df."Lens diameter")...)]>0 &&
            ↪ push!(ldvec,mean(skipmissing(i_df."Lens diameter")))
25     end
26
27     for i_df in groupby(t_df,"BSR")

```

```

28     if length(skipmissing(i_df."CMZ Sum")...])>0 &&
29         mean(skipmissing(i_df."CMZ Sum")) > 0
30             if length(skipmissing(i_df."Lens diameter")...])>0
31                 push!(pevec,mean(skipmissing(i_df."CMZ
32                     → Sum"))*mean(skipmissing(i_df."Lens diameter"))/14.)
33             else
34                 push!(pevec,mean(skipmissing(i_df."CMZ
35                     → Sum"))*mean(ldvec)/14.)
36             end
37         end
38     if length(skipmissing(i_df."Retina thickness")...])>0 &&
39         length(skipmissing(i_df."IPL")...])>0 &&
40         length(skipmissing(i_df."OPL")...])>0 &&
41         length(skipmissing(i_df."RPE length")...])>0
42
43         rthi=mean(skipmissing(i_df."Retina
44             → thickness"))-mean(skipmissing(i_df."OPL"))-mean(skipmissing(i_df."IPL"))
45         rpel=mean(skipmissing(i_df."RPE length"))
46         if length(skipmissing(i_df."Lens diameter")...])>0
47             rcirc=rpel+mean(skipmissing(i_df."Lens diameter"))
48         else
49             rcirc=rpel+mean(ldvec)
50         end
51
52         or=rcirc/2π
53         ir=or-.5*rthi
54
55         ov=(4/3)*π*(or^3)
56         iv=(4/3)*π*(ir^3)
57
58         push!(spvec,(ov-iv)*(4/5))
59     end
60 end
61
62 gmc=GMC_DEFAULTS
63 gmc[1]=5
64
65 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]]
66

```

```

67 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
68
69 evdict=Dict{String,Measurement}()
70
71 pmax_μ=2e5
72 pmin_μ=1
73 pmax_λ=1e-3
74 pmin_λ=1e-10
75 vmax_μ=1e10
76 vmin_μ=100
77 vmax_λ=1e-10
78 vmin_λ=1e-20
79
80 pn_priors=[Uniform(pmin_μ,pmax_μ),Uniform(pmin_λ,pmax_λ)]
81 pn_box=[pmin_μ pmax_μ;pmin_λ pmax_λ]
82 vn_priors=[Uniform(vmin_μ,vmax_μ),Uniform(vmin_λ,vmax_λ)]
83 vn_box=[vmin_μ vmax_μ;vmin_λ vmax_λ]
84
85 n_models=100
86
87 for (pth,prior,box,eststring) in zip([pne_pth, vne_pth],[pn_priors,
88   ← vn_priors],[pn_box,vn_box],[ "PopEst", "VolEst"])
89   for (nx,x) in enumerate(x)
90     enspth=pth*"/$x"
91     if isfile(enspth*"/ens")
92       e=deserialize(enspth*"/ens")
93     else
94       e=Normal_Ensemble(enspth,n_models,filter(i→!iszero(i),
95         ← measure_dict[eststring][nx]), prior, box, gmc ... )
96     end
97   end
98 end
99
100 ln_pmax_μ=log(pmax_μ^2/sqrt(pmax_μ^2 + inv(pmax_λ)))
101 ln_pmin_μ=log(pmin_μ^2/sqrt(pmin_μ^2 + inv(pmin_λ)))

```

```

102 ln_pmax_λ=1/log(1+(inv(pmax_λ)/pmax_μ^2))
103 ln_pmin_λ=1/log(1+(inv(pmin_λ)/pmin_μ^2))

104 ln_vmax_μ=log(vmax_μ^2/sqrt(vmax_μ^2 + inv(vmax_λ)))
105 ln_vmin_μ=log(vmin_μ^2/sqrt(vmin_μ^2 + inv(vmin_λ)))
106 ln_vmax_λ=1/log(1+(inv(vmax_λ)/vmax_μ^2))
107 ln_vmin_λ=1/log(1+(inv(vmin_λ)/vmin_μ^2))

108
109
110 pln_priors=[Uniform(ln_pmin_μ,ln_pmax_μ),Uniform(ln_pmin_λ,ln_pmax_λ)]
111 pln_box=[ln_pmin_μ ln_pmax_μ;ln_pmin_λ ln_pmax_λ]
112 vln_priors=[Uniform(ln_vmin_μ,ln_vmax_μ),Uniform(ln_vmin_λ,ln_vmax_λ)]
113 vln_box=[ln_vmin_μ ln_vmax_μ;ln_vmin_λ ln_vmax_λ]

114
115 for (pth,prior,box,eststring) in zip([plne_pth, vlne_pth],[pn_priors,
    ← vn_priors],[pn_box,vn_box],["PopEst", "VolEst"])
    for (nx,x) in enumerate(x)
        enspth=pth*"/$x"
        if isfile(enspth*"/ens")
            e=deserialize(enspth*"/ens")
        else
            ← e=LogNormal_Ensemble(enspth,n_models,filter(i→i>0,measure_dict[estst
            ← prior, box, gmc ... ])
        end
    end
116
117
118
119
120
121
122
123
124     pth in keys(evdict) ? (evdict[pth] +=
        ← converge_ensemble!(e,backup=true,10000),upper_displays=uds,
        ← lower_displays=lds, disp_rot_its=10000,
        ← converge_factor=1e-6)) : (evdict[pth] =
        ← converge_ensemble!(e,backup=true,10000),upper_displays=uds,
        ← lower_displays=lds, disp_rot_its=10000,
        ← converge_factor=1e-6))
125
126 end
127
128 pop_evidence_ratio=evdict[plne_pth]-evdict[pne_pth]
129 vol_evidence_ratio=evdict[vlne_pth]-evdict[vne_pth]
130
131 println("CMZ Population & $(round(evdict[pne_pth],digits=3)) &
    ← $(round(evdict[plne_pth],digits=3)) &
    ← $(round(pop_evidence_ratio,digits=3)) &
    ← $(pop_evidence_ratio.val/pop_evidence_ratio.err)\\\\' \\\\hline")

```

---

```

132 println("Estimated Retinal Volume & $(round(evdict[vne_pth],digits=3)) &
→   $(round(evdict[vlne_pth],digits=3)) &
→   $(round(vol_evidence_ratio,digits=3)) &
→   $(vol_evidence_ratio.val/vol_evidence_ratio.err)\\\\\\ \\hline")

```

---

### 17.8.8 /cmz/a10periodisation.jl

---

```

1 using CSV,DataFrames,Distributions,StatsBase,GMC_NS,Serialization,
→   CMZNicheSims, Random, Plots
2 gr()
3 default(legendfont = (10), guidefont = (12), tickfont = (10))
4
5 Random.seed!(786)
6
7 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
8 e2ph="/bench/PhD/NGS_binaries/CNS/A10/e2ph"
9 e3ph="/bench/PhD/NGS_binaries/CNS/A10/e3ph"
10
11 paths=[e2ph, e3ph]
12
13 a10df=DataFrame(CSV.read(a10pth))
14 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
→   eachrow(a10df)]
15
16 X=Vector{Float64}()
17 measure_dict=Dict{String,Vector{Vector{Float64}}}()
18
19 measure_dict["PopEst"]=Vector{Vector{Float64}}()
20 measure_dict["VolEst"]=Vector{Vector{Float64}}()
21 measure_dict["SphEst"]=Vector{Vector{Float64}}()
22 measure_dict["CircEst"]=Vector{Vector{Float64}}()
23 measure_dict["ThiEst"]=Vector{Vector{Float64}}()
24
25 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
26 for t_df in groupby(a10df, "Time point (d)")
27     push!(X,Float64(unique(t_df."Time point (d)")[1]))
28     pevec,vevec,tvec,ldvec,spvec,cvec,thivec = [zeros(0) for i in 1:7]
29
30     for i_df in groupby(t_df,"BSR")
31         length([skipmissing(i_df."Lens diameter")...])>0 &&
→           push!(ldvec,mean(skipmissing(i_df."Lens diameter"))))
32     end

```

```

33
34     for i_df in groupby(t_df, "BSR")
35         if length([skipmissing(i_df."CMZ Sum") ... ])>0 &&
36             mean(skipmissing(i_df."CMZ Sum")) > 0
37             push!(tvec,mean(skipmissing(i_df."CMZ Sum")))
38             if length([skipmissing(i_df."Lens diameter") ... ])>0
39                 push!(pevec,mean(skipmissing(i_df."CMZ
40                     → Sum"))*mean(skipmissing(i_df."Lens diameter"))/14.)
41             else
42                 push!(pevec,mean(skipmissing(i_df."CMZ
43                     → Sum"))*mean(ldvec)/14.)
44             end
45         end
46         if length([skipmissing(i_df."Retina thickness") ... ])>0 &&
47             length([skipmissing(i_df."IPL") ... ])>0 &&
48             length([skipmissing(i_df."OPL") ... ])>0 &&
49             length([skipmissing(i_df."RPE length") ... ])>0
50
51             rthi=mean(skipmissing(i_df."Retina
52                 → thickness"))-mean(skipmissing(i_df."OPL"))-mean(skipmissing(i_df."IPL"))
53             push!(thivec,rthi)
54             rpel=mean(skipmissing(i_df."RPE length"))
55             if length([skipmissing(i_df."Lens diameter") ... ])>0
56                 rcirc=rpel+mean(skipmissing(i_df."Lens diameter"))
57             else
58                 rcirc=rpel+mean(ldvec)
59             end
60
61             push!(cvec,rcirc)
62
63             or=rcirc/2π
64             ir=or-rthi
65
66             ov=(4/3)*π*(or^3)
67             iv=(4/3)*π*(ir^3)
68
69             push!(spvec,(ov-iv)*(4/5))
70         end
71     end
72
73     push!(measure_dict["PopEst"],pevec)
74     push!(measure_dict["SphEst"],spvec)
75     push!(measure_dict["CircEst"],cvec)

```

```

72     push!(measure_dict["ThiEst"],thivec)
73 end
74
75 obs=[(measure_dict["PopEst"][t],measure_dict["SphEst"][t]) for t in
76   ↪ 1:length(X)]
77
78 nucpop3d=2075.6 #from a38 sib_measure_dict NucPop mean
79 or=mean(measure_dict["CircEst"][[1]])/2π
80 ir=or-.5*mean(measure_dict["ThiEst"][[1]])
81 sxvol_3d=((π*or^2)-(π*ir^2))*14)*(4/5)
82 vol_const=sxvol_3d/nucpop3d
83 mc_its=Int64(1.e6)
84
85 popdist=fit(LogNormal,measure_dict["PopEst"][[1]])
86 voldist=fit(LogNormal,measure_dict["SphEst"][[1]])
87
88 ph2_constants=[X, popdist, voldist, vol_const, mc_its, 2]
89 ph3_constants=[X, popdist, voldist, vol_const, mc_its, 3]
90 constants=[ph2_constants, ph3_constants]
91
92 cycle_prior=[Uniform(10.,144.), LogNormal(log(.9),log(2))]
93 end_prior=[Uniform(4.,359.)]
94
95 phase_max=[144.,5.]
96 phase_min=[10.,0.]
97 time_max=359
98 time_min=4.
99
100 function compose_priors(phases)
101   prs=Vector{Vector{Distribution}}()
102   bxes=Vector{Matrix{Float64}}()
103   for p in phases
104     pr=[repeat(cycle_prior,p) ... ,repeat(end_prior,p-1) ... ]
105     push!(prs,pr)
106     bx=hcat([repeat(phase_min,p) ... ,fill(time_min,p-1) ... ],
107              [repeat(phase_max,p) ... ,fill(time_max,p-1) ... ])
108     push!(bxes,GMC_NS.to_unit_ball.(bx,pr))
109   end
110   return prs,bxes
111 end
112
113 phases=[2,3]
114 priors,boxes=compose_priors(phases)

```

```

114
115 uds=Vector{Vector{Function}}([[],[liwi_display],[convergence_display]])
116 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display],[ensemble_displa
117
118 gmc_settings=GMC_DEFAULTS
119 gmc_settings[end]=100
120
121 for (pth,prior,constants,box) in zip(paths,priors,constants,boxes)
122     if isfile(pth*"/ens")
123         e=deserialize(pth*"/ens")
124     else
125         e=CMZ_Ensemble(pth,1000,obs, prior, constants, box, gmc_settings)
126     end
127
128     converge_ensemble!(e,backup=(true,50),upper_displays=uds,
129     ↪ lower_displays=lds, disp_rot_its=100, mc_noise=.3,
130     ↪ converge_factor=1e-6)
131 end
132
133
134 e2ev=measure_evidence(e2)
135 e3ev=measure_evidence(e3)
136
137 evidence_ratio=e2ev-e3ev
138
139 println("\\"begin{tabular}{l|l|l|l}")
140 println("\\"hline")
141 println("{\\bf 2-phase logZ} & {\\bf 3-phase logZ} & {\\bf logZR} & {\bf
142     ↪ $\sigma$ Significance}\\"hline")
143 println("\\"textbf{${(round(e2ev,digits=3))} & ${round(e3ev,digits=3)} &
144     ↪ ${round(evidence_ratio,digits=3)} &
145     ↪ ${round(evidence_ratio.val/evidence_ratio.err,digits=3)}\\\\
146     ↪ \\"hline")
147
148 println("\\"end{tabular}")

```

```

149 println("\hline")
150 println("{\bf Parameter} & {\bf 2-phase MAP} & {\bf 3-phase MAP}\\"
    ↵ \hline)
151 println("Phase 1 \$CT\$ (h) & $(round(map2.θ[1], digits=1)) &
    ↵ $(round(map3.θ[1], digits=1))\\\\\\hline")
152 println("Phase 1 \$\epsilon\$ & $(round(map2.θ[2], digits=2)) &
    ↵ $(round(map3.θ[2], digits=2))\\\\\\hline")
153 println("Phase 2 \$CT\$ (h) & $(round(map2.θ[3], digits=1)) &
    ↵ $(round(map3.θ[3], digits=1))\\\\\\hline")
154 println("Phase 2 \$\epsilon\$ & $(round(map2.θ[4], digits=2)) &
    ↵ $(round(map3.θ[4], digits=2))\\\\\\hline")
155 println("Phase 3 \$CT\$ (h) & NA & $(round(map3.θ[5], digits=1))\\\\\\
    ↵ \hline")
156 println("Phase 3 \$\epsilon\$ & NA & $(round(map3.θ[6], digits=2))\\\\\\
    ↵ \hline")
157 println("Transition 1 age & $(round(map2.θ[5], digits=1)) &
    ↵ $(round(map3.θ[7], digits=1))\\\\\\hline")
158 println("Transition 2 age & NA & $(round(map3.θ[8], digits=1))\\\\\\
    ↵ \hline")
159 println("\end{tabular}")

160
161 X=e2.constants[1]
162 plotticks=[30*i for i in 1:12]

163
164 catpobs=vcat([e2.obs[t][1] for t in 1:length(X)] ... )
165 catvobs=vcat([e2.obs[t][2] for t in 1:length(X)] ... )
166 Xs=vcat([[X[n] for i in 1:length(e2.obs[n][1])] for n in 1:length(X)] ... )

167
168 p2_mean=map2.disp_mat[:,2,1]
169 p2_upper=map2.disp_mat[:,1,1]-map2.disp_mat[:,2,1]
170 p2_lower=map2.disp_mat[:,2,1]-map2.disp_mat[:,3,1]

171
172 map2_popplt=scatter(Xs,catpobs, marker=:cross, color=:black,
    ↵ markersize=3, label="CMZ population estimate", ylabel="Population",
    ↵ showaxis=:y, xticks=plotticks, xformatter=_→'', legend=:topright,
    ↵ background_color_legend=nothing)
173 plot!(map2_popplt, X, p2_mean, ribbon=(p2_lower,p2_upper),
    ↵ color=:darkmagenta, label="2-phase model population")
174 lens!([2, 32], [300, 5700], inset = (1, bbox(0.6, 0.0, 0.38, 0.7)))
175 annotate!([(30,1.25e4,Plots.text("A1",18))])

176
177 v2_mean=map2.disp_mat[:,2,2]
178 v2_upper=map2.disp_mat[:,1,2]-map2.disp_mat[:,2,2]

```

```

179 v2_lower=map2.disp_mat[:,2,2]--map2.disp_mat[:,3,2]
180
181 map2_volplt=scatter(Xs,catvobs, marker=:cross, color=:black,
    ↵ markersize=3, label="Retinal volume estimate", ylabel="Volume
    ↵ ( $\mu\text{m}^3$ )", xlabel="Age (dpf)", legend=:topleft, showaxis=:y,
    ↵ xticks=plotticks, xformatter=_ $\rightarrow$  "", background_color_legend=nothing)
182 plot!(map2_volplt, X, v2_mean, ribbon=(v2_lower,v2_upper), color=:green,
    ↵ label="2-phase model volume")
183 lens!([2, 32], [2e6, 3e7], inset = (1, bbox(0.6, 0.0, 0.34, 0.5)))
184 annotate!([(30,3.5e8,Plots.text("A2",18))])
185
186
187 p3_mean=map3.disp_mat[:,2,1]
188 p3_upper=map3.disp_mat[:,1,1]--map3.disp_mat[:,2,1]
189 p3_lower=map3.disp_mat[:,2,1]--map3.disp_mat[:,3,1]
190
191 map3_popplt=scatter(Xs,catpobs, marker=:cross, color=:black,
    ↵ markersize=3, label="CMZ population estimate", ylabel="Population",
    ↵ showaxis=:y, xticks=plotticks, xformatter=_ $\rightarrow$  "", legend=:top,
    ↵ background_color_legend=nothing)
192 plot!(map3_popplt, X, p3_mean, ribbon=(p3_lower,p3_upper),
    ↵ color=:darkmagenta, label="3-phase model population")
193 lens!([2, 32], [300, 5700], inset = (1, bbox(0.6, 0.0, 0.38, 0.7)))
194 annotate!([(30,1.5e4,Plots.text("B1",18))])
195
196 v3_mean=map3.disp_mat[:,2,2]
197 v3_upper=map3.disp_mat[:,1,2]--map3.disp_mat[:,2,2]
198 v3_lower=map3.disp_mat[:,2,2]--map3.disp_mat[:,3,2]
199
200 map3_volplt=scatter(Xs,catvobs, marker=:cross, color=:black,
    ↵ markersize=3, label="Retinal volume estimate", ylabel="Volume
    ↵ ( $\mu\text{m}^3$ )", xlabel="Age (dpf)", legend=:topleft, xticks=plotticks,
    ↵ background_color_legend=nothing)
201 plot!(map3_volplt, X, v3_mean, ribbon=(v3_lower,v3_upper), color=:green,
    ↵ label="3-phase model volume")
202 lens!([2, 32], [2e6, 3e7], inset = (1, bbox(0.6, 0.0, 0.34, 0.5)))
203 annotate!([(30,3.5e8,Plots.text("B2",18))])
204
205 combined_map=Plots.plot(map2_popplt,map2_volplt,map3_popplt,map3_volplt,layout=grid(4
    ↵ size=(1200,1200), link=:x)
206
207 savefig(combined_map,"/bench/PhD/Thesis/images/cmz/a10pMAP.png")
208

```

```

209 kdes=posterior_kde(e2)
210
211 ctmarg=plot(Uniform(10,144), color=:darkmagenta, fill=true,
212   ← fillalpha=.5,xlims=[0,144], xlabel="Cycle Time (hr)",
213   ← ylabel="Probability density", label="Prior")
214 plot!(kdes[1].x,kdes[1].density,color=:green, fill=true, fillalpha=.5,
215   ← label="Ph1 Posterior")
216 plot!(kdes[3].x,kdes[3].density,color=:orange, fill=true, fillalpha=.5,
217   ← label="Ph2 Posterior")
218 plot!([map2.θ[1],map2.θ[1]],[0,maximum(kdes[1].density)],color=:darkgreen,
219   ← label="Ph1 MAP")
220 plot!([map2.θ[3],map2.θ[3]],[0,maximum(kdes[3].density)],color=:darkred,
221   ← label="Ph2 MAP")
222 annotate!([(0,.11,Plots.text("A",18))])
223
224 ymarg=map(x→pdf(LogNormal(log(.9),log(2)),x),kdes[2].x)
225
226 ermarg=plot(kdes[2].x, ymarg, color=:darkmagenta, fill=true,
227   ← fillalpha=.5, xlims=[0,4], xlabel="Niche exit rate  $\epsilon$ ,
228   ← ylabel="Probability density", label="Prior")
229 plot!(kdes[2].x,kdes[2].density,color=:green, fill=true, fillalpha=.5,
230   ← label="Ph1 Posterior")
231 plot!(kdes[4].x,kdes[4].density,color=:orange, fill=true, fillalpha=.5,
232   ← label="Ph2 Posterior")
233 plot!([map2.θ[2],map2.θ[2]],[0,maximum(kdes[2].density)],color=:darkgreen,
234   ← label="Ph1 MAP")
235 plot!([map2.θ[4],map2.θ[4]],[0,maximum(kdes[4].density)],color=:darkred,
236   ← label="Ph2 MAP")
237 annotate!([(0,7.,Plots.text("B",18))])
238
239 transmarg=plot(Uniform(4.,359.), color=:darkmagenta, fill=true,
240   ← fillalpha=.5,xlims=[0,360], xlabel="Phase transition age (dpf)",
241   ← ylabel="Probability density", label="Prior")
242 plot!(kdes[5].x,kdes[5].density,color=:green, fill=true, fillalpha=.5,
243   ← label="Ph1-2 posterior")
244 plot!([map2.θ[5],map2.θ[5]],[0,maximum(kdes[5].density)],color=:darkgreen,
245   ← label="Ph1-2 MAP")
246 annotate!([(0,.04,Plots.text("C",18))])
247
248 ph1c=plot(ctmarg,ermarg,transmarg, layout=grid(3,1),size=(1000,1200))
249 savefig(ph1c,"/bench/PhD/Thesis/images/cmz/a10pmarginals.png")

```

### 17.8.9 /cmz/a10popsurvey.jl

```

1 using CSV,DataFrames,Distributions,NGRefTools,StatsBase,Plots
2 gr()
3 default(legendfont = (10), guidefont = (11), tickfont = (10))
4
5 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
6
7 a10df=DataFrame(CSV.read(a10pth))
8 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
9      eachrow(a10df)]
10
11 X=Vector{Float64}()
12 measure_dict=Dict{String,Vector{Vector{Float64}}}()
13 measure_dict["PopEst"]=Vector{Vector{Float64}}()
14 measure_dict["CMZ Sum"]=Vector{Vector{Float64}}()
15 measure_dict["Dorsal CMZ (#)"]=Vector{Vector{Float64}}()
16 measure_dict["Ventral CMZ (#)"]=Vector{Vector{Float64}}()
17 measure_dict["VolEst"]=Vector{Vector{Float64}}()
18 measure_dict["Retina thickness"]=Vector{Vector{Float64}}()
19 measure_dict["RPE length"]=Vector{Vector{Float64}}()
20 measure_dict["Lens diameter"]=Vector{Vector{Float64}}()
21 measure_dict["ON diameter"]=Vector{Vector{Float64}}()
22 measure_dict["ONL"]=Vector{Vector{Float64}}()
23 measure_dict["OPL"]=Vector{Vector{Float64}}()
24 measure_dict["INL"]=Vector{Vector{Float64}}()
25 measure_dict["IPL"]=Vector{Vector{Float64}}()
26 measure_dict["GCL"]=Vector{Vector{Float64}}()
27 measure_dict["Pop/VolEst"]=Vector{Vector{Float64}}()
28
29 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
30 for t_df in groupby(a10df, "Time point (d)")
31     push!(X,Float64(unique(t_df."Time point (d)")[1]))
32     pevec,tvec,dvec,vvec,vevec, rtvec, rplvec, rptvec, ldvec, ondvec,
33         onlvec, oplvec, inlvec, iplvec, gclvec, pvest = [zeros(0) for i
34             in 1:16]
35
36 for i_df in groupby(t_df,"BSR")
37     length(skipmissing(i_df."Lens diameter")...)])>0 &&
38         push!(ldvec.mean(skipmissing(i df."Lens diameter")))

```

```

36     end
37
38     for i_df in groupby(t_df, "BSR")
39         ipop=false
40         ivol=false
41         if length([skipmissing(i_df."CMZ Sum") ... ])>0
42             push!(tvec,mean(skipmissing(i_df."CMZ Sum")))
43             if length([skipmissing(i_df."Lens diameter") ... ])>0
44                 push!(pevec,mean(skipmissing(i_df."CMZ
45                     → Sum"))*mean(skipmissing(i_df."Lens diameter"))/14.)
46                 ipop=true
47             else
48                 push!(pevec,mean(skipmissing(i_df."CMZ
49                     → Sum"))*mean(ldvec)/14.)
50                 ipop=true
51             end
52         end
53
54         length([skipmissing(i_df."Dorsal CMZ (#)") ... ])>0 &&
55             → push!(dvec,mean(skipmissing(i_df."Dorsal CMZ (#)")))
56         length([skipmissing(i_df."Ventral CMZ (#)") ... ])>0 &&
57             → push!(vvec,mean(skipmissing(i_df."Ventral CMZ (#)")))
58         if length([skipmissing(i_df."Retina thickness") ... ])>0 &&
59             length([skipmissing(i_df."IPL") ... ])>0 &&
60             length([skipmissing(i_df."OPL") ... ])>0 &&
61             length([skipmissing(i_df."RPE length") ... ])>0
62
63             rthi=mean(skipmissing(i_df."Retina
64                 → thickness"))-mean(skipmissing(i_df."OPL"))-mean(skipmissing(i_df."IPL"))
65             rpel=mean(skipmissing(i_df."RPE length"))
66             if length([skipmissing(i_df."Lens diameter") ... ])>0
67                 rcirc=rpel+mean(skipmissing(i_df."Lens diameter"))
68             else
69                 rcirc=rpel+mean(ldvec)
70             end
71
72             or=rcirc/2π
73             ir=or-.5*rthi
74
75             ov=(4/3)*π*(or^3)
76             iv=(4/3)*π*(ir^3)
77
78             push!(vevec,(ov-iv)*(4/5))

```

```

74     end
75     length(skipmissing(i_df."Retina thickness") ... ])>0 &&
76         → push!(rtvec,mean(skipmissing(i_df."Retina thickness")))
77     length(skipmissing(i_df."RPE length") ... ])>0 &&
78         → push!(rptvec,mean(skipmissing(i_df."RPE length")))
79     length(skipmissing(i_df."ON diameter") ... ])>0 &&
80         → push!(ondvec,mean(skipmissing(i_df."ON diameter")))
81     length(skipmissing(i_df.ONL) ... ])>0 &&
82         → push!(onlvec,mean(skipmissing(i_df.ONL)))
83     length(skipmissing(i_df.OPL) ... ])>0 &&
84         → push!(oplvec,mean(skipmissing(i_df.OPL)))
85     length(skipmissing(i_df.INL) ... ])>0 &&
86         → push!(inlvec,mean(skipmissing(i_df.INL)))
87     length(skipmissing(i_df.IPL) ... ])>0 &&
88         → push!(iplvec,mean(skipmissing(i_df.IPL)))
89     length(skipmissing(i_df.GCL) ... ])>0 &&
90         → push!(gclvec,mean(skipmissing(i_df.GCL)))
91     ivol && ipop &&
92     push!(pvest,
93         mean(skipmissing(i_df."CMZ Sum"))/
94             ((mean(skipmissing(i_df."Retina
95                 → thickness"))-mean(skipmissing(i_df."OPL"))-mean(skipmissing(i_df."IPL
96                     *mean(skipmissing(i_df."RPE length"))*(π/4)))
97     end
98
99
100    push!(measure_dict["PopEst"],pevec)
101    push!(measure_dict["CMZ Sum"],tvec)
102    push!(measure_dict["Dorsal CMZ (#)"],dvec)
103    push!(measure_dict["Ventral CMZ (#)"],vvec)
104    push!(measure_dict["VolEst"],vevec)
105    push!(measure_dict["Retina thickness"],rtvec)
106    push!(measure_dict["RPE length"],rptvec)
107    push!(measure_dict["Lens diameter"],ldvec)
108    push!(measure_dict["ON diameter"],ondvec)
109    push!(measure_dict["ONL"],onlvec)
110    push!(measure_dict["OPL"],oplvec)
111    push!(measure_dict["INL"],inlvec)
112    push!(measure_dict["IPL"],iplvec)
113    push!(measure_dict["GCL"],gclvec)
114    push!(measure_dict["Pop/VolEst"],pvest)
115 end
116
117 #TEST IF DATA IS BETTER MODELLED NORMALLY OR LOGNORMALLY

```

```

108 for ms in ["CMZ Sum", "Lens diameter", "PopEst", "RPE length", "Retina
  ↪ thickness", "VolEst"]
109   normal_mts=[fit(Normal,measure_dict[ms][n]) for n in 1:length(X)]
110   logn_mts=[fit(LogNormal,filter!(i→!iszero(i),measure_dict[ms][n]))
  ↪ for n in 1:length(X)]
111
112   normal_lh=sum([sum(logpdf(normal_mts[n],measure_dict[ms][n])) for n
  ↪ in 1:length(X)])
113   logn_lh=sum([sum(logpdf(logn_mts[n],measure_dict[ms][n])) for n in
  ↪ 1:length(X)])
114
115   println("$ms & $(round(normal_lh,digits=3)) &
  ↪ $(round(logn_lh,digits=3)) &
  ↪ $(round(logn_lh-normal_lh,digits=3))\\\ \\\hline")
116
117 end
118
119 pe_n_mts=[fit(MarginalTDist,measure_dict["PopEst"][n]) for n in
  ↪ 1:length(X)]
120 pen_lower=[mean(mt)-quantile(mt,.025) for mt in pe_n_mts]
121 pen_upper=[quantile(mt,.975)-mean(mt) for mt in pe_n_mts]
122
123 pe_logn_mts=[fit(MarginalTDist,log.(filter!(i→!iszero(i),measure_dict["PopEst"][n]))
  ↪ for n in 1:length(X)]
124 pe_mean=[exp(mean(mt)) for mt in pe_logn_mts]
125 pe_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in pe_logn_mts]
126 pe_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in pe_logn_mts]
127
128 plotticks=[30*i for i in 1:12]
129
130 ann_chart=scatter(vcat([[X[n] for i in
  ↪ 1:length(measure_dict["PopEst"][n])] for n in
  ↪ 1:length(X)] ... ),vcat(measure_dict["PopEst"] ... ), marker=:cross,
  ↪ color=:black, markersize=3, label="Pop. data", ylabel="Est. CMZ
  ↪ annulus population", showaxis=:y, xticks=plotticks, xformatter=_→ ""))
131 plot!(ann_chart, X, pe_mean, ribbon=(pe_lower,pe_upper),
  ↪ color=:darkmagenta, label="Pop. mean")
132 lens!([2, 20], [500, 2500], inset = (1, bbox(0.3, 0.1, 0.45, 0.45)))
133 annotate!([(8,4500,Plots.text("A",18))])
134
135 poprate_mean,poprate_lower,poprate_upper=[Vector{Float64}() for i in 1:3]
136 for (nx, x) in enumerate(X)
  ↪ if nx > 1

```

```

138     d=X[nx]-X[nx-1]
139     l,m,u=MTDist_MC_func((a,b)→((exp(a)-exp(b))/d),
140     ↵ [log.(filter!(i→!iszero(i),measure_dict["PopEst"])[nx])),
141     ↵ log.(filter!(i→!iszero(i),measure_dict["PopEst"])[nx-1])),  

142     ↵ summary=true)
143
144 poprate_chart=plot(X[2:end],poprate_mean,ribbon=(poprate_lower,
145   ↵ poprate_upper), color=:grey, ylabel="Est. CMZ pop. rate of change  

146   ↵ (cells/d)", label="Simulated mean rate", legend=:topright,  

147   ↵ showaxis=:y, xticks=plotticks, xformatter=_→"")
148 annotate!([(8,400,Plots.text("B",18))])
149
150 l3,m3,u3=MTDist_MC_func((a,b)→(exp(a)-exp(b))/5.,
151   ↵ [filter!(isinf,log.(measure_dict["PopEst"])[2])),  

152   ↵ filter!(isinf,log.(measure_dict["PopEst"])[1])), summary=true)
153
154 ratedist23=MTDist_MC_func((a,b)→(exp(a)-exp(b))/6.,
155   ↵ [filter!(isinf,log.(measure_dict["PopEst"])[6])),  

156   ↵ filter!(isinf,log.(measure_dict["PopEst"])[5])), summary=false)
157
158 println("3dpf poprate mean: $(m3) 23dpf poprate mean
159   ↵ $(mean(ratedist23))poprate 23dpf quantile :
160   ↵ $(quantile(ratedist23,.025))")
161
162 ve_logn_mts=[fit(MarginalTDist,log.(filter!(i→!iszero(i),measure_dict["VolEst"])[n]))
163   ↵ for n in 1:length(X)]
164
165 ve_mean=[exp(mean(mt)) for mt in ve_logn_mts]
166
167 ve_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in ve_logn_mts]
168
169 ve_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in ve_logn_mts]
170
171 vol_chart=scatter(vcat([[X[n] for i in
172   ↵ 1:length(measure_dict["VolEst"])[n]]) for n in
173   ↵ 1:length(X)] ... ),vcat(measure_dict["VolEst"] ... ), marker=:cross,
174   ↵ color=:black, markersize=3, label="Vol. data", ylabel="Est. retinal  

175   ↵ volume ( $\mu\text{m}^3$ )", showaxis=:y, xticks=plotticks, xformatter=_→"",
176   ↵ legend=:bottomright)
177
178 plot!(vol_chart, X, ve_mean, ribbon=(ve_lower,ve_upper), color=:blue,
179   ↵ label="Vol. mean")

```

```

161 lens!([2, 20], [2.5e6, 1e7], inset = (1, bbox(0.12, 0., 0.39, 0.40)))
162 annotate!([(8,4.5e8,Plots.text("C",18))])
163
164 println("$(ccdf(ve_logn_mts[2],log(ve_mean[1]))) % of marginal posterior
→ mass of 5dpf volume estimate is greater than the 3dpf estimated
→ mean")
165
166 volrate_mean,volrate_lower,volrate_upper=[Vector{Float64}() for i in 1:3]
167 for (nx, x) in enumerate(x)
168     if nx > 1
169         d=X[nx]-X[nx-1]
170         l,m,u=MTDist_MC_func((a,b)→(exp(a)-exp(b))/d,
171             [filter{!(isinf,log.(measure_dict["VolEst"])[nx])},
172             filter{!(isinf,log.(measure_dict["VolEst"])[nx-1])}],
173             summary=true)
174             → push!(volrate_mean,m);push!(volrate_lower,l);push!(volrate_upper,u)
175     end
176 end
177 volrate_lower=volrate_mean--volrate_lower
178 volrate_upper=volrate_upper--volrate_mean
179
180 l=@layout [a{0.3h}
181             b{0.2h}
182             c{0.3h}
183             d]
184 combined=plot(ann_chart,poprate_chart,vol_chart,volrate_chart,layout=(4,1),size=(1200
185 savefig(combined, "/bench/PhD/Thesis/images/cmz/CMZoverall.png")
186
187 l30,m30,u30=MTDist_MC_func((a,b)→(exp(a)-exp(b))/7.,
188     [filter{!(isinf,log.(measure_dict["VolEst"])[7])},
189     filter{!(isinf,log.(measure_dict["VolEst"])[6])}], summary=true)
190 ratedist60=MTDist_MC_func((a,b)→(exp(a)-exp(b))/30.,
191     [filter{!(isinf,log.(measure_dict["VolEst"])[8])},
192     filter{!(isinf,log.(measure_dict["VolEst"])[7])}], summary=false)

```

---

```

191 println("6*30dpf volrate mean: ${6*m30) volrate 60dpf quantile :
→   $(quantile(ratedist60,.01))")

```

---

### 17.8.10 /cmz/a19lineagetrace.jl

---

```

1 using CSV,DataFrames,Distributions,NGRefTools,StatsBase,Plots,
→   Measurements,GMC_NS,Serialization
2 gr()
3
4 a19_1pth="/bench/PhD/datasets/A19GR1.csv"
5 a19_2pth="/bench/PhD/datasets/A19GR2.csv"
6 a19_3pth="/bench/PhD/datasets/A19GR3.csv"
7
8 gr1df=DataFrame(CSV.read(a19_1pth))
9 gr2df=DataFrame(CSV.read(a19_2pth))
10 gr3df=DataFrame(CSV.read(a19_3pth))
11
12 X=sort(unique(gr1df."Pulse age (dpf)"))
13 t_measure_dict=Dict{String,Vector{Vector{Float64}}}()
14 d_measure_dict=Dict{String,Vector{Vector{Float64}}}()
15 v_measure_dict=Dict{String,Vector{Vector{Float64}}}()
16
17 mds=[t_measure_dict,d_measure_dict,v_measure_dict]
18
19 for md in mds
20     md["ONL"]=[zeros(0) for i in 1:length(X)]
21     md["INL"]=[zeros(0) for i in 1:length(X)]
22     md["GCL"]=[zeros(0) for i in 1:length(X)]
23     md["Isl"]=[zeros(0) for i in 1:length(X)]
24     md["Pax6"]=[zeros(0) for i in 1:length(X)]
25     md["Isl_Pax6"]=[zeros(0) for i in 1:length(X)]
26     md["INL_Pax6"]=[zeros(0) for i in 1:length(X)]
27     md["PKCB"]=[zeros(0) for i in 1:length(X)]
28     md["GS"]=[zeros(0) for i in 1:length(X)]
29     md["HM"]=[zeros(0) for i in 1:length(X)]
30     md["Zpr1"]=[zeros(0) for i in 1:length(X)]
31 end
32
33 function check_layerfrac(row)
34     !ismissing(row.#GCL) && !ismissing(row.#INL) &&
→       !ismissing(row.#PRL) && !ismissing(row.Total) ? (return true)
→       : (return false)

```

```

35 end
36
37 function gr1_dfcheck(subdf)
38     pass=false( size(subdf,1) )
39     for (n,row) in enumerate(eachrow(subdf))
40         !ismissing(row."GCL - Isl") && !ismissing(row."GCL - Pax6") &&
41             !ismissing(row."GCL - Isl/Pax6") && !ismissing(row."INL -
42             Pax6") ? (pass[n]=true) : (pass[n]=false)
43     end
44     all(pass) ? (return true) : (return false)
45 end
46
47 for df in [gr1df,gr2df,gr3df]
48     for agedf in groupby(df, "Pulse age (dpf)")
49         xidx=findfirst(i→i=unique(agedf."Pulse age (dpf)")[1],X)
50         for inddf in groupby(agedf, "Individual")
51             if df === gr1df
52                 if gr1_dfcheck(inddf)
53                     clean=false(2)
54                     for row in eachrow(inddf)
55                         if row."D/V" == "D"
56                             push!(d_measure_dict["Isl"][xidx],row."GCL -
57                                 Isl"/row.#GCL")
58                             push!(d_measure_dict["Pax6"][xidx],row."GCL -
59                                 Pax6"/row.#GCL")
60
61                             push!(d_measure_dict["Isl_Pax6"][xidx],row."GCL -
62                                 - Isl/Pax6"/row.#GCL")
63
64                             push!(d_measure_dict["INL_Pax6"][xidx],row."INL -
65                                 - Pax6"/row.#INL")
66                         if check_layerfrac(row)
67
68                             push!(d_measure_dict["GCL"][xidx],row.#GCL/row.
69
70                             push!(d_measure_dict["INL"][xidx],row.#INL/row.
71
72                             push!(d_measure_dict["ONL"][xidx],row.#PRL/row.
73
74                         clean[1]=true
75                     end
76                 else
77                     push!(v_measure_dict["Isl"][xidx],row."GCL -
78                         Isl"/row.#GCL")
79
80                     push!(v_measure_dict["Pax6"][xidx],row."GCL -
81                         Pax6"/row.#Pax6")
82
83                     push!(v_measure_dict["INL"][xidx],row."INL -
84                         Pax6"/row.#INL")
85
86                     push!(v_measure_dict["ONL"][xidx],row."PRL -
87                         INL"/row.#ONL")
88
89                     clean[1]=true
90                 end
91             end
92         end
93     end
94 end

```

```

66         push!(v_measure_dict["Pax6"][xidx],row."GCL -
67             ↳ " Pax6"/row. "#GCL")
68
69             ↳ push!(v_measure_dict["Isl_Pax6"][xidx],row."GCL
70                 - Isl/Pax6"/row. "#GCL")
71
72             ↳ push!(v_measure_dict["INL_Pax6"][xidx],row."INL
73                 - Pax6"/row. "#INL")
74     if check_layerfrac(row)
75
76             ↳ push!(v_measure_dict["GCL"][xidx],row. "#GCL"/row.
77
78             ↳ push!(v_measure_dict["INL"][xidx],row. "#INL"/row.
79
80             ↳ push!(v_measure_dict["ONL"][xidx],row. "#PRL"/row.
81                 clean[2]=true
82
83             end
84
85         end
86
87     elseif df == gr2df
88         for row in eachrow(inddf)
89             if row."D/V" == "D"
90
91                 ↳ push!(d_measure_dict["PKCB"][xidx],row."INL-PKCB"/row. "#INL")
92
93                 ↳ push!(d_measure_dict["GS"][xidx],row."INL-GS"/row. "#INL")

```

```

92          ↳ push!(d_measure_dict["HM"][xidx],row."INL-HM"/row."#INL")
93
94          ↳ push!(d_measure_dict["GCL"][xidx],row.#GCL"/row.Total")
95
96          ↳ push!(d_measure_dict["INL"][xidx],row.#INL"/row.Total")
97
98          ↳ push!(d_measure_dict["ONL"][xidx],row.#PRL"/row.Total")
99
100     else
101
102         ↳ push!(v_measure_dict["PKCB"][xidx],row."INL-PKCB"/row.#INL")
103
104         ↳ push!(v_measure_dict["GS"][xidx],row."INL-GS"/row.#INL")
105
106         ↳ push!(v_measure_dict["HM"][xidx],row."INL-HM"/row.#INL")
107
108         ↳ push!(v_measure_dict["GCL"][xidx],row.#GCL"/row.Total")
109
110         ↳ push!(v_measure_dict["INL"][xidx],row.#INL"/row.Total")
111
112     end
113
114
115         ↳ push!(t_measure_dict["PKCB"][xidx],sum(inddf."INL-PKCB")/sum(inddf."
116
117         ↳ push!(t_measure_dict["GS"][xidx],sum(inddf."INL-GS")/sum(inddf.#INL")
118
119         ↳ push!(t_measure_dict["HM"][xidx],sum(inddf."INL-HM")/sum(inddf.#INL")
120
121         ↳ push!(t_measure_dict["GCL"][xidx],sum(inddf.#GCL")/sum(inddf.Total")
122
123         ↳ push!(t_measure_dict["INL"][xidx],sum(inddf.#INL")/sum(inddf.Total")
124
125
126         ↳ push!(t_measure_dict["ONL"][xidx],sum(inddf.#PRL")/sum(inddf.Total")
127
128 elseif df == gr3df
129     for row in eachrow(inddf)
130         if row."D/V" == "D"
131             push!(d_measure_dict["Zpr1"][xidx],row.PRL -
132                   ↳ Zpr1"/row.#PRL")
133
134
135         ↳ push!(d_measure_dict["GCL"][xidx],row.#GCL"/row.Total")

```

```

116             ↳ push!(d_measure_dict["INL"][xidx],row.">#INL"/row."Total")
117             ↳ push!(d_measure_dict["ONL"][xidx],row.">#PRL"/row."Total")
118         else
119             push!(v_measure_dict["Zpr1"][xidx],row."PRL -
120             ↳ Zpr1"/row."#PRL")
121             push!(v_measure_dict["GCL"][xidx],row.">#GCL"/row."Total")
122             ↳ push!(v_measure_dict["INL"][xidx],row.">#INL"/row."Total")
123             ↳ push!(v_measure_dict["ONL"][xidx],row.">#PRL"/row."Total")
124         end
125         push!(t_measure_dict["Zpr1"][xidx],sum(inddf."PRL -
126             ↳ Zpr1")/sum(inddf."#PRL"))
127             ↳ push!(t_measure_dict["GCL"][xidx],sum(inddf."#GCL")/sum(inddf."To
128             ↳ push!(t_measure_dict["INL"][xidx],sum(inddf."#INL")/sum(inddf."To
129             ↳ push!(t_measure_dict["ONL"][xidx],sum(inddf."#PRL")/sum(inddf."To
130         end
131     end
132 end
133
134 for d in mds
135     for (k,v) in d
136         for ovec in v
137             if any(i→i==0.,ovec)
138                 ovec[findall(i→i==0.,ovec)]*=1e-3
139             end
140         end
141         d[k]=v
142     end
143 end
144
145 onl_n_mts=[fit(MarginalTDist,t_measure_dict["ONL"][n]) for n in
146     ↳ 1:length(X)]
146 onl_mean=[mean(mt) for mt in onl_n_mts]
147 onl_lower=[mean(mt)-quantile(mt,.025) for mt in onl_n_mts]

```

```

148 onl_upper=[quantile(mt,.975)-mean(mt) for mt in onl_n_mts]
149
150 inl_n_mts=[fit(MarginalTDist,t_measure_dict["INL"][n]) for n in
    ↪ 1:length(X)]
151 inl_mean=[mean(mt) for mt in inl_n_mts]
152 inl_lower=[mean(mt)-quantile(mt,.025) for mt in inl_n_mts]
153 inl_upper=[quantile(mt,.975)-mean(mt) for mt in inl_n_mts]
154
155 gcl_ln_mts=[fit(MarginalTDist,log.(t_measure_dict["GCL"][n])) for n in
    ↪ 1:length(X)]
156 gcl_mean=[exp(mean(mt)) for mt in gcl_ln_mts]
157 gcl_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in gcl_ln_mts]
158 gcl_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in gcl_ln_mts]
159
160 layers_chart=scatter(vcat([[X[n] for i in
    ↪ 1:length(t_measure_dict["GCL"][n])] for n in
    ↪ 1:length(X)] ... ),vcat(t_measure_dict["GCL"] ... ), marker=:utriangle,
    ↪ color=:darkmagenta, markersize=3, markerstrokecolor=:darkmagenta,
    ↪ label="GCL data", ylabel="Fraction of cohort in layer", xlabel="Age
    ↪ (dpf)", xticks=X, foreground_color_legend=nothing,
    ↪ background_color_legend=nothing, legend=:inside)
161 plot!(X, gcl_mean, ribbon=(gcl_lower,gcl_upper), color=:darkmagenta,
    ↪ label="GCL mean")
162 scatter!(vcat([[X[n] for i in 1:length(t_measure_dict["INL"][n])] for n
    ↪ in 1:length(X)] ... ),vcat(t_measure_dict["INL"] ... ), marker=:cross,
    ↪ color=:blue, markersize=3, label="INL data")
163 plot!(X, inl_mean,ribbon=(inl_lower,inl_upper), color=:blue, label="INL
    ↪ mean")
164 scatter!(vcat([[X[n] for i in 1:length(t_measure_dict["ONL"][n])] for n
    ↪ in 1:length(X)] ... ),vcat(t_measure_dict["ONL"] ... ),
    ↪ marker=:dtriangle, color=:green, markersize=3,
    ↪ markerstrokecolor=:green, label="ONL data")
165 plot!(X, onl_mean,ribbon=(onl_lower,onl_upper), color=:green, label="ONL
    ↪ mean")
166 annotate!([(15,.7,Plots.text("A",18))])
167
168 isl_ln_mts=[fit(MarginalTDist,log.(t_measure_dict["Isl"][n])) for n in
    ↪ 1:length(X)]
169 isl_mean=[exp(mean(mt)) for mt in isl_ln_mts]
170 isl_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in isl_ln_mts]
171 isl_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in isl_ln_mts]
172

```

```

173 islchart=scatter(vcat([[X[n] for i in 1:length(t_measure_dict["Isl"])[n]])
  ↵ for n in 1:length(X) ... ), vcat(t_measure_dict["Isl"] ... ),
  ↵ marker=:square, color=:darkmagenta, markerstrokecolor=:darkmagenta,
  ↵ markersize=3, label="Isl+ data", xticks=X,
  ↵ foreground_color_legend=nothing, background_color_legend=nothing,
  ↵ legend=:bottom, xformatter=_→ "", showaxis=:y)
174 plot!(X, isl_mean, ribbon=(isl_lower,isl_upper), color=:darkmagenta,
  ↵ label="Isl+ mean", ylims=[0,.5])
175 annotate!([(15,.05,Plots.text("B(+)"),18))])

176
177 pax6_n_mts=[fit(MarginalTDist,t_measure_dict["Pax6"][n]) for n in
  ↵ 1:length(X)]
178 pax6_mean=[mean(mt) for mt in pax6_n_mts]
179 pax6_lower=[mean(mt)-quantile(mt,.025) for mt in pax6_n_mts]
180 pax6_upper=[quantile(mt,.975)-mean(mt) for mt in pax6_n_mts]

181
182 pax6chart=scatter(vcat([[X[n] for i in
  ↵ 1:length(t_measure_dict["Pax6"])[n]]] for n in
  ↵ 1:length(X) ... ), vcat(t_measure_dict["Pax6"] ... ), marker=:circle,
  ↵ color=:darkmagenta, markersize=3, markerstrokecolor=:darkmagenta,
  ↵ label="Pax6+ data", xticks=X, foreground_color_legend=nothing,
  ↵ background_color_legend=nothing, legend=:top, xformatter=_→ "",
  ↵ showaxis=:y, ylabel="Fraction of GCL cohort labelled")
183 plot!(X, pax6_mean, ribbon=(pax6_lower,pax6_upper), color=:darkmagenta,
  ↵ label="Pax6+ mean", ylims=[0,.6])
184 annotate!([(15,.45,Plots.text("C(N)",18))])

185
186 islp_ln_mts=[fit(MarginalTDist,log.(t_measure_dict["Isl_Pax6"])[n])) for n
  ↵ in 1:length(X)]
187 islp_mean=[exp(mean(mt)) for mt in islp_ln_mts]
188 islp_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in islp_ln_mts]
189 islp_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in islp_ln_mts]

190
191 islpchart=scatter(vcat([[X[n] for i in
  ↵ 1:length(t_measure_dict["Isl_Pax6"])[n]]] for n in
  ↵ 1:length(X) ... ), vcat(t_measure_dict["Isl_Pax6"] ... ),
  ↵ marker=:triangle, color=:darkmagenta, markersize=3,
  ↵ markerstrokecolor=:darkmagenta, label="Isl/Pax6+ data", xticks=X,
  ↵ foreground_color_legend=nothing, background_color_legend=nothing,
  ↵ legend=:bottom, xlabel="Age (dpf)")
192 plot!(X, islp_mean, ribbon=(islp_lower,islp_upper), color=:darkmagenta,
  ↵ label="Isl/Pax6+ mean")
193 annotate!([(15,.05,Plots.text("D(+)"),18))])

```

```

194
195 gcl_subplot=plot(islchart, pax6chart, islpchart, layout=grid(3,1),
  ↵ link=:x)
196
197 pax6i_ln_mts=[fit(MarginalTDist,log.(t_measure_dict["INL_Pax6"][n])) for
  ↵ n in 1:length(X)]
198 pax6i_mean=[exp(mean(mt)) for mt in pax6i_ln_mts]
199 pax6i_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in pax6i_ln_mts]
200 pax6i_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in pax6i_ln_mts]
201
202 inlpax6chart=scatter(vcat([[X[n] for i in
  ↵ 1:length(t_measure_dict["INL_Pax6"][n])] for n in
  ↵ 1:length(X)] ... ),vcat(t_measure_dict["INL_Pax6"] ... ), marker=:square,
  ↵ color=:blue, markersize=3, markerstrokecolor=:blue, label="Pax6+
  ↵ data", xticks=X, foreground_color_legend=nothing,
  ↵ background_color_legend=nothing, legend=:bottom, xformatter=_→"",
  ↵ showaxis=:y)
203 plot!(X, pax6i_mean,ribbon=(pax6i_lower,pax6i_upper), color=:blue,
  ↵ label="Pax6+ mean")
204 annotate!([(15,.075,Plots.text("E(+)"),18))])
205
206 pkc_ln_mts=[fit(MarginalTDist,log.(t_measure_dict["PKCB"][n])) for n in
  ↵ 1:length(X)]
207 pkc_mean=[exp(mean(mt)) for mt in pkc_ln_mts]
208 pkc_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in pkc_ln_mts]
209 pkc_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in pkc_ln_mts]
210
211 pkcchart=scatter(vcat([[X[n] for i in
  ↵ 1:length(t_measure_dict["PKCB"][n])] for n in
  ↵ 1:length(X)] ... ),vcat(t_measure_dict["PKCB"] ... ), marker=:circle,
  ↵ color=:blue, markersize=3, markerstrokecolor=:blue, label="PKCβ+
  ↵ data", xticks=X, foreground_color_legend=nothing,
  ↵ background_color_legend=nothing, legend=:top, xformatter=_→"",
  ↵ showaxis=:y, ylabel="Fraction of INL cohort labelled")
212 plot!(X, pkc_mean,ribbon=(pkc_lower,pkc_upper), color=:blue, label="PKCβ+
  ↵ mean", ylims=[0,.2])
213 annotate!([(15,.18,Plots.text("F(+)"),18))])
214
215 gs_ln_mts=[fit(MarginalTDist,log.(t_measure_dict["GS"][n])) for n in
  ↵ 1:length(X)]
216 gs_mean=[exp(mean(mt)) for mt in gs_ln_mts]
217 gs_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in gs_ln_mts]
218 gs_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in gs_ln_mts]

```

```

219
220 gschart=scatter(vcat([[X[n] for i in 1:length(t_measure_dict["GS"])[n]])
  ↵ for n in 1:length(X) ... ), vcat(t_measure_dict["GS"] ... ),
  ↵ marker=:diamond, color=:blue, markersize=3, markerstrokecolor=:blue,
  ↵ label="GS+ data", xticks=X, foreground_color_legend=nothing,
  ↵ background_color_legend=nothing, legend=:topright, xformatter=_→"",
  ↵ showaxis=:y)
221 plot!(X, gs_mean, ribbon=(gs_lower,gs_upper), color=:blue, label="GS+
  ↵ mean")
222 annotate!([(15,.125,Plots.text("G(t)",18))])
223
224 hm_ln_mts=[fit(MarginalTDist,log.(t_measure_dict["HM"])[n])) for n in
  ↵ 1:length(X)]
225 hm_mean=[exp(mean(mt)) for mt in hm_ln_mts]
226 hm_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in hm_ln_mts]
227 hm_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in hm_ln_mts]
228
229 hmchart=scatter(vcat([[X[n] for i in 1:length(t_measure_dict["HM"])[n]])
  ↵ for n in 1:length(X) ... ), vcat(t_measure_dict["HM"] ... ),
  ↵ marker=:cross, color=:blue, markersize=3, markerstrokecolor=:blue,
  ↵ label="HM+ data", xticks=X, foreground_color_legend=nothing,
  ↵ background_color_legend=nothing, legend=:topright, xlabel="Age
  ↵ (dpf)")
230 plot!(X, hm_mean, ribbon=(hm_lower,hm_upper), color=:blue, label="HM+
  ↵ mean")
231 annotate!([(15,.13,Plots.text("H",18))])
232
233 inl_subplot=plot(inlpax6chart, pkcchart, gschart, hmchart,
  ↵ layout=grid(4,1), link=:x)
234
235 z_n_mts=[fit(MarginalTDist,t_measure_dict["Zpr1"])[n]) for n in
  ↵ 1:length(X)]
236 z_mean=[mean(mt) for mt in z_n_mts]
237 z_lower=[mean(mt)-quantile(mt,.025) for mt in z_n_mts]
238 z_upper=[quantile(mt,.975)-mean(mt) for mt in z_n_mts]
239
240 zchart=scatter(vcat([[X[n] for i in 1:length(t_measure_dict["Zpr1"])[n]])
  ↵ for n in 1:length(X) ... ), vcat(t_measure_dict["Zpr1"] ... ),
  ↵ marker=:square, color=:green, markerstrokecolor=:green, markersize=3,
  ↵ label="Zpr1+ data", xticks=X, foreground_color_legend=nothing,
  ↵ background_color_legend=nothing, legend=:bottom, xlabel="Age (dpf)",
  ↵ ylabel="Frac. OPL labelled")

```

```

241 plot!(X, z_mean, ribbon=(z_lower,z_upper), color=:green, label="HM+ mean",
242   ↪ ylims=[0,.55])
243 annotate!([(15,.15,Plots.text("I(N)",18))])
244
245 l=@layout [[layers{0.5h};gcl] [inl{0.8h};opl]]
246 combined=plot(layers_chart,gcl_subplot, inl_subplot,zchart,
247   ↪ size=(1200,1200),layout=l)
248 savefig(combined, "/bench/PhD/Thesis/images/cmz/layercontributions.png")
249
250
251
252
253
254
255
256
257
258 gmc=GMC_DEFAULTS
259 gmc[1]=5
260 gmc[2]=1.e-15
261
262 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]])
263 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
264
265 n_models=50
266
267 evdict=Dict{String,Measurement}()
268
269 max_mu=1.
270 min_mu=0.
271 max_lambda=4000.
272 min_lambda=1.
273
274 prior=[Uniform(min_mu,max_mu),Uniform(min_lambda,max_lambda)]
275 box=[min_mu max_mu;min_lambda max_lambda]

```

```

276
277 for ms in keys(t_measure_dict)
278     for (n,ovec) in enumerate(t_measure_dict[ms])
279         e_basepth="/bench/PhD/NGS_binaries/GMC_NS/A19/n_}*ms
280         x=X[n]
281         enspth=e_basepth*"/$x"
282         if isfile(enspth*"/ens")
283             e=deserialize(enspth*"/ens")
284         else
285             e=Normal_Ensemble(enspth,n_models,filter(i→i>0,ovec), prior,
286                             → box, gmc ... )
287         end
288
289         "n_}*ms in keys(evdict) ? (evdict["n_}*ms] +=
290         → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
291         → lower_displays=lds, disp_rot_its=10000,
292         → converge_factor=1e-6)) : (evdict["n_}*ms] =
293         → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
294         → lower_displays=lds, disp_rot_its=10000,
295         → converge_factor=1e-6))
296     end
297
298     for (n,ovec) in enumerate(t_measure_dict[ms])
299         e_basepth="/bench/PhD/NGS_binaries/GMC_NS/A19/ln_}*ms
300         x=X[n]
301         enspth=e_basepth*"/$x"
302         if isfile(enspth*"/ens")
303             e=deserialize(enspth*"/ens")
304         else
305             e=LogNormal_Ensemble(enspth,n_models,filter(i→i>0,ovec),
306                                   → prior, box, gmc ... )
307         end
308
309         "ln_}*ms in keys(evdict) ? (evdict["ln_}*ms] +=
310         → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
311         → lower_displays=lds, disp_rot_its=10000,
312         → converge_factor=1e-6)) : (evdict["ln_}*ms] =
313         → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
314         → lower_displays=lds, disp_rot_its=10000,
315         → converge_factor=1e-6))
316     end
317
318 end
319
320 end

```

```

305 for ms in keys(t_measure_dict)
306     nev=evdict["n_}*ms]
307     lnev=evdict["ln_}*ms]
308     ratio=lnev-nev
309     println("$ms & $(round(nev,digits=3)) & $(round(lnev,digits=3)) &
310             → $(round(ratio,digits=3)) & $(round(ratio.val/ratio.err,
311             → digits=1))\\\\\\ \\\\hline")
310 end
311
312 for ms in keys(t_measure_dict)
313     if ms in ["Pax6","INL","ONL","Zpr1"]
314         efunc=Normal_Engsemble
315         pf="cn_"
316     else
317         efunc=LogNormal_Engsemble
318         pf="cln_"
319     end
320
321     ovec=vcat(t_measure_dict[ms] ... )
322     e_basepth="/bench/PhD/NGS_binaries/GMC_NS/A19/*pf*ms
323
324     if isfile(e_basepth*/ens")
325         e=deserialize(e_basepth*/ens")
326     else
327         e=efunc(e_basepth,n_models,filter(i→i>0,ovec), prior, box,
328             → gmc ... )
329     end
330
331     evdict[pf*ms] =
332         → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
333         → lower_displays=lds, disp_rot_its=10000, converge_factor=1e-6)
334 end
335
336 for ms in keys(t_measure_dict)
337     if ms in ["Pax6","INL","ONL","Zpr1"]
338         pf="n_"
339     else
340         pf="ln_"
341     end
342
343     cev=evdict["c"*pf*ms]
344     sev=evdict[pf*ms]
345     ratio=cev-sev

```

```

343     println("$ms & $(round(cev,digits=3)) & $(round(sev,digits=3)) &
344         ↪ $(round(ratio,digits=3)) &
345         ↪ $(round(ratio.val/ratio.err,digits=1))\\\\\\ \\\\hline")
346 end
347
348 dv_evdict=Dict{String,Measurement}()
349
350 for ms in keys(d_measure_dict)
351     if ms in ["INL_Pax6","INL","Isl","Isl_Pax6","ONL","Zpr1"]
352         efunc=Normal_Ensemble
353         pf="d_cn_"
354     else
355         efunc=LogNormal_Ensemble
356         pf="d_cln_"
357     end
358
359     ovec=vcat(d_measure_dict[ms] ... )
360     e_basepth="/bench/PhD/NGS_binaries/GMC_NS/A19/*pf*ms
361
362     if isfile(e_basepth*/ens")
363         e=deserialize(e_basepth*/ens")
364     else
365         e=efunc(e_basepth,n_models,filter(i→!isnan(i),ovec), prior, box,
366             ↪ gmc ... )
367     end
368
369     dv_evdict[pf*ms] =
370         ↪ converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
371         ↪ lower_displays=lds, disp_rot_its=10000, converge_factor=1e-6)
372 end
373
374 for ms in keys(v_measure_dict)
375     if ms in ["INL_Pax6","INL","Isl","Isl_Pax6","ONL","Zpr1"]
376         efunc=Normal_Ensemble
377         pf="v_cn_"
378     else
379         efunc=LogNormal_Ensemble
380         pf="v_cln_"
381     end
382
383     ovec=vcat(d_measure_dict[ms] ... )
384     e_basepth="/bench/PhD/NGS_binaries/GMC_NS/A19/*pf*ms

```

```

381     if isfile(e_basepth*"/ens")
382         e=deserialize(e_basepth*"/ens")
383     else
384         e=efunc(e_basepth,n_models,filter(i→!isnan(i),ovec), prior, box,
385             ↵ gmc ... )
386     end
387
388     dv_evdict[pf*ms] =
389         → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
390             → lower_displays=lds, disp_rot_its=10000, converge_factor=1e-6)
391 end
392
393
394 for ms in keys(t_measure_dict)
395     if ms in ["INL_Pax6","INL","Isl","Isl_Pax6","ONL","Zpr1"]
396         pf="n_"
397     else
398         pf="ln_"
399     end
400
401     cev=dv_evdict["c"*pf*ms]
402     dev=dv_evdict["d_c"*pf*ms]
403     vev=dv_evdict["v_c"*pf*ms]
404     sev=dev+vev
405     ratio=cev-sev
406     println("$ms & $(round(cev,digits=3)) & $(round(sev,digits=3)) &
407         → $(round(ratio,digits=3)) &
408         → $(round(ratio.val/ratio.err,digits=1))\\\\ \\hline")
409 end

```

---

### 17.8.11 /cmz/a20dvratio.jl

---

```

1 using BayesianLinearRegression,CSV,DataFrames,Distributions, GMC_NS,
2   ↵ Serialization, Plots, NGRefTools
3 a20pth="/bench/PhD/Thesis/datasets/a20.csv"
4
5 a20df=DataFrame(CSV.read(a20pth))
6 a20df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
7   ↵ eachrow(a20df)]
8
9 X=Vector{Float64}(){}
10 measure_dict=Dict{String,Vector{Vector{Float64}}}(){}

```

```

9
10 measure_dict["Nasal CMZ (#)"] = Vector{Vector{Float64}}(){}
11 measure_dict["Temporal CMZ (#)"] = Vector{Vector{Float64}}(){}
12 measure_dict["Ratio"] = Vector{Vector{Float64}}(){}
13
14 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
15 for t_df in groupby(a20df, "Time point (d)")
16     push!(X, Float64(unique(t_df."Time point (d)")[1]))
17     rvec, nvec, tvec = [zeros(0) for i in 1:3]
18
19     for i_df in groupby(t_df, "BSR")
20         length(skipmissing(i_df."Nasal CMZ (#)")) > 0 &&
21             mean(skipmissing(i_df."Nasal CMZ (#)")) > 0. &&
22             push!(nvec, mean(skipmissing(i_df."Nasal CMZ (#)")))
23         length(skipmissing(i_df."Temporal CMZ (#)")) > 0 &&
24             mean(skipmissing(i_df."Temporal CMZ (#)")) > 0. &&
25             push!(tvec, mean(skipmissing(i_df."Temporal CMZ (#)")))
26         length(skipmissing(i_df."Nasal CMZ (#)")) > 0 &&
27             mean(skipmissing(i_df."Nasal CMZ (#)")) > 0. &&
28             length(skipmissing(i_df."Temporal CMZ (#)")) > 0 &&
29             mean(skipmissing(i_df."Temporal CMZ (#)")) > 0. &&
30             push!(rvec, mean(skipmissing(i_df."Nasal CMZ (#)")) / mean(skipmissing(i_df."Temporal CMZ (#)")))
31     end
32
33     push!(measure_dict["Ratio"], rvec)
34     push!(measure_dict["Nasal CMZ (#)"], nvec)
35     push!(measure_dict["Temporal CMZ (#)"], tvec)
36 end
37
38 n_login_mts=[fit(MarginalTDist, log.(measure_dict["Nasal CMZ (#)"][n])) for
39   n in 1:length(X)]
40 n_mean=[exp(mean(mt)) for mt in n_login_mts]
41 n_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in n_login_mts]
42 n_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in n_login_mts]
43
44 t_login_mts=[fit(MarginalTDist, log.(measure_dict["Temporal CMZ (#)"][n])) for
45   n in 1:length(X)]
46 t_mean=[exp(mean(mt)) for mt in t_login_mts]
47 t_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in t_login_mts]
48 t_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in t_login_mts]

```

```

40 r_n_mts=[fit(MarginalTDist,measure_dict["Ratio"][n]) for n in
41   1:length(X)]
42 r_mean=[mean(mt) for mt in r_n_mts]
43 r_lower=[mean(mt)-quantile(mt,.025) for mt in r_n_mts]
44 r_upper=[quantile(mt,.975)-mean(mt) for mt in r_n_mts]
45 plotticks=[30*i for i in 1:12]
46
47 nt_chart=Plots.plot(X, n_mean, ribbon=(n_lower,n_upper), color=:green,
48   label="Nasal mean", ylabel="CMZ sectional population size",
49   showaxis=:y, xticks=plotticks, xformatter=_→"")
50 plot!(X, t_mean, ribbon=(t_lower,t_upper), color=:darkmagenta,
51   label="Temporal mean")
52 scatter!(vcat([[X[n] for i in 1:length(measure_dict["Nasal CMZ (#)"][n])...
53   for n in 1:length(X)]...]),vcat(measure_dict["Nasal CMZ (#)"]...),
54   marker=:utriangle, color=:green, markersize=3,
55   markerstrokecolor=:green, label="Nasal data")
56 scatter!(vcat([[X[n] for i in 1:length(measure_dict["Temporal CMZ
57   (#)"][n])] for n in 1:length(X)]...),vcat(measure_dict["Temporal CMZ
58   (#)"]...), marker=:dtriangle, color=:darkmagenta, markersize=3,
59   markerstrokecolor=:darkmagenta, label="Temporal data")
60 annotate!([(1.5,125,Plots.text("A",18))])
61
62 ratio_chart=scatter(vcata[[X[n] for i in
63   1:length(measure_dict["Ratio"][n])] for n in
64   1:length(X)]...),vcat(measure_dict["Ratio"]...), marker=:cross,
65   color=:black, markersize=3, markerstrokecolor=:black, label="Ratio
66   data", ylabel="Individual asymmetry ratio", xlabel="Age (dpf)",
67   xticks=plotticks)
68 plot!(X, r_mean, ribbon=(r_lower,r_upper), color=:green, label="Ratio
69   mean")
70 plot!(X, [1. for i in 1:length(X)], style=:dot, color=:black, label="Even
71   ratio")
72 annotate!([(10,.5,Plots.text("B",18))])
73
74 l=@layout [a{.6h}
75           b]
76
77 combined=Plots.plot(nt_chart,ratio_chart,layout=l, link=:x,
78   size=(900,600))
79
80 savefig(combined, "/bench/PhD/Thesis/images/cmz/NTontology.png")

```

### 17.8.12 /cmz/a25dvlinreg.jl

```
1 using Distributions, Serialization, CSV, BayesianLinearRegression, Plots,
2   ↵  DataFrames, Measurements
3 import Measurements:value,uncertainty
4
5 a25path="/bench/PhD/datasets/a25.csv"
6 a25df=DataFrame(CSV.File(a25path))
7
8 X=Vector{Float64}()
9 yD=Vector{Vector{Float64}}()
10 yV=Vector{Vector{Float64}}()
11
12 for timedf in groupby(a25df,"Time")
13     push!(X,timedf[1,"Time"])
14     dvec=Vector{Float64}()
15     vvec=Vector{Float64}()
16     for row in eachrow(timedf)
17         dv=row."D/V"
18         if dv == "D"
19             push!(dvec, row.EdU/row.PCNA)
20         else
21             push!(vvec, row.EdU/row.PCNA)
22         end
23     end
24
25     push!(yD,dvec)
26     push!(yV,vvec)
27 end
28
29 sp=sortperm(X)
30 yD=yD[sp]; yV=yV[sp]
31 sort!(X)
32
33 xYD=zeros(0); xYV=zeros(0)
34 YD=zeros(0); YV=zeros(0)
35
36 for (n,vec) in enumerate(yD)
37     global xYD=vcat(xYD,[X[n] for i in 1:length(vec)])
38     global YD=vcat(YD,vec)
```

```

39 end
40
41 for (n,vec) in enumerate(yV)
42     global xYV=vcat(xYV,[X[n] for i in 1:length(vec)])
43     global YV=vcat(YV,vec)
44 end
45
46 xYD=hcat(ones(length(xYD)),xYD)
47 xYV=hcat(ones(length(xYV)),xYV)
48
49 xY=vcat(xYD,xYV);Y=vcat(YD,YV)
50
51 dm=BayesianLinearRegression.fit!(BayesianLinReg(xYD,YD))
52 vm=BayesianLinearRegression.fit!(BayesianLinReg(xYV,YV))
53 tm=BayesianLinearRegression.fit!(BayesianLinReg(xY,Y))
54
55 println("Dorsal model logZ: $(logEvidence(dm))")
56 println("Ventral model logZ: $(logEvidence(vm))")
57 println("Joint D/V logZ: $(logEvidence(dm) + logEvidence(vm)) ")
58 println("Combined model logZ: $(logEvidence(tm))")
59 println("Combined/Joint ratio : $(logEvidence(tm)-(logEvidence(dm) +
60   ↪ logEvidence(vm)))")
61
62 function blr_plt(x,y,Xs,blrs,colors,labels,q=.975)
63
64     ↪ plt=scatter(x,y,marker=:cross,markersize=3,markercolor=:black,label="Data",yl
65     ↪ labelled fraction", xlabel="Pulse time (hr)", legend=:bottomright,
66     ↪ ylims=(-0.1,1.))
67     for (X,blr,color,label) in zip(Xs, blrs,colors,labels)
68         line=zeros(size(X,1))
69         ribbon=zeros(size(X,1))
70         predictions=BayesianLinearRegression.predict(blr,X)
71         for (n,p) in enumerate(predictions)
72             normal=Normal(value(p),uncertainty(p))
73             ribbon[n]=quantile(normal,q)-mean(normal)
74             line[n]=mean((value(p)))
75         end
76         plot!(plt,x,line,ribbon=ribbon,color=color,label=label)
77     end
78     return plt
79 end
80
81 dplt=blr_plt(xYD[:,2],YD,[xYD],[dm],[green],"Dorsal model")

```

```

78 annotate!([(1,.9,Plots.text("A",12))])
79 vplt=blr_plt(xYV[:,2],YV,[xYV],[vm],[::magenta],["Ventral model"])
80 annotate!([(1,.9,Plots.text("B",12))])
81 tplt=blr_plt(xY[:,2],Y,[xY],[tm],[::black],["Combined model"])
82 annotate!([(1,.9,Plots.text("C",12))])

83
84 combined=plt(dplt,vplt,tplt,layout=(3,1), size=(600,900))
85
86 savefig(combined, "/bench/PhD/Thesis/images/cmz/3ddvlinreg.png")
87 @info "Saved fig."
88
89
90 println("\\begin{tabular}{l|l|l|l}")
91 println("\\hline")
92 println("{\\bf Model} & {\\bf Implied \$T_c\$} & {\\bf Implied \$T_s\$} &
93   \\bf logZ}\\\\ \\hline")
94 for (name,model) in zip(["Dorsal","Ventral","Combined"],[dm,vm,tm])
95   local intercept,slope=posteriorWeights(model)
96   println("$name & $(1/slope) & $(intercept*1/slope) &
97     $(round(logEvidence(model),digits=3))\\\\ \\hline")
98 end
99 println("\\end{tabular}")

```

---

### 17.8.13 /cmz/a25thymidinesim.jl

```

1 using GMC_NS, CMZNicheSims, ConjugatePriors, CSV, DataFrames,
2   Distributions, Serialization, NGRefTools
3
4 a10path="/bench/PhD/datasets/A10 measurements 2018update.csv"
5 a10df=DataFrame(CSV.File(a10path))
6
7 df3=a10df[a10df."Time point (d)" .== 3, :]
8
9 inddict=Dict{Tuple,Vector}()
10 for row in eachrow(df3)
11   ind=(row.Block, row.Slide, row.Row)
12   if !(ind in keys(inddict))
13     inddict[ind]=[(row["Dorsal CMZ (#)"]+row["Ventral CMZ (#)"])]
14   else

```

```

14     push!(inddict[ind],((row["Dorsal CMZ (#)"]+row["Ventral CMZ
15         ↵      (#)"]))
16 end
17
18 popvec=Vector{Float64}(){}
19 for (ind, pops) in inddict
20     push!(popvec,mean(pops))
21 end
22
23 a25path="/bench/PhD/datasets/a25.csv"
24 a25df=DataFrame(CSV.File(a25path))
25
26 X=Vector{Float64}(){}
27 y=Vector{Vector{Int64}}(){}
28
29 for timedf in groupby(a25df,"Time")
30     push!(X,timedf[1,"Time"])
31     dict=Dict{Tuple,Vector{Int64}}(){}
32     for row in eachrow(timedf)
33         ind=(row.Block,row.Slide,row.Row)
34         if !(ind in keys(dict))
35             dict[ind]=[row.PCNA,row.EdU]
36         else
37             dict[ind][1]+=row.PCNA
38             dict[ind][2]+=row.EdU
39         end
40     end
41
42     tcounts=Vector{Int64}(){}
43     for (ind,counts) in dict
44         global popvec=vcat(popvec, counts[1])
45         push!(tcounts, counts[2])
46     end
47     push!(y,tcounts)
48 end
49
50 y=y[sortperm(X)]
51 sort!(X)
52
53 pop_dist=fit(LogNormal,popvec)
54
55 tc_prior=NormalInverseGamma(3.5,1.,5.,2.)

```

```

56 g1_prior=Beta(2.,2.)
57 s_prior=Beta(2.,5.)
58 sister_prior=Beta(8.,75.)
59
60 tc_prior2=NormalInverseGamma(3.75,1.,5.,2.)
61 pop2_prior=Beta(1.1,10.)
62
63 priors_1_pop=[marginals(tc_prior) ... , g1_prior, s_prior, sister_prior]
64 priors_2_pop=[marginals(tc_prior2) ... , g1_prior, s_prior,
   ↵ sister_prior,pop2_prior]
65
66 priors_2_pop=vcat(priors_1_pop,priors_2_pop)
67
68 p1_box=[2. 8.
   .1 2
   eps() 1.
   eps() 1.
   eps() .5]
69
70 p2_box=vcat(p1_box,vcat(p1_box,[eps() 1.]))
71
72 p1_box=GMC_NS.to_unit_ball.(p1_box,priors_1_pop)
73 p2_box=GMC_NS.to_unit_ball.(p2_box, priors_2_pop)
74
75
76 ep1="/bench/PhD/NGS_binaries/CNS/A25/1pop"
77 ep2="/bench/PhD/NGS_binaries/CNS/A25/2pop"
78 const pulse_time=10.5
79 const mc_its=Int64(1e5)
80 const end_time=10.5
81 constants=[pop_dist, X, pulse_time, mc_its, end_time]
82
83 for (ep,priors,box) in
   ↵ zip([ep1,ep2],[priors_1_pop,priors_2_pop],[p1_box,p2_box])
84   if isfile(ep*"/ens")
85     e=deserialize(ep*"/ens")
86   else
87     @info "Assembling ensemble at $ep"
88     gmcd=GMC_DEFAULTS
89     gmcd[1]=5
90     gmcd[end]=100
91     e=Thymidine_Ensemble(ep, 100, y, priors, constants, box, gmcd)
92   end
93
94
95
96

```

```

97      ↵  uds=Vector{Vector{Function}}([[liwi_display],[convergence_display],[ensemble_])
98      ↵  lds=Vector{Vector{Function}}([[model_obs_display],[model_obs_display],[model_])
99
100    converge_ensemble!(e,backup=(true,1),upper_displays=uds,
101      ↵  lower_displays=lds, disp_rot_its=5, mc_noise=.14,
102      ↵  converge_factor=1e-3)
103  end
104
105
106 e1=deserialize(ep1*"/ens")
107 e2=deserialize(ep2*"/ens")
108
109 CMZNicheSims.print_MAP_output(e1,
110   ↵  "/bench/PhD/Thesis/images/cmz/a25MAP.png", its=Int64(1e7))
111 CMZNicheSims.print_marginals(e1,"/bench/PhD/Thesis/images/cmz/a25marginals.png")

```

---

### 17.8.14 /cmz/a27GMC\_NS.jl

---

```

1 using Distributions, Serialization, CSV, BayesianLinearRegression, Plots,
2   ↵  DataFrames, Measurements, GMC_NS
2 import Measurements:value,uncertainty
3
4 a27path="/bench/PhD/datasets/a27_r.csv"
5 a27df=DataFrame(CSV.File(a27path))
6
7 X=unique(a27df["Age"]).*30.
8 CR=Dict{Float64,Vector{Float64}}(){}
9 mo1=Dict{Float64,Vector{Float64}}{}
10
11 for x in X
12   CR[x]=Vector{Float64}(){}
13   x<90. && (mo1[x]=Vector{Float64}{}())
14 end
15
16 for row in eachrow(a27df)
17   x=row["Age"]*30.
18   !ismissing(row["LR"]) && push!(CR[x],row["LR"])

```

```

19     !ismissing(row."D/N1") && !ismissing(row."V/T1") &&
20     → push!(mo1[x],sum([row."D/N1",row."V/T1"]))
21 end
22
23 gmc=GMC_DEFAULTS
24 gmc[1]=5
25 gmc[2]=1.e-15
26
27 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]]
28 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
29
30 n_models=50
31
32 evdict=Dict{String,Measurement}()
33
34 max_μ=2000.
35 min_μ=0.
36 max_λ=500.
37 min_λ=1e-6
38
39 prior=[Uniform(min_μ,max_μ),Uniform(min_λ,max_λ)]
40 box=[min_μ max_μ;min_λ max_λ]
41
42 cobs=vcat([v for v in values(CR)]...)
43 mobs=vcat([v for v in values(mo1)]...)
44
45 for (md, pstring) in zip([CR,mo1],["CR","mo1"])
46     for (k,v) in md
47         pfx=pstring*string(k)
48         enspth="/bench/PhD/NGS_binaries/GMC_NS/A27/*pfx
49         if isfile(enspth*"/ens")
50             e=deserialize(enspth*"/ens")
51         else
52             e=LogNormal_Ensemble(enspth,n_models,v, prior, box, gmc...)
53         end
54

```

```

55     pstring in keys(evdict) ?
56         (evdict[pstring]+=converge_ensemble!(e,backup=(true,10000),upper_displays=
57             lower_displays=lds, disp_rot_its=10000,
58             converge_factor=1e-6)) :
59         (evdict[pstring]=converge_ensemble!(e,backup=(true,10000),upper_displays=
60             lower_displays=lds, disp_rot_its=10000,
61             converge_factor=1e-6))
62     end
63 end
64
65 for (obs,pstring) in zip([cobs,mobs],["cCR","cmo1"])
66     enspth="/bench/PhD/NGS_binaries/GMC_NS/A27/*pstring
67     if isfile(enspth*"/ens")
68         e=deserialize(enspth*"/ens")
69     else
70         e=LogNormal_Ensemble(enspth,n_models,obs, prior, box, gmc ... )
71     end
72
73     evdict[pstring]=converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
74         lower_displays=lds, disp_rot_its=10000, converge_factor=1e-6)
75 end
76
77 for ms in ["CR","mo1"]
78     cev=evdict["c"*ms]
79     sev=evdict[ms]
80     ratio=cev-sev
81     println("$ms & $(round(cev,digits=3)) & $(round(sev,digits=3)) &
82         $(round(ratio,digits=3)) &
83         $(round(ratio.val/ratio.err,digits=1))\\\\ \\hline")
84 end

```

---

### 17.8.15 /cmz/a27linreg.jl

---

```

1 using Distributions, Serialization, CSV, BayesianLinearRegression, Plots,
2     DataFrames, Measurements
3 import Measurements:value,uncertainty
4
5 a27path="/bench/PhD/datasets/a27_r.csv"
6 a27df=DataFrame(CSV.File(a27path))
7 X=unique(a27df["Age"]).*30.

```

```

8 CR=Dict{Float64,Vector{Float64}} }()
9 mo1=Dict{Float64,Vector{Float64}} ()

10
11 for x in X
12     CR[x]=Vector{Float64}()
13     x<90. && (mo1[x]=Vector{Float64}())
14 end
15
16 for row in eachrow(a27df)
17     x=row."Age"*30.
18     !ismissing(row."LR") && push!(CR[x],row."LR")
19     !ismissing(row."D/N1") && !ismissing(row."V/T1") &&
20         → push!(mo1[x],sum([row."D/N1",row."V/T1"]))
21 end
22 xCR=zeros(0); xmo1=zeros(0)
23 yCR=zeros(0); ymo1=zeros(0)
24
25 for (age,vec) in CR
26     println([age for i in 1:length(vec)])
27     global xCR=vcat(xCR,[age for i in 1:length(vec)])
28     global yCR=vcat(yCR,vec)
29 end
30
31 for (time,vec) in mo1
32     global xmo1=vcat(xmo1,[time for i in 1:length(vec)])
33     global ymo1=vcat(ymo1,vec)
34 end
35
36 xCR=hcat(ones(length(xCR)),xCR)
37 xmo1=hcat(ones(length(xmo1)),xmo1)
38
39 uncorrCR2=BayesianLinearRegression.fit!(BayesianLinReg(ones(length(yCR),1),yCR))
40 corrCR2=BayesianLinearRegression.fit!(BayesianLinReg(xCR,yCR))
41 uncorrCRX=ones(length(X),1)
42 corrCRX=hcat(ones(length(X)),X)
43
44 uncorrmo1=BayesianLinearRegression.fit!(BayesianLinReg(ones(length(ymo1),1),ymo1))
45 corrmo1=BayesianLinearRegression.fit!(BayesianLinReg(xmo1,ymo1))
46 uncorrmo1X=ones(length(X[1:2]),1)
47 corrmo1X=hcat(ones(length(X[1:2])),X[1:2])
48
49 function blr_plt(i,x,y,Xs,blrs,colors,labels,q=.975)

```

```

50
    ↵ plt=scatter(x,y,marker=:diamond,markersize=8,markerstrokestyle=:bold,markerco
    ↵ size", xlabel="Age
    ↵ (dpf)", legend=:topright, ylims=[0,maximum(y)+25])
51 for (X,blr,color,label) in zip(xs, blrs,colors,labels)
52     line=zeros(size(X,1))
53     ribbon=zeros(size(X,1))
54     predictions=BayesianLinearRegression.predict(blr,X)
55     for (n,p) in enumerate(predictions)
56         normal=Normal(value(p),uncertainty(p))
57         ribbon[n]=quantile(normal,q)-mean(normal)
58         line[n]=mean((value(p)))
59     end
60     plot!(plt,i,line,ribbon=ribbon,color=color,label=label)
61 end
62 return plt
63 end
64
65 pCR=blr_plt(X,xCR[:,2],yCR,[corrCRX,uncorrCRX],[corrCR2,uncorrCR2],[magenta,:green],
    ↵ "Model - CR", "Uncorrelated Model - CR"])
66 annotate!([(35,250,Plots.text("A",18))])
67
68 pmo1=blr_plt(X[1:2],xmo1[:,2],ymo1,[corrmo1X,uncorrmo1X],[corrmo1,uncorrmo1],[magenta,
    ↵ "Model - 1mo", "Uncorrelated Model - 1mo CMZ"])
69 annotate!([(32,350,Plots.text("B",18))])
70
71 combined=plot(pCR,pmo1,layout=grid(2,1),size=(600,800))
72 savefig(combined,"/bench/PhD/Thesis/images/cmz/a27linreg.png")
73
74 println("\\begin{tabular}{l|l|l|l}")
75 println("\\hline")
76 println("{\\bf Measurement} & {\\bf Uncorrelated logZ} & {\\bf Correlated
    ↵ logZ} & {\\bf logZR}\\\"\\\"\\hline")
77 println("1dpf Central Remnant & {\bf
    ↵ $(round(logEvidence(uncorrCR2),digits=3))} &
    ↵ $(round(logEvidence(corrCR2),digits=3)) &
    ↵ $(round(logEvidence(uncorrCR2)-logEvidence(corrCR2),digits=3))\\\"\\"
    ↵ \\hline")
78 println("30dpf Cohort & {\bf $(round(logEvidence(uncorrmo1),digits=3))} &
    ↵ & $(round(logEvidence(corrmo1),digits=3)) &
    ↵ $(round(logEvidence(uncorrmo1)-logEvidence(corrmo1),digits=3))\\\"\\"
    ↵ \\hline")
79 println("\\end{tabular}")

```

### 17.8.16 /rys/a35thymidinesim.jl

```

1  using GMC_NS, CMZNicheSims, ConjugatePriors, CSV, DataFrames,
   ↵ Distributions, Serialization, NGRefTools, KernelDensity, Plots,
   ↵ StatsPlots
2  import BioBackgroundModels: lps
3  a35path="/bench/PhD/Thesis/datasets/A35.csv"
4  a35df=DataFrame(CSV.File(a35path))
5
6  X=Vector{Float64}()
7  s_popvec=Vector{Float64}()
8  r_popvec=Vector{Float64}()
9  sib_y=Vector{Vector{Int64}}()
10rys_y=Vector{Vector{Int64}}()

11
12 for timedf in groupby(a35df,"Time")
13     push!(X,timedf[1,"Time"])
14     r_dict=Dict{Tuple,Vector{Int64}}(){}
15     s_dict=Dict{Tuple,Vector{Int64}}(){}
16     for row in eachrow(timedf)
17         if !ismissing(row.PCNA) && !ismissing(row.EdU)
18             row.Treatment=="Sib" ? (dict=s_dict; y=sib_y) : (dict=r_dict;
19                         ↵ y=rys_y)
20             ind=(row.Block,row.Slide,row.Row)
21             if !(ind in keys(dict))
22                 dict[ind]=[row.PCNA,row.EdU]
23             else
24                 dict[ind][1]+=row.PCNA
25                 dict[ind][2]+=row.EdU
26             end
27         end
28
29         rcounts=Vector{Int64}()
30         scounts=Vector{Int64}()
31         for (dict,ctvec,pvec) in
32             ↵ zip([r_dict,s_dict],[rcounts,scounts],[r_popvec,s_popvec])
33             for (ind,counts) in dict
34                 push!(pvec, counts[1])
35                 push!(ctvec, counts[2])
36             end

```

```

36     end
37
38     push!(sib_y,scounts)
39     push!(rys_y,rcounts)
40 end
41
42 sib_y=sib_y[sortperm(X)]
43 rys_y=rys_y[sortperm(X)]
44 sort!(X)
45
46 s_pop_prior=fit(NormalInverseGamma,log.(s_popvec))
47 r_pop_prior=fit(NormalInverseGamma,log.(r_popvec))
48
49 g1_prior=Normal(4.,2.)
50 tc_prior=NormalInverseGamma(3.5,1.,5.,2.)
51 s_prior=NormalInverseGamma(4.,2.,1.,1.)
52 sister_prior=Beta(8.,75.)
53
54 s_priors=[marginals(s_pop_prior) ... , g1_prior, marginals(tc_prior) ... ,
55           ↳ marginals(s_prior) ... , sister_prior]
55 r_priors=[marginals(r_pop_prior) ... , g1_prior, marginals(tc_prior) ... ,
56           ↳ marginals(s_prior) ... , sister_prior]
56
57 box=[3.3 5.8
58       .07 .26
59       1. 10.
60       .1 10.
61       eps() 2.7
62       eps() 20.
63       .1 200.
64       eps() .5]
65
66 s_box=GMC_NS.to_unit_ball.(box,s_priors)
67 r_box=GMC_NS.to_unit_ball.(box,s_priors)
68
69 sp="/bench/PhD/NGS_binaries/CNS/A35/sib"
70 rp="/bench/PhD/NGS_binaries/CNS/A35/rys"
71 const pulse_time=10.5
72 const mc_its=Int64(1.5e5)
73 const end_time=10.5
74 constants=[X, pulse_time, mc_its, end_time]
75

```

```

76 for (ep, priors, bx, y) in zip([sp, rp], [s_priors, r_priors],
77   → [s_box, r_box], [sib_y, rys_y])
78   if isfile(ep*/ens")
79     e=deserialize(ep*/ens")
80   else
81     @info "Assembling ensemble at $ep"
82     gmcd=GMC_DEFAULTS
83     gmcd[1]=5
84     e=Thymidine_Ensemble(ep, 100, y, priors, constants, bx, gmcd)
85   end
86
87   → uds=Vector{Vector{Function}}([[liwi_display],[convergence_display],[evidence_
88   → lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display],[ensembl
89   converge_ensemble!(e,backup=true,1),upper_displays=uds,
90   → lower_displays=lds, disp_rot_its=5, mc_noise=.14,
91   → converge_factor=1e-3)
92 end
93 se=deserialize(sp*/ens")
94 re=deserialize(rp*/ens")
95 measure_evidence(se)
96 measure_evidence(re)
97
98 CMZNicheSims.print_MAP_output(se,"/bench/PhD/Thesis/images/rys/a35sibMAP.png",
99   → its=Int64(1e7))
100 CMZNicheSims.print_MAP_output(re,"/bench/PhD/Thesis/images/rys/a35rysMAP.png",
101   → its=Int64(1e7))
102
103 lt=length(se.priors)
104 swt=zeros(0); rwt=zeros(0)
105 sθ=[zeros(0) for param in 1:lt]
106 rθ=[zeros(0) for param in 1:lt]
107
108 for (e,wtvec,θvecs) in zip([se,re],[swt,rwt],[sθ,rθ])
109   for (n,rec) in enumerate(e.posterior_samples)
110     m=deserialize(rec.path)
111     for i in 1:length(θvecs)
112       push!(θvecs[i],m.θ[i])
113     end

```

```

112     push!(wtvec,e.log_Liwi[n+1])
113   end
114
115   for rec in e.models
116     m=deserialize(rec.path)
117     for i in 1:length(θvecs)
118       push!(θvecs[i],m.θ[i])
119     end
120
121     → push!(wtvec,(lps(m.log_Li,lps(e.log_Xi[end],-log(length(e.models))))))
122   end
123
124 deleteat!(sθ,3); deleteat!(sθ,4)
125 deleteat!(rθ,3); deleteat!(rθ,4)
126
127 n_pops=1; plotrows=5; param_names=[ "LogNormal Tc  $\mu$ ", "LogNormal Tc  $\sigma^2$ ",
128   ← "G1 Fraction", "S Fraction", "Sister Shift Fraction" ]
129
130 maps=deserialize(se.models[findmax([m.log_Li for m in
131   ← se.models])[2]].path)
132 mapr=deserialize(re.models[findmax([m.log_Li for m in
133   ← re.models])[2]].path)
134
135 plots=Vector{Plots.Plot}()
136 for (n,(sθvec,rθvec)) in enumerate(zip(sθ,rθ))
137   n==3 && (n=4)
138   p_label=param_names[n]
139   occursin("Fraction",p_label) ? (xls=[0,1]) :
140     ← (xls=[quantile(se.priors[n],.01),quantile(se.priors[n],.99)])
141
142 sθkde=kde(sθvec,weights=exp.(swt))
143 rθkde=kde(rθvec,weights=exp.(rwt))
144 n==1 ? (lblpos=:right) : (lblpos=:none)
145 plt=StatsPlots.plot(se.priors[n], color=:darkmagenta, fill=true,
146   ← fillalpha=.5, label="Prior", xlabel=p_label, ylabel="Density",
147   ← xlims=xls)
148 plot!(sθkde.x,sθkde.density, color=:green, fill=true, fillalpha=.5,
149   ← label="Sib Posterior", legend=lblpos)
150
151 → plot!([maps.θ[n],maps.θ[n]],[0,maximum(sθkde.density)],color=:darkgreen,
152   ← label="sib MAP", legend=lblpos)

```

```

145 plot!(rθkde.x,rθkde.density, color=:orange, fill=true, fillalpha=.5,
    ↳ label="Rys Posterior", legend=lblpos)
146 ↳ plot!([mapr.θ[n],mapr.θ[n]],[0,maximum(rθkde.density)],color=:darkred,
    ↳ label="Rys MAP", legend=lblpos)
147
148 push!(plots,plt)
149 n_pops>1 && n==5 && push!(plots,plot())
150 end
151
152 combined=plot(plots ... , layout=grid(3,n_pops),size=(600*n_pops,800))
153
154 savefig(combined, "/bench/PhD/Thesis/images/rys/a35marginals.png")

```

---

### 17.8.17 /rys/a37.jl

```

1 using CSV,DataFrames,Distributions,Measurements,GMC_NS, NGRefTools,
    ↳ Plots,Measures, Serialization
2 gr()
3 default(fontsize = 10, guidefont = (12,:bold), tickfont = 10, guide =
    ↳ "x")
4
5
6 pth="/bench/PhD/Thesis/datasets/a37.csv"
7
8 a37df=DataFrame(CSV.read(pth))
9
10 atg_measure_dict=Dict{String,Vector{Float64}}(){}
11 spl_measure_dict=Dict{String,Vector{Float64}}(){}
12 ctl_measure_dict=Dict{String,Vector{Float64}}(){}
13 uninj_measure_dict=Dict{String,Vector{Float64}}(){}
14
15 msd=[atg_measure_dict,spl_measure_dict,ctl_measure_dict,uninj_measure_dict]
16
17 for dict in msd
18     dict["DCMZ"]=Vector{Float64}(){}
19     dict["VCMZ"]=Vector{Float64}(){}
20     dict["TCMZ"]=Vector{Float64}(){}
21     dict["NucPop"]=Vector{Float64}(){}
22     dict["Sphericity"]=Vector{Float64}(){}
23     dict["Volume"]=Vector{Float64}(){}
24 end

```

```

25
26 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
27 for (gdf,dict) in zip(groupby(a37df, "Group"),msd)
28     for row in eachrow(gdf)
29         if row."D/V"=="D"
30             push!(dict["DCMZ"],row."PCNA +ve")
31             !ismissing(row."Total PCNA") && push!(dict["TCMZ"],row."Total
32             ↪ PCNA")
33             !ismissing(row."Total nuclei") &&
34             ↪ push!(dict["NucPop"],row."Total nuclei")
35             push!(dict["Sphericity"],row."Sphericity")
36             push!(dict["Volume"],row."Volume")
37         else
38             push!(dict["VCMZ"],row."PCNA +ve")
39
40             ↪ dict["Sphericity"][[end]]+=row."Sphericity";dict["Sphericity"][[end]]/=2
41             dict["Volume"][[end]]+=row."Volume";dict["Volume"][[end]]/=2
42         end
43     end
44 end
45
46 for md in msd
47     md["NucPer"] = md["NucPop"] ./ md["TCMZ"]
48 end
49
50 gmc=GMC_DEFAULTS
51 gmc[1]=5
52 gmc[2]=1.e-15
53 gmcdir="/bench/PhD/NGS_binaries/BSS/A37"
54
55 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]])
56 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
57
58 n_models=50
59
60 max_mu=300.;min_mu=0.
61 max_lambda=1.;min_lambda=1e-5
62 prior=[Uniform(min_mu,max_mu),Uniform(min_lambda,max_lambda)]
63 box=[min_mu max_mu;min_lambda max_lambda]
64 np_prior=[Uniform(min_mu,5000),Uniform(min_lambda,max_lambda)]

```

```

65 np_box=[min_μ 5000;min_λ max_λ]
66
67 s_max_μ=1.;s_min_μ=0.
68 s_max_λ=5e5;s_min_λ=1e3
69 s_prior=[Uniform(s_min_μ,s_max_μ),Uniform(s_min_λ,s_max_λ)]
70 s_box=[s_min_μ s_max_μ;s_min_λ s_max_λ]
71
72 totalmsvec=[ "TCMZ", "NucPop", "NucPer", "DCMZ", "VCMZ", "Volume", "Sphericity" ]
73 popmsvec=[ "TCMZ", "NucPop", "NucPer", "DCMZ", "VCMZ" ]
74 normmsvec=[ "Volume", "Sphericity" ]
75
76 for (md,morph) in zip([atg_measure_dict,spl_measure_dict],[ "ATG", "Spl" ])
    evdict=Dict{String,Measurement}()
77
78
79     for ms in totalmsvec
        evdict[ms*"Combined"]=measurement(0,0)
        evdict[ms*"Separate"]=measurement(0,0)
    end
80
81
82
83
84
85     for ms in popmsvec
86         ms=="NucPop" ? (pr=np_prior; bx=np_box) : (pr=prior; bx=box)
87
88
89         c_ens=gmcdir*/" * morph * "_c_*ms
90         c_obs=vcat(md[ms],ctl_measure_dict[ms])
91         if isfile(c_ens*/ens")
92             e=deserialize(c_ens*/ens")
93         else
94             e=LogNormal_Ensemble(c_ens,n_models, c_obs, pr, bx, gmc ... )
95         end
96
97
98         ← evdict[ms*"Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_di
99         ← lower_displays=lds, disp_rot_its=10000,
100         ← converge_criterion="compression", converge_factor=1e-6)
101
102
103
104

```

```

105             e=LogNormal_Ensemble(ens,n_models, obs, pr, bx, gmc ... )
106         end
107
108     end
109 end
110
111 for ms in normmsvec
112     c_ens=gmcdir*"/" * morph * "_c_"*ms
113     c_obs=vcat(md[ms],ctl_measure_dict[ms])
114     if isfile(c_ens*"/ens")
115         e=deserialize(c_ens*"/ens")
116     else
117         if ms=="Sphericity"
118             e=Normal_Ensemble(c_ens,n_models, c_obs, s_prior, s_box,
119                                 → gmc ... )
120         else
121             e=Normal_Ensemble(c_ens,n_models, c_obs, prior, box,
122                                 → gmc ... )
123         end
124     end
125
126     evdict[ms*"_Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_di
127     → lower_displays=lds, disp_rot_its=10000,
128     → converge_criterion="compression", converge_factor=1e-6)
129
130 for (msd,p) in zip([md,ctl_measure_dict],["/"*morph*"_","/ctl_"])
131     ens=gmcdir*p*ms
132     obs=msd[ms]
133     if isfile(ens*"/ens")
134         e=deserialize(ens*"/ens")
135     else
136         if ms=="Sphericity"
137             e=Normal_Ensemble(ens,n_models, obs, s_prior, s_box,
138                                 → gmc ... )
139         else
140             e=Normal_Ensemble(ens,n_models, obs, prior, box,
141                                 → gmc ... )
142         end
143     end
144 end

```

```

138
139
140     ↵ evdict[ms*"_Separate"]+=converge_ensemble!(e,backup=true,10000),upper
141     ↵ lower_displays=lds, disp_rot_its=10000,
142     ↵ converge_criterion="compression", converge_factor=1e-6)
143 end
144 end
145
146 for ms in totalmsvec
147     sev=evdict[ms*"_Separate"]
148     cev=evdict[ms*"_Combined"]
149     ratio=sev-cev
150     sig=ratio.val/ratio.err
151
152     println("$morph $ms & $(round(sev,digits=3)) &
153             & $(round(cev,digits=3)) & $(round(ratio,digits=3)) &
154             & $(round(sig,digits=1)) \\\hline")
155 end
156 end
157
158 colvec=[:darkmagenta,:darkorange,:green]
159
160 np_plt=plot_logn_MTDist([atg_measure_dict["NucPer"],spl_measure_dict["NucPer"],ctl_me
161     ↵ Mo.,"Spl Mo.,"Ctrl Mo."],"Mean Sectional Population per CMZ
162     ↵ Cell","Posterior mean lh.")
163
164 savefig(np_plt, "/bench/PhD/Thesis/images/rys/morphnuclei.png")

```

---

### 17.8.18 /rys/a38.jl

```

1 using CSV,DataFrames,Distributions,NGRefTools,StatsBase,StatsPlots
2 gr()
3 default(fontsize = 10, guidefont = (12,:bold), tickfont = 10, guide =
4     ↵ "x")
5
6 pth="/bench/PhD/datasets/A38.csv"
7
8 a38df=DataFrame(CSV.read(pth))
9 a38df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
10     ↵ eachrow(a38df)]
11 a38df.Group[1:10].="sib"

```

```

11 a38df.Group[11:22]•="rys"
12
13 sib_measure_dict=Dict{String,Vector{Float64}}(){}
14 rys_measure_dict=Dict{String,Vector{Float64}}(){}
15
16 for dict in [sib_measure_dict,rys_measure_dict]
17     dict["DCMZ"]=Vector{Float64}(){}
18     dict["VCMZ"]=Vector{Float64}(){}
19     dict["TCMZ"]=Vector{Float64}(){}
20     dict["NucPop"]=Vector{Float64}(){}
21     dict["Sphericity"]=Vector{Float64}(){}
22     dict["Volume"]=Vector{Float64}(){}
23 end
24
25 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
26 for (gdf,dict) in zip(groupby(a38df,
27     ↪ "Group"),[sib_measure_dict,rys_measure_dict])
28     for row in eachrow(gdf)
29         if row."D/V"=="D"
30             push!(dict["DCMZ"],row."PCNA +ve")
31             push!(dict["TCMZ"],row."Total PCNA")
32             push!(dict["NucPop"],row."Total nuclei")
33             push!(dict["Sphericity"],row."Sphericity")
34             push!(dict["Volume"],row."Volume")
35         else
36             push!(dict["VCMZ"],row."PCNA +ve")
37
38             ↪ dict["Sphericity"][]+=row."Sphericity";dict["Sphericity"][]/=2
39             dict["Volume"][]+=row."Volume";dict["Volume"][]/=2
40         end
41     end
42 end
43
44 #TEST IF DATA IS BETTER MODELLED NORMALLY OR LOGNORMALLY
45 for ms in ["DCMZ", "VCMZ", "NucPop", "Sphericity", "Volume"]
46     n_ms=[fit(Normal,dict[ms]) for dict in [sib_measure_dict,
47         ↪ rys_measure_dict]]
48     logn_ms=[fit(LogNormal,dict[ms]) for dict in [sib_measure_dict,
49         ↪ rys_measure_dict]]
50
51     ↪ normal_lh=sum(logpdf(n_ms[1],sib_measure_dict[ms]))+sum(logpdf(n_ms[2],rys_me

```



```

77 combined=plot(TCMZ_plt,PopRatio_plt,DCMZ_plt,VCMZ_plt,Vol_plt,Sph_plt,layout=grid(3,2
    ↵   size=(900,900))
78
79 savefig(combined,"/bench/PhD/Thesis/images/rys/nuclearstudy.png")

```

---

### 17.8.19 /rys/a38GMC\_NS.jl

```

1 using CSV,DataFrames,Distributions,GMC_NS,Measurements,Serialization
2
3 pth="/bench/PhD/datasets/A38.csv"
4
5 a38df=DataFrame(CSV.read(pth))
6 a38df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
    ↵ eachrow(a38df)]
7 a38df.Group[1:10].="sib"
8 a38df.Group[11:22].="rys"
9
10 sib_measure_dict=Dict{String,Vector{Float64}}(){}
11 rys_measure_dict=Dict{String,Vector{Float64}}(){}
12
13 for dict in [sib_measure_dict,rys_measure_dict]
14     dict["DCMZ"]=Vector{Float64}(){}
15     dict["VCMZ"]=Vector{Float64}(){}
16     dict["TCMZ"]=Vector{Float64}(){}
17     dict["NucPop"]=Vector{Float64}(){}
18     dict["Sphericity"]=Vector{Float64}(){}
19     dict["Volume"]=Vector{Float64}(){}
20 end
21
22 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
23 for (gdf,dict) in zip(groupby(a38df,
    ↵ "Group"),[sib_measure_dict,rys_measure_dict])
24     for row in eachrow(gdf)
25         if row."D/V"=="D"
26             push!(dict["DCMZ"],row."PCNA +ve")
27             push!(dict["TCMZ"],row."Total PCNA")
28             push!(dict["NucPop"],row."Total nuclei")
29             push!(dict["Sphericity"],row."Sphericity")
30             push!(dict["Volume"],row."Volume")
31         else
32             push!(dict["VCMZ"],row."PCNA +ve")

```

```

33         → dict["Sphericity"][[end]]+=row."Sphericity";dict["Sphericity"][[end]]/=2
34         dict["Volume"][[end]]+=row."Volume";dict["Volume"][[end]]/=2
35     end
36 end
37 end
38
39 for md in [sib_measure_dict,rys_measure_dict]
40     md["NucPer"]=md["NucPop"]./md["TCMZ"]
41 end
42
43
44 gmc=GMC_DEFAULTS
45 gmc[1]=15
46 gmc[2]=1.e-15
47 gmcdir="/bench/PhD/NGS_binaries/GMC_NS/A38"
48
49
50 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]])
51 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
52
53 n_models=150
54
55 evdict=Dict{String,Measurement}()
56
57 totalmsvec=[ "TCMZ", "NucPer", "DCMZ", "VCMZ", "Volume", "Sphericity" ]
58 popmsvec=[ "TCMZ", "NucPer", "DCMZ", "VCMZ" ]
59 normmsvec=[ "Volume", "Sphericity" ]
60
61 for ms in totalmsvec
62     evdict[ms*"_Combined"] = measurement(0,0)
63     evdict[ms*"_Separate"] = measurement(0,0)
64 end
65
66 max_mu=300.;min_mu=0.
67 max_lambda=.3;min_lambda=0.
68 prior=[Uniform(min_mu,max_mu),Uniform(min_lambda,max_lambda)]
69 box=[min_mu max_mu;min_lambda max_lambda]
70
71 s_max_mu=1.;s_min_mu=0.
72 s_max_lambda=5e5;s_min_lambda=0.
73 s_prior=[Uniform(s_min_mu,s_max_mu),Uniform(s_min_lambda,s_max_lambda)]
74 s_box=[s_min_mu s_max_mu;s_min_lambda s_max_lambda]

```

```

75
76 for ms in popmsvec
77     c_ens=gmmdir*"/c_}*ms
78     c_obs=vcat(sib_measure_dict[ms],rys_measure_dict[ms])
79     if isfile(c_ens*"/ens")
80         e=deserialize(c_ens*"/ens")
81     else
82         e=LogNormal_Ensemble(c_ens,n_models, c_obs, prior, box, gmc ... )
83     end
84
85     → evdict[ms*"_Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_displa
     → lower_displays=lds, disp_rot_its=10000, converge_factor=1e-6)
86
87     for (msd,p) in zip([sib_measure_dict,rys_measure_dict],["/s_","/r_"])
88         ens=gmmdir*p*ms
89         obs=msd[ms]
90         if isfile(ens*"/ens")
91             e=deserialize(ens*"/ens")
92         else
93             e=LogNormal_Ensemble(ens,n_models, obs, prior, box, gmc ... )
94         end
95
96         → evdict[ms*"_Separate"]+=converge_ensemble!(e,backup=(true,10000),upper_di
         → lower_displays=lds, disp_rot_its=10000, converge_factor=1e-6)
97     end
98
99     for ms in normmsvec
100        c_ens=gmmdir*"/c_}*ms
101        c_obs=vcat(sib_measure_dict[ms],rys_measure_dict[ms])
102        if isfile(c_ens*"/ens")
103            e=deserialize(c_ens*"/ens")
104        else
105            if ms=="Sphericity"
106                e=Normal_Ensemble(c_ens,n_models, c_obs, s_prior, s_box,
107                                → gmc ... )
108            else
109                e=Normal_Ensemble(c_ens,n_models, c_obs, prior, box, gmc ... )
110            end
111        end

```

```

112      ↳ evdict[ms*"_Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_displa
113      ↳ lower_displays=lds, disp_rot_its=10000, converge_factor=1e-6)
114
115      for (msd,p) in zip([sib_measure_dict,rys_measure_dict],["/s_","/r_"])
116          ens=gmcdir*p*ms
117          obs=msd[ms]
118          if isfile(ens*"/ens")
119              e=deserialize(ens*"/ens")
120          else
121              if ms=="Sphericity"
122                  e=Normal_Ensemble(ens,n_models,obs,s_prior,s_box,
123                      ↳ gmc ... )
124              else
125                  e=Normal_Ensemble(ens,n_models,obs,prior,box,gmc ... )
126              end
127          end
128      end
129  end
130
131  for ms in totalmsvec
132      sev=evdict[ms*"_Separate"]
133      cev=evdict[ms*"_Combined"]
134      ratio=sev-cev
135      sig=ratio.val/ratio.err
136
137      println("$ms & $(round(sev,digits=3)) & $(round(cev,digits=3)) &
138          ↳ $(round(ratio,digits=3)) & $(round(sig,digits=1)) \\\\" \\\\hline")
139  end

```

---

### 17.8.20 /rys/caspase.jl

---

```

1 using CSV, DataFrames, Plots, Distributions, Measures
2 import Plots:Plot
3
4 gr()
5
6 caspase_pth="/bench/PhD/Thesis/datasets/rys_caspase.csv"

```

```

7 caspasedf=DataFrame(CSV.read(caspase_pth))
8
9 sib_ms_dict=Dict{String,Vector{Vector{Float64}}}()
10 rys_ms_dict=Dict{String,Vector{Vector{Float64}}}()
11
12 X=[4.,6.]
13
14 for md in [sib_ms_dict,rys_ms_dict]
15     md["CMZ"]=[zeros(0) for i in 1:2]
16     md["Border"]=[zeros(0) for i in 1:2]
17     md["Central"]=[zeros(0) for i in 1:2]
18     md["Total"]=[zeros(0) for i in 1:2]
19 end
20
21 for row in eachrow(caspasedf)
22     row.dpf==4 ? (xidx=1) : (xidx=2)
23     row.genotype=="sib" ? (md=sib_ms_dict) : (md=rys_ms_dict)
24     push!(md["CMZ"][xidx],row.CMZ)
25     push!(md["Border"][xidx],row."CMZ/central border region")
26     push!(md["Central"][xidx],row."central retina")
27     push!(md["Total"][xidx],row.CMZ + row."CMZ/central border region" +
28         ↪ row."central retina")
29 end
30 plot_halves=Vector{Plot}()
31 for (n, age) in enumerate(X)
32     subplots=Vector{Plot}()
33     for (m,ms) in enumerate(["CMZ","Border","Central","Total"])
34         s_data=sib_ms_dict[ms][n]
35         sl=length(s_data)
36         r_data=rys_ms_dict[ms][n]
37         rl=length(r_data)
38         if m == 1
39             ↪ subplot=bar(1:1:sl,s_data,barwidths=1,color=:green,xticks=(1:1:sl+rl,
40             ↪ for i in 1:sl],[ "R" for i in 1:rl])),legend=:none,
41             ↪ xlabel=ms, margin=0mm,ylabel="Caspase-3+ cells")
42         else
43             ↪ subplot=bar(1:1:sl,s_data,barwidths=1,color=:green,xticks=(1:1:sl+rl,
44             ↪ for i in 1:sl],[ "R" for i in 1:rl])),legend=:none,
45             ↪ xlabel=ms, leftmargin=-7.5mm, yformatter=_→"")
46         end

```

```

43         bar!(sl+1:1:sl+rl,r_data,color=:darkmagenta, barwidths=1.)
44         push!(subplots,subplt)
45     end
46     title = plot(title = "$(Int64(age)) dpf", grid = false, showaxis =
47       ↪ nothing, bottom_margin = -35Plots.px)
47     l=@layout [t{.02h};[a{.25w} b{.25w} c{.25w} d{.25w}]]
48     half=plot(title, subplots..., layout=l, link=:y)
49     push!(plot_halves,half)
50 end
51
52 combined=plot(plot_halves..., layout=grid(1,2), size=(900,450))
53
54 savefig(combined, "/bench/PhD/Thesis/images/rys/caspase.png")
55
56 println("Rys CMZ caspase 4dpf: $(mean(rys_ms_dict["CMZ"][[1]])) ±
57   ↪ $(std(rys_ms_dict["CMZ"][[1]]))")
57 println("Rys CMZ caspase 6dpf: $(mean(rys_ms_dict["CMZ"][[2]])) ±
58   ↪ $(std(rys_ms_dict["CMZ"][[2]]))")
58
59 println("Rys central caspase 4dpf: $(mean(rys_ms_dict["Central"][[1]])) ±
60   ↪ $(std(rys_ms_dict["Central"][[1]]))")
60 println("Rys central caspase 6dpf: $(mean(rys_ms_dict["Central"][[2]])) ±
61   ↪ $(std(rys_ms_dict["Central"][[2]]))")
61 println("Sib central caspase 4dpf: $(mean(sib_ms_dict["Central"][[1]])) ±
62   ↪ $(std(sib_ms_dict["Central"][[1]]))")
62 println("Sib central caspase 6dpf: $(mean(sib_ms_dict["Central"][[2]])) ±
63   ↪ $(std(sib_ms_dict["Central"][[2]]))")

```

---

### 17.8.21 /rys/em.jl

---

```

1 using Plots, Images, FileIO, Plots.PlotMeasures
2
3 sib36=load("/bench/PhD/Thesis/images/rys/C2 cell 1/Rys sib C2 36k cell
4   ↪ 1.jpg")
4 sib36=sib36[1:2465,:]
5 sib110=load("/bench/PhD/Thesis/images/rys/C2 cell 1/Rys sib C2 110k cell
6   ↪ 1.jpg")
6 sib110=sib110[1:2465,:]
7 sib210=load("/bench/PhD/Thesis/images/rys/C2 cell 1/Rys sib C2 210k cell
8   ↪ 1.jpg")
8 sib210=sib210[1:2465,:]
9

```

```

10 rys36=load("/bench/PhD/Thesis/images/rys/D1 cell 1/Rys mut D1 CMZ
   ↳ 36k.jpg")
11 rys36=rys36[1:2465,:]
12 rys110=load("/bench/PhD/Thesis/images/rys/D1 cell 1/Rys mut D1 CMZ
   ↳ 110k.jpg")
13 rys110=rys110[1:2465,:]
14 rys210=load("/bench/PhD/Thesis/images/rys/D1 cell 1/Rys mut D1 CMZ
   ↳ 210k.jpg")
15 rys210=rys210[1:2465,:]

16
17
18 s36p=plot(sib36, axis=nothing, border=:none, margin=0mm)
19 annotate!([(200,105,Plots.text("SIB",25,:white,:bold))])
20 s110p=plot(sib110, axis=nothing, border=:none, margin=0mm)
21 annotate!([(200,105,Plots.text("SIB",25,:white,:bold))])
22 s210p=plot(sib210, axis=nothing, border=:none, margin=0mm)
23 annotate!([(200,105,Plots.text("SIB",25,:white,:bold))])
24 r36p=plot(rys36, axis=nothing, border=:none, margin=0mm)
25 annotate!([(200,105,Plots.text("RYS",25,:white,:bold))])
26 r110p=plot(rys110, axis=nothing, border=:none, margin=0mm)
27 annotate!([(200,105,Plots.text("RYS",25,:white,:bold))])
28 r210p=plot(rys210, axis=nothing, border=:none, margin=0mm)
29 annotate!([(200,105,Plots.text("RYS",25,:white,:bold))])

30
31 combined=plot(s36p, r36p, s110p, r110p, s210p, r210p,
   ↳ layout=grid(3,2),size=(1200,2400))
32
33 savefig(combined, "/bench/PhD/Thesis/images/emtogimp.png")

```

---

### 17.8.22 /rys/occupancy.py

```

1 import os
2 import numpy as np
3 import pandas as pd
4 from Bio import SeqIO
5 import matplotlib.pyplot as plt
6 from matplotlib.gridspec import GridSpec
7 from matplotlib import cm
8 from matplotlib import colors
9
10 from PIL import Image
11 from io import BytesIO

```

```
12
13 def get_chr_props(genome):
14     genome_dict = SeqIO.index(genome, "fasta")
15     length_dict = {}
16     prop_dict = {}
17
18     total_length = 0
19     nonchr_length = 0
20     for entry in genome_dict:
21         length=len(genome_dict[entry].seq)
22         if entry.isdigit():
23             length_dict[entry] = length
24             total_length += length
25         else:
26             nonchr_length += length
27             total_length += length
28     length_dict['nonchr']=nonchr_length
29
30     for entry in length_dict:
31         prop_dict[entry] = length_dict[entry]/total_length
32
33     return prop_dict
34
35 def build_position_dict(df):
36     observed_dict={}
37     observed_nonchr_positions = 0
38
39     for chr, chrgp_data in df.groupby('chr'):
40         length=len(chrgp_data)
41         if chr.isdigit():
42             observed_dict[chr]=length
43         else:
44             observed_nonchr_positions+=length
45     observed_dict['nonchr']=observed_nonchr_positions
46
47     return observed_dict
48
49 def build_signal_dict(df):
50     observed_dict={}
51     observed_nonchr_signal = 0
52
53     for chr, chrgp_data in df.groupby('chr'):
54         signal=chrgp_data['smt_value'].sum()
```

```

55     if chr.isdigit():
56         observed_dict[chr]=signal
57     else:
58         observed_nonchr_signal+=signal
59     observed_dict['nonchr']=observed_nonchr_signal
60
61     return observed_dict
62
63 #relevant file locations
64 reference_genome = flow_variables['reference_genome_input']
65 sib_positions =
66     ↪ os.path.join(input_table_1['PathToDanposResults'][0], 'pooled', input_table_2['Danp
66 rys_positions =
67     ↪ os.path.join(input_table_1['PathToDanposResults'][0], 'pooled', input_table_2['Danp
68
68 #read in dataframes
69 sib_pos_df = pd.read_table(sib_positions, dtype={'chr':'category',
70     ↪ 'start':'uint32', 'end':'uint32', 'smt_pos':'uint32',
71     ↪ 'smt_value':'float32', 'fuzziness_score':'float32'})
70 rys_pos_df = pd.read_table(rys_positions, dtype={'chr':'category',
71     ↪ 'start':'uint32', 'end':'uint32', 'smt_pos':'uint32',
72     ↪ 'smt_value':'float32', 'fuzziness_score':'float32'})
72
72 #get proportional lengths of extrachromosomal material from reference
73     ↪ genome
73 prop_dict=get_chr_props(reference_genome)
74
75 #get total positions and signals from sib and rys dfs
76 sib_total_positions=len(sib_pos_df)
77 sib_total_signal=sib_pos_df['smt_value'].sum()
78 rys_total_positions=len(rys_pos_df)
79 rys_total_signal=rys_pos_df['smt_value'].sum()
80
81 #build dict of expected position numbers and signal sums for sibs based
82     ↪ on percentage of genome comprised by relevant chr
82 sib_expected_pos_d={}
83 sib_expected_sig_d={}
84 for entry in prop_dict:
85     sib_expected_pos_d[entry]=sib_total_positions * prop_dict[entry]
86     sib_expected_sig_d[entry]=sib_total_signal * prop_dict[entry]
87
88 #build dicts of observed position numbers and signal sumit sums by
89     ↪ chromosome

```

```
89 sib_posn_d=build_position_dict(sib_pos_df)
90 sib_sig_d=build_signal_dict(sib_pos_df)
91 rys_posn_d=build_position_dict(rys_pos_df)
92 rys_sig_d=build_signal_dict(rys_pos_df)
93
94 #build arrays of observed positions, signals, and proportional
95 #    ↳ differences
95 sib_observed_positions=[]
96 sib_observed_signal=[]
97 rys_observed_positions=[]
98 rys_observed_signal=[]
99
100 rys_expected_posn_from_sib=[]
101 rys_expected_sig_from_sib=[]
102
103 sib_posn_dif_from_expected=[]
104 sib_sig_dif_from_expected=[]
105 rys_posn_dif_from_sib=[]
106 rys_sig_dif_from_sib=[]
107
108 #for chr1-25
109 for i in range (1,26):
110     #build observed position & signal arrays
111     sib_observed_positions.append(sib_posn_d[str(i)])
112     sib_observed_signal.append(sib_sig_d[str(i)])
113     rys_observed_positions.append(rys_posn_d[str(i)])
114     rys_observed_signal.append(rys_sig_d[str(i)])
115
116     #build rys expected arrays
117
118     ↳ rys_expected_posn_from_sib.append(rys_total_positions*(sib_posn_d[str(i)]/sib_
119
120     ↳ rys_expected_sig_from_sib.append(rys_total_signal*(sib_sig_d[str(i)]/sib_total_
121
122     #build proportional difference arrays
123
124     ↳ sib_posn_dif_from_expected.append((sib_posn_d[str(i)]-sib_expected_posn_d[str(i)]/sib_
125
126     ↳ sib_sig_dif_from_expected.append((sib_sig_d[str(i)]-sib_expected_sig_d[str(i)]/sib_
127
128     ↳ rys_posn_dif_from_sib.append((rys_posn_d[str(i)]-rys_expected_posn_from_sib[i-1]/rys_
129
130     ↳ rys_sig_dif_from_sib.append((rys_sig_d[str(i)]-rys_expected_sig_from_sib[i-1]/rys_
```

```

125
126 #lastly append nonchromosomal material to all arrays
127 sib_observed_positions.append(sib_posn_d['nonchr'])
128 sib_observed_signal.append(sib_sig_d['nonchr'])
129 rys_observed_positions.append(rys_posn_d['nonchr'])
130 rys_observed_signal.append(rys_sig_d['nonchr'])

131
132 rys_expected_posn_from_sib.append(rys_total_positions*(sib_posn_d['nonchr']/sib_total))
133 rys_expected_sig_from_sib.append(rys_total_signal*(sib_sig_d['nonchr']/sib_total_signal))

134
135 sib_posn_dif_from_expected.append((sib_posn_d['nonchr']-sib_expected_posn_d['nonchr']))
136 sib_sig_dif_from_expected.append((sib_sig_d['nonchr']-sib_expected_sig_d['nonchr'])/sib_total)
137 rys_posn_dif_from_sib.append((rys_posn_d['nonchr']-rys_expected_posn_from_sib[25])/rys_total)
138 rys_sig_dif_from_sib.append((rys_sig_d['nonchr']-rys_expected_sig_from_sib[25])/rys_total)

139
140 #!!!sib nonchromosomal signal is blown out relative to other scaffolds.
    ↳ store value and annotate, color separately
141 NC_sig_dif = sib_sig_dif_from_expected[25]
142 sib_sig_dif_from_expected[25] = 0

143
144 #build proportional difference color arrays
145 sib_norm =
    ↳ colors.Normalize(vmin=-abs(max(sib_posn_dif_from_expected+sib_sig_dif_from_expected,
    ↳ key=abs)),
    ↳ vmax=abs(max(sib_posn_dif_from_expected+sib_sig_dif_from_expected,
    ↳ key=abs)))
146 rys_norm =
    ↳ colors.Normalize(vmin=-abs(max(rys_posn_dif_from_sib+rys_sig_dif_from_sib,
    ↳ key=abs)), vmax=abs(max(rys_posn_dif_from_sib+rys_sig_dif_from_sib,
    ↳ key=abs)))

147
148 sib_pos_colors = cm.PiYG(sib_norm(sib_posn_dif_from_expected))
149 sib_sig_colors = cm.PiYG(sib_norm(sib_sig_dif_from_expected))
150 sib_sig_colors[25] = [0, .1, 1, 1]
151 rys_pos_colors = cm.PR Gn(rys_norm(rys_posn_dif_from_sib))
152 rys_sig_colors = cm.PR Gn(rys_norm(rys_sig_dif_from_sib))

153
154 #build figure
155 fig = plt.figure(figsize=(6,6.5))

156
157 gs1 = GridSpec(2,2, wspace=.1, hspace=.05, left=.05, right=.95, top=.95,
    ↳ bottom=.15)
158 pos_sib = plt.subplot(gs1[0, 0])

```

```

159 pos_rys = plt.subplot(gs1[0, 1])
160 sig_sib = plt.subplot(gs1[1, 0])
161 sig_rys = plt.subplot(gs1[1, 1])
162
163 pos_sib.set_ylabel('Positions', fontsize=12, weight='bold')
164 sig_sib.set_ylabel('Occupancy', fontsize=12, weight='bold')
165
166 pos_sib.text(.01,.95, 'A', transform=pos_sib.transAxes,
    ↪ fontweight='bold', va='top')
167 pos_rys.text(.01,.95, 'B', transform=pos_rys.transAxes,
    ↪ fontweight='bold', va='top')
168 sig_sib.text(.01,.95, 'C', transform=sig_sib.transAxes,
    ↪ fontweight='bold', va='top')
169 sig_rys.text(.01,.95, 'D', transform=sig_rys.transAxes,
    ↪ fontweight='bold', va='top')
170
171 gs2 = GridSpec(4, 2, wspace=.1, hspace=.05, left=.05, right=.95, top=.15,
    ↪ bottom=.05)
172 sib_cbax = plt.subplot(gs2[3, 0])
173 rys_cbax = plt.subplot(gs2[3, 1])
174
175 #colorbar stuff
176 ssm = cm.ScalarMappable(cmap=cm.PiYG, norm=sib_norm)
177 ssm._A =[]
178 sib_cb=plt.colorbar(ssm, cax=sib_cbax, orientation='horizontal',
    ↪ extend='max',
    ↪ ticks=[-abs(max(sib_posn_dif_from_expected+sib_sig_dif_from_expected,
    ↪ key=abs)), -.3, -.2, -.1, 0, .1, .2, .3])
179 sib_cb.ax.set_xticklabels(['{\0:.2f}'.format(-abs(max(sib_posn_dif_from_expected+sib_s
    ↪ key=abs))), '-0.3', '-0.2', '-0.1', '0.0', '0.1', '0.2', '0.3'])
180 sib_cbax.set_xlabel('Fold difference from per-kb uniformity')
181 sib_cbax.xaxis.set_label_position('top')
182
183 rsm = cm.ScalarMappable(cmap=cm.PRGn, norm=rys_norm)
184 rsm._A =[]
185 rsm_cb=plt.colorbar(rsm, cax=rys_cbax, orientation='horizontal',
    ↪ ticks=[-abs(max(rys_posn_dif_from_sib+rys_sig_dif_from_sib,
    ↪ key=abs)), -.05, -.025, 0, .025, .05,
    ↪ abs(max(rys_posn_dif_from_sib+rys_sig_dif_from_sib, key=abs))])
186 rsm_cb.ax.set_xticklabels(['{\0:.2f}'.format(-abs(max(rys_posn_dif_from_sib+rys_sig_di
    ↪ key=abs))), '-0.05', '-0.025', '0.0', '0.025', '0.05', '{\0:.2f}'.format(abs(max(rys_pos
    ↪ key=abs))))]
187 rys_cbax.set_xlabel('Fold difference from sib proportions')

```

```
188 rys_cbax.xaxis.set_label_position('top')
189
190 #labels for chr wedges
191 labels=list(range(1,26))
192 labels.append('NC')
193
194 #make the donut plots
195 pos_sib.pie(sib_observed_positions, colors=sib_pos_colors, radius=1,
   ↵ startangle=90, counterclock=False,
   ↵ wedgeprops=dict(width=.7,edgecolor='w'), labels=labels,
   ↵ textprops=dict(fontsize=6))
196 pos_sib.text(0.5, 0.5, 'sib', horizontalalignment='center',
   ↵ verticalalignment='center', transform=pos_sib.transAxes,
   ↵ weight='bold', fontsize='16')
197 pos_rys.pie(rys_observed_positions, colors=rys_pos_colors, radius=1,
   ↵ startangle=90, counterclock=False,
   ↵ wedgeprops=dict(width=.7,edgecolor='w'), labels=labels,
   ↵ textprops=dict(fontsize=6))
198 pos_rys.text(0.5, 0.5, 'rys', horizontalalignment='center',
   ↵ verticalalignment='center', transform=pos_rys.transAxes,
   ↵ weight='bold', fontsize='16')
199
200 wedges, texts = sig_sib.pie(sib_observed_signal, colors=sib_sig_colors,
   ↵ radius=1, startangle=90, counterclock=False,
   ↵ wedgeprops=dict(width=.7,edgecolor='w'), labels=labels,
   ↵ textprops=dict(fontsize=6))
201 sig_sib.text(0.5, 0.5, 'sib', horizontalalignment='center',
   ↵ verticalalignment='center', transform=sig_sib.transAxes,
   ↵ weight='bold', fontsize='16')
202 sig_rys.pie(rys_observed_signal, colors=rys_sig_colors, radius=1,
   ↵ startangle=90, counterclock=False,
   ↵ wedgeprops=dict(width=.7,edgecolor='w'), labels=labels,
   ↵ textprops=dict(fontsize=6))
203 sig_rys.text(0.5, 0.5, 'rys', horizontalalignment='center',
   ↵ verticalalignment='center', transform=sig_rys.transAxes,
   ↵ weight='bold', fontsize='16')
204
205 #annotate NC wedge
206 wedges[25].set_hatch('//')
207 ang = (wedges[25].theta2 - wedges[25].theta1)/2. + wedges[25].theta1
208 y = np.sin(np.deg2rad(ang))
209 x = np.cos(np.deg2rad(ang))
```

```

210 horizontalalignment = {-1: "right", 1:
211     "left"}[int(np.sign(x))]
212 connectionstyle = "angle,angleA=0,angleB={}{}".format(ang)
213 kw = dict(xycoords='data', textcoords='data',
214     arrowprops=dict(arrowstyle="-"),
215     zorder=0, va="center", fontsize=9)
216 kw["arrowprops"].update({"connectionstyle": connectionstyle})
217 sig_sib.annotate('{0:.2f}'.format(NC_sig_dif) + '-fold', xy=(x, y),
218     xytext=(.8 * np.sign(x), 1.30 * y),
219     horizontalalignment=horizontalalignment, **kw)
220
221 fig.savefig(os.path.join(input_table_1['PathToDanposResults'][0], 'proportional_chromo'
222 png_memory = BytesIO()
223 fig.savefig(png_memory, format='png', dpi=600)
224 PILpng = Image.open(png_memory)
225 PILpng.save(os.path.join(input_table_1['PathToDanposResults'][0], 'proportional_chromo'
226 png_memory.close())
227
228 # Copy input to output
229 output_table_1 = input_table_1.copy()
230 output_table_2 = input_table_2.copy()

```

---

### 17.8.23 /rys/qPCR.jl

---

```

1 using CSV, DataFrames, Measurements, Plots, Distributions
2 import Plots:Plot
3
4 rand6pth="/bench/PhD/Thesis/datasets/6dpfRys-Histones random primers
4     qPCR.csv"
5 rand8pth="/bench/PhD/Thesis/datasets/8dpfRys-Histones random primers
5     qPCR.csv"
6 dt6pth="/bench/PhD/Thesis/datasets/6dpfRys-Histones oligo dT qPCR.csv"
7 dt8pth="/bench/PhD/Thesis/datasets/8dpfRys-Histones oligo dT qPCR.csv"
8
9 rand6=DataFrame(CSV.read(rand6pth))
10 rand8=DataFrame(CSV.read(rand8pth))
11 dt6=DataFrame(CSV.read(dt6pth))
12 dt8=DataFrame(CSV.read(dt8pth))
13
14 randdfs=[rand6,rand8]
15 dtdfs=[dt6,dt8]

```

```

16 samples=Dict{Tuple{String,Int64},Vector{String}}}()
17
18 measure_dict=Dict{Tuple{String,Int64,Int64,String,String},Vector{Float64}}(){}
19
20 for (df,d) in zip(randdfs,[6,8])
21     samples[("rand",d)]=unique(df.Sample)
22     for row in eachrow(df)
23         str=("rand",d,row.Plate,row.Sample,row.Detector)
24         if !ismissing(row.Ct)
25             str in keys(measure_dict) ? push!(measure_dict[str],row.Ct) :
26                 measure_dict[str]=[row.Ct]
27         end
28     end
29
30 for (df,d) in zip(dtdfs,[6,8])
31     samples[("dt",d)]=unique(df."Sample Name")
32     for row in eachrow(df)
33         str=("dt",d,row.Plate,row."Sample Name",row."Target Name")
34         if !ismissing(row.CT)
35             str in keys(measure_dict) ? push!(measure_dict[str],row.CT) :
36                 measure_dict[str]=[row.CT]
37         end
38     end
39
40 X=[6.,8.]
41 rand_sib=Dict{String,Vector{Vector{Measurement}}}(){}
42 rand_rys=Dict{String,Vector{Vector{Measurement}}}(){}
43 dt_sib=Dict{String,Vector{Vector{Measurement}}}(){}
44 dt_rys=Dict{String,Vector{Vector{Measurement}}}(){}
45
46
47 for ms in ["H2A","H2B","H3","H4"]
48     for d in [6,8]
49         xidx=findfirst(isequal(d),X)
50         for plate in [1,2,3]
51             wt_hk=measure_dict[("rand",d,plate,"WT cDNA 10x","ActinB")]
52             length(wt_hk)>1 ? wt_hk_m=measurement(mean(wt_hk),std(wt_hk)) :
53                 : wt_hk_m=measurement(mean(wt_hk),0.)
54             wt_trg=measure_dict[("rand",d,plate,"WT cDNA 10x",ms)]

```

```

54         length(wt_hk)>1 ?
55             ↳ wt_trg_m=measurement(mean(wt_trg),std(wt_trg)) :
56             ↳ wt_trg_m=measurement(mean(wt_trg),0.)
57             wt_DeltaCt=wt_trg_m-wt_hk_m
58
59         for (smpl, msdict) in zip(["Sib cDNA 10x","Rys cDNA
60             ↳ 10x"],[rand_sib,rand_rys])
61             smpl_hk=measure_dict[("rand",d,plate,smpl,"ActinB")]
62             length(smpl_hk)>1 ?
63                 ↳ smpl_hk_m=measurement(mean(smpl_hk),std(smpl_hk)) :
64                 ↳ smpl_hk_m=measurement(mean(smpl_hk),0.)
65             smpl_trg=measure_dict[("rand",d,plate,smpl,ms)]
66             length(wt_hk)>1 ?
67                 ↳ smpl_trg_m=measurement(mean(smpl_trg),std(smpl_trg))
68                 ↳ : smpl_trg_m=measurement(mean(smpl_trg),0.)
69             smpl_DeltaCt=smpl_trg_m-smpl_hk_m
70             DeltaDeltaCt=smpl_DeltaCt-wt_DeltaCt
71             fold_change=2^(-DeltaDeltaCt)
72             ms in keys(msdict) ? push!(msdict[ms][xidx],fold_change)
73             ↳ : msdict[ms]=[[fold_change],Vector{Float64}()]
74         end
75     end
76   end
77 end
78
79 msvec=[ "H2A", "H2B", "H3" , "H4" ]
80
81 for ms in msvec
82     for d in [6,8]
83         xidx=findfirst(isequal(d),X)
84         for plate in [1,2,3]
85             wt_hk=measure_dict[("dt",d,plate,"WT cDNA","ActinB")]
86             length(wt_hk)>1 ? wt_hk_m=measurement(mean(wt_hk),std(wt_hk))
87             ↳ : wt_hk_m=measurement(mean(wt_hk),0.)
88             wt_trg=measure_dict[("dt",d,plate,"WT cDNA",ms)]
89             length(wt_trg)>1 ?
90                 ↳ wt_trg_m=measurement(mean(wt_trg),std(wt_trg)) :
91                 ↳ wt_trg_m=measurement(mean(wt_trg),0.)
92             wt_DeltaCt=wt_trg_m-wt_hk_m
93
94             for (smpl, msdict) in zip(["Sib cDNA","Rys
95                 ↳ cDNA"],[dt_sib,dt_rys])
96                 smpl_hk=measure_dict[("dt",d,plate,smpl,"ActinB")]

```

```

85         length(smpl_hk)>1 ?
86             ↳ smpl_hk_m=measurement(mean(smpl_hk),std(smpl_hk)) :
87             ↳ smpl_hk_m=measurement(mean(smpl_hk),0.)
88         smpl_trg=measure_dict[("dt",d,plate,smpl,ms)]
89         length(smpl_trg)>1 ?
90             ↳ smpl_trg_m=measurement(mean(smpl_trg),std(smpl_trg))
91             ↳ : smpl_trg_m=measurement(mean(smpl_trg),0.)
92         smpl_Dt=smpl_trg_m-smpl_hk_m
93         ΔDt=smpl_Dt-wt_Dt
94         fold_change=2^ΔDt
95         ms in keys(msdict) ? push!(msdict[ms][xidx],fold_change)
96             ↳ : msdict[ms]=[[fold_change],Vector{Float64}()]
97     end
98   end
99 end
100
101 plot_halves=Vector{Plot}()
102 for ((sib_dict, rys_dict),assay_name) in
103   zip([(rand_sib,rand_rys),(dt_sib,dt_rys)],["Total transcript",
104         "Polyadenylated Transcript"])
105   subplots=Vector{Plot}()
106   for (n,ms) in enumerate(msvec)
107     plt=0.
108     mx=0.
109     mnn=Inf
110     for (dict,color,symbol,leg) in zip([sib_dict,
111       ↳ rys_dict],[:green,:darkmagenta,:circle,:diamond],["Sib","rys"])
112       means=[mean(valvec) for valvec in dict[ms]]
113       stdevs=[std(valvec) for valvec in dict[ms]]
114       upper=Vector{Float64}()
115       mns=[mn.val for mn in means]
116       stds=[sd.val for sd in stdevs]
117       lower=Vector{Float64}()
118
119       for (mn,sd) in zip(mns,stds)
120         dist=Normal(mn, sd)
121         push!(upper,quantile(dist,.975)-mn)
122         push!(lower,mn-quantile(dist,.025))
123     end
124
125     xs=vcat([[X[n] for i in 1:length(dict[ms][n])] for n in
126           ↳ 1:length(X)] ... )

```

```

119         ys=vcat([[m.val for m in dict[ms][n]] for n in
120             ↪ 1:length(X)] ... )
121
122         mx=max(mx,maximum(vcat(upper,ys)))
123         mnn=min(mnn,minimum(vcat(lower,ys)))
124
125         if dict==sib_dict
126             if n!=4 && n!=1
127                 plt=scatter(xs,ys, marker=symbol, color=color,
128                     ↪ markerstrokecolor=color, markersize=3,
129                     ↪ label=leg*" data", showaxis=:y, xticks=X,
130                     ↪ xformatter=_→" ", ylabel="Fold
131                     ↪ change", xlims=[5.5,8.5],legend=:none)
132             elseif n==1
133                 plt=scatter(xs,ys, marker=symbol, color=color,
134                     ↪ markerstrokecolor=color, markersize=3,
135                     ↪ label=leg*" data", showaxis=:y, xticks=X,
136                     ↪ xformatter=_→" ", ylabel="Fold
137                     ↪ change", xlims=[5.5,8.5],legend=:topleft,
138                     ↪ title=assay_name)
139             else
140                 plt=scatter(xs,ys, marker=symbol, color=color,
141                     ↪ markerstrokecolor=color, markersize=3,
142                     ↪ label=leg*" data", xticks=X, ylabel="Fold
143                     ↪ change", xlabel="Age (dpf)",
144                     ↪ xlims=[5.5,8.5],legend=:none)
145             end
146         else
147             scatter!(plt, xs,ys, marker=symbol, color=color,
148                 ↪ markerstrokecolor=color, markersize=3, label=leg)
149         end
150
151         plot!(plt, X, mns, ribbon=(lower,upper), color=color,
152             ↪ label=leg*" mean")
153     end
154     plot!(X, [1. for i in 1:length(X)], linewidth=4., style=:dot,
155         ↪ color=:black, label="WT std.")
156     annotate!([(5.75,mnn+.25*(mx-mnn),Plots.text(ms,16))])
157     println(ms)
158     println("max: $mx min: $mnn")
159     println(mnn+.25*(mx-mnn))
160     push!(subplots,plt)
161
162 end

```

```

145
146     assay_combined=plot(subplots ... ,layout=grid(4,1),link=:x)
147     push!(plot_halves,assay_combined)
148 end
149
150 combined=plot(plot_halves ... ,layout=grid(1,2),size=(900,900))
151 savefig(combined,"/bench/PhD/Thesis/images/rys/qPCR.png")
152
153 stdmat=zeros(4,2,2,2)
154
155 for ((sib_dict, rys_dict),assay_idx) in
156     zip([(rand_sib,rand_rys),(dt_sib,dt_rys)],[1,2])
157     for (n,ms) in enumerate(msvec)
158         sib_means=0.
159         for dict in [sib_dict, rys_dict]
160             means=[measurement(mean(valvec).val,std(valvec).val) for
161                     valvec in dict[ms]]
162             if dict==sib_dict
163                 sib_means=means
164             else
165                 sdifs=[measurement(mean(valvec).val,std(valvec).val) for
166                         valvec in dict[ms]]-sib_means
167                 stds=[v.val/v.err for v in sdifs]
168                 stdmat[n,assay_idx,1,:]=stds
169
170                 wtdifs=[measurement(mean(valvec).val,std(valvec).val) for
171                         valvec in dict[ms]]-1.
172                 stds=[v.val/v.err for v in wtdifs]
173                 stdmat[n,assay_idx,2,:]=stds
174             end
175         end
176     end
177 end
178
179 for (a,assay) in enumerate(["Total","polyA"])
180     for (m,ms) in enumerate(msvec)
181         for (d,age) in enumerate([6,8])
182             println("$assay & $ms & $age &
183                     $(round(stdmat[m,a,1,d],digits=2)) &
184                     $(round(stdmat[m,a,2,d],digits=2))\\ \\ \\ \\hline")
185         end
186     end
187 end
188
189 end

```

```

182
183 rr_h2a=[Normal(m,s) for (m,s) in
    ↳ zip(mean.(rand_rys["H2A"]),std.(rand_rys["H2A"]))]
184 rs_h2a=[Normal(m,s) for (m,s) in
    ↳ zip(mean.(rand_sib["H2A"]),std.(rand_sib["H2A"]))]
185 rand_h2a_joint=exp(sum(logccdf.(rr_h2a,[d.μ for d in rs_h2a])))

186
187 rr_h2b=[Normal(m,s) for (m,s) in
    ↳ zip(mean.(rand_rys["H2B"]),std.(rand_rys["H2B"]))]
188 rs_h2b=[Normal(m,s) for (m,s) in
    ↳ zip(mean.(rand_sib["H2B"]),std.(rand_sib["H2B"]))]
189 rand_h2b_joint=exp(sum(logccdf.(rr_h2b,[d.μ for d in rs_h2b])))

190
191 dtr_h2a=[Normal(m,s) for (m,s) in
    ↳ zip(mean.(dt_rys["H2A"]),std.(dt_rys["H2A"]))]
192 dts_h2a=[Normal(m,s) for (m,s) in
    ↳ zip(mean.(dt_sib["H2A"]),std.(dt_sib["H2A"]))]
193 dt_h2a_joint=exp(sum(logccdf.(dtr_h2a,[d.μ for d in dts_h2a])))

194
195 dtr_h2b=[Normal(m,s) for (m,s) in
    ↳ zip(mean.(dt_rys["H2B"]),std.(dt_rys["H2B"]))]
196 dts_h2b=[Normal(m,s) for (m,s) in
    ↳ zip(mean.(dt_sib["H2B"]),std.(dt_sib["H2B"]))]
197 dt_h2b_joint=exp(sum(logccdf.(dtr_h2b,[d.μ for d in dts_h2b])))

198
199 all_joint=rand_h2b_joint*dt_h2a_joint*dt_h2b_joint

```

---

### 17.8.24 /rys/rysp\_GMC\_NS.jl

---

```

1 using CSV,DataFrames,Distributions,GMC_NS,Measurements,Serialization
2
3 a38pth="/bench/PhD/datasets/A38.csv"
4 a49pth="/bench/PhD/datasets/A49.csv"
5
6 a38df=DataFrame(CSV.read(a38pth))
7 a38df.Group[1:10]::="sib"
8 a38df.Group[11:22]::="rys"
9 a49df=DataFrame(CSV.read(a49pth))
10 a49df=a49df[2:end,:]
11
12 a38df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
    ↳ eachrow(a38df)]

```

```

13
14 sib_measure_dict=Dict{String,Vector{Vector{Float64}}}()
15 rys_measure_dict=Dict{String,Vector{Vector{Float64}}}()
16
17 XCMZ=Vector{Float64}()
18 XEdU=Vector{Float64}()

19
20 for dict in [sib_measure_dict,rys_measure_dict]
21     dict["DCMZ"]=Vector{Float64}()
22     dict["VCMZ"]=Vector{Float64}()
23     dict["TCMZ"]=Vector{Float64}()
24     dict["EdU"]=Vector{Float64}()
25 end
26
27 for tdf in groupby(a49df, "Age")
28     age=unique(tdf."Age")[1]
29     push!(XCMZ,age);push!(XEdU,age)
30     xidx=length(XCMZ)
31     for sdf in groupby(tdf, "Rys/Sib")
32         sr=unique(sdf."Rys/Sib")[1]
33         for mdf in groupby(sdf, "D/V/P/WE")
34             if unique(mdf."D/V/P/WE")[1] == "D"
35                 if sr=="sib"
36                     push!(sib_measure_dict["DCMZ"],mdf."PCNA +ve")
37                     push!(sib_measure_dict["TCMZ"],mdf."PCNA +ve")
38                     push!(sib_measure_dict["EdU"],mdf."EdU +ve, PCNA
39                         +ve")
40                 else
41                     push!(rys_measure_dict["DCMZ"],mdf."PCNA +ve")
42                     push!(rys_measure_dict["TCMZ"],mdf."PCNA +ve")
43                     push!(rys_measure_dict["EdU"],mdf."EdU +ve, PCNA
44                         +ve")
45             end
46             elseif unique(mdf."D/V/P/WE")[1] == "V"
47                 if sr=="sib"
48                     push!(sib_measure_dict["VCMZ"],mdf."PCNA +ve")
49                     sib_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"
50                     sib_measure_dict["EdU"][xidx].+=mdf."EdU +ve, PCNA
51                         +ve"
52             else
53                 push!(rys_measure_dict["VCMZ"],mdf."PCNA +ve")
54                 rys_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"

```

```

52             rys_measure_dict["EdU"][xidx].+=mdf."EdU +ve, PCNA
53             ↪      +ve"
54         end
55     end
56   end
57 end
58
59 push!(XCMZ,5.)
60 for sdf in groupby(a38df,"Group")
61     xidx=length(XCMZ)
62     if unique(sdf."Group")[1]=="sib"
63         for mdf in groupby(sdf,"D/V")
64             if unique(mdf."D/V")[1] == "D"
65                 push!(sib_measure_dict["DCMZ"],mdf."PCNA +ve")
66                 push!(sib_measure_dict["TCMZ"],mdf."PCNA +ve")
67             else
68                 push!(sib_measure_dict["VCMZ"],mdf."PCNA +ve")
69                 sib_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"
70             end
71         end
72     else
73         for mdf in groupby(sdf,"D/V")
74             if unique(mdf."D/V")[1] == "D"
75                 push!(rys_measure_dict["DCMZ"],mdf."PCNA +ve")
76                 push!(rys_measure_dict["TCMZ"],mdf."PCNA +ve")
77             else
78                 push!(rys_measure_dict["VCMZ"],mdf."PCNA +ve")
79                 rys_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"
80             end
81         end
82     end
83 end
84
85 perm=sortperm(XCMZ)
86 for ms in ["DCMZ","VCMZ","TCMZ"]
87     sib_measure_dict[ms]=sib_measure_dict[ms][perm]
88     rys_measure_dict[ms]=rys_measure_dict[ms][perm]
89 end
90 XCMZ=[4.,10.]
91
92 for msd in [sib_measure_dict,rys_measure_dict]
93     for (k,v) in msd

```

```

94     for vec in v
95         if any(i→i=0,vec)
96             vec[findall(i→i=0,vec)]*=1e-3
97         end
98     end
99 end
100 end
101
102 gmc=GMC_DEFAULTS
103 gmc[1]=5
104 gmc[2]=1.e-15
105 gmc[end]=2e4
106
107 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]])
108 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
109
110 n_models=50
111
112 evdict=Dict{String,Measurement}()
113
114 for ms in ["TCMZ","EdU"]
115     evdict[ms*"Combined"]=measurement(0,0)
116     evdict[ms*"Separate"]=measurement(0,0)
117 end
118
119 max_μ=2000.
120 min_μ=0.
121 max_λ=500.
122 min_λ=1e-6
123
124 prior=[Uniform(min_μ,max_μ),Uniform(min_λ,max_λ)]
125 box=[min_μ max_μ;min_λ max_λ]
126
127 gmcdir="/bench/PhD/NGS_binaries/GMC_NS/rysp/"
128
129 for ms in ["TCMZ","EdU"]
130     for (n,x) in enumerate(XCMZ)
131         c_ens=gmcdir*/c_*ms*string(Int64(x))
132         c_obs=vcat(sib_measure_dict[ms][n],rysp_measure_dict[ms][n])
133         if isfile(c_ens*/ens")
134             e=deserialize(c_ens*/ens")
135         else

```

```

136         e=LogNormal_Ensemble(c_ens,n_models, c_obs, prior, box,
137             ↵   gmc ... )
138
139
140             ↵ evdict[ms*"_Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_di
141             ↵ lower_displays=lds, disp_rot_its=10000, converge_factor=1e-6)
142
143
144     for (msd,p) in
145         ↵ zip([sib_measure_dict,rys_measure_dict],["/s_","/r_"])
146         ens=gmcdir*p*ms*string(Int64(x))
147         obs=msd[ms][n]
148         if isfile(ens*"/ens")
149             e=deserialize(ens*"/ens")
150         else
151             ↵ e=LogNormal_Ensemble(ens,n_models, obs, prior, box,
152             ↵   gmc ... )
153         end
154
155             ↵ evdict[ms*"_Separate"]+=converge_ensemble!(e,backup=(true,10000),upper_
156             ↵ lower_displays=lds, disp_rot_its=10000,
157             ↵ converge_factor=1e-6)
158     end
159   end
160 end

```

---

### 17.8.25 /rys/ryspont.jl

---

```

1 using CSV,DataFrames,Distributions,NGRefTools,StatsBase,StatsPlots
2 gr()
3 default(legendfont = (8), guidefont = (12,:bold), tickfont = (8), guide =
4             ↵ "x")
5
6 a38pth="/bench/PhD/datasets/A38.csv"
7 a49pth="/bench/PhD/datasets/A49.csv"
8
9 a38df=DataFrame(CSV.read(a38pth))
10 a38df.Group[1:10]::"sib"
11 a38df.Group[11:22]::"rys"
12 a49df=DataFrame(CSV.read(a49pth))

```

```

13 a49df=a49df[2:end,:]

14
15 a38df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
   ↪ eachrow(a38df)]

16
17 sib_measure_dict=Dict{String,Vector{Vector{Float64}}}()
18 rys_measure_dict=Dict{String,Vector{Vector{Float64}}}()

19
20 XCMZ=Vector{Float64}()
21 XEdU=Vector{Float64}()

22
23 for dict in [sib_measure_dict,rys_measure_dict]
24     dict["DCMZ"]=Vector{Float64}()
25     dict["VCMZ"]=Vector{Float64}()
26     dict["TCMZ"]=Vector{Float64}()
27     dict["EdU"]=Vector{Float64}()
28 end

29
30 for tdf in groupby(a49df, "Age")
31     age=unique(tdf."Age")[1]
32     push!(XCMZ,age);push!(XEdU,age)
33     xidx=length(XCMZ)
34     for sdf in groupby(tdf, "Rys/Sib")
35         sr=unique(sdf."Rys/Sib")[1]
36         for mdf in groupby(sdf, "D/V/P/WE")
37             if unique(mdf."D/V/P/WE")[1] == "D"
38                 if sr=="sib"
39                     push!(sib_measure_dict["DCMZ"],mdf."PCNA +ve")
40                     push!(sib_measure_dict["TCMZ"],mdf."PCNA +ve")
41                     push!(sib_measure_dict["EdU"],mdf."EdU +ve, PCNA
   ↪ +ve")
42                 else
43                     push!(rys_measure_dict["DCMZ"],mdf."PCNA +ve")
44                     push!(rys_measure_dict["TCMZ"],mdf."PCNA +ve")
45                     push!(rys_measure_dict["EdU"],mdf."EdU +ve, PCNA
   ↪ +ve")
46                 end
47             elseif unique(mdf."D/V/P/WE")[1] == "V"
48                 if sr=="sib"
49                     push!(sib_measure_dict["VCMZ"],mdf."PCNA +ve")
50                     sib_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"
51                     sib_measure_dict["EdU"][xidx].+=mdf."EdU +ve, PCNA
   ↪ +ve"

```

```

52         else
53             push!(rys_measure_dict["VCMZ"], mdf."PCNA +ve")
54             rys_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"
55             rys_measure_dict["EdU"][xidx].+=mdf."EdU +ve, PCNA
56                         ↪ +ve"
57         end
58     end
59 end
60 end
61
62 push!(XCMZ, 5.)
63 for sdf in groupby(a38df, "Group")
64     xidx=length(XCMZ)
65     if unique(sdf."Group")[1] == "sib"
66         for mdf in groupby(sdf, "D/V")
67             if unique(mdf."D/V")[1] == "D"
68                 push!(sib_measure_dict["DCMZ"], mdf."PCNA +ve")
69                 push!(sib_measure_dict["TCMZ"], mdf."PCNA +ve")
70             else
71                 push!(sib_measure_dict["VCMZ"], mdf."PCNA +ve")
72                 sib_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"
73             end
74         end
75     else
76         for mdf in groupby(sdf, "D/V")
77             if unique(mdf."D/V")[1] == "D"
78                 push!(rys_measure_dict["DCMZ"], mdf."PCNA +ve")
79                 push!(rys_measure_dict["TCMZ"], mdf."PCNA +ve")
80             else
81                 push!(rys_measure_dict["VCMZ"], mdf."PCNA +ve")
82                 rys_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"
83             end
84         end
85     end
86 end
87
88 perm=sortperm(XCMZ)
89 for ms in ["DCMZ", "VCMZ", "TCMZ"]
90     sib_measure_dict[ms]=sib_measure_dict[ms][perm]
91     rys_measure_dict[ms]=rys_measure_dict[ms][perm]
92 end
93 XCMZ=XCMZ[perm]

```

```

94
95 sp_login_mts=[fit(MarginalTDist,log.(sib_measure_dict["TCMZ"][[n])) for n
  ↵ in 1:length(XCMZ)]
96 sp_mean=[exp(mean(mt)) for mt in sp_login_mts]
97 sp_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in sp_login_mts]
98 sp_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in sp_login_mts]
99
100 rp_login_mts=[fit(MarginalTDist,log.(rys_measure_dict["TCMZ"][[n])) for n
    ↵ in 1:length(XCMZ)]
101 rp_mean=[exp(mean(mt)) for mt in rp_login_mts]
102 rp_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in rp_login_mts]
103 rp_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in rp_login_mts]
104
105 log_mean_mass_comparator(rys_measure_dict["TCMZ"][[1]],sib_measure_dict["TCMZ"][[1]],labe
  ↵ $(XCMZ[1]) dpf sectional CMZ population","sib $(XCMZ[1]) dpf
  ↵ sectional CMZ population")
106 log_mean_mass_comparator(rys_measure_dict["TCMZ"][[end]],sib_measure_dict["TCMZ"][[end]],l
  ↵ $(XCMZ[[end]]) dpf sectional CMZ population","sib $(XCMZ[[end]]) dpf
  ↵ sectional CMZ population")
107
108 println("At 4dpf,
  ↵ $(mean(sib_measure_dict["EdU"][[1]]./sib_measure_dict["TCMZ"][[1]])) ±
  ↵ $(std(sib_measure_dict["EdU"][[1]]./sib_measure_dict["TCMZ"][[1]])) of
  ↵ sib PCNA+ve are EdU positive")
109
110 println("At 4dpf,
  ↵ $(mean(rys_measure_dict["EdU"][[1]]./rys_measure_dict["TCMZ"][[1]])) ±
  ↵ $(std(rys_measure_dict["EdU"][[1]]./rys_measure_dict["TCMZ"][[1]])) of
  ↵ rys PCNA+ve are EdU positive")
111
112 println("At 10dpf,
  ↵ $(mean(sib_measure_dict["EdU"][[end]]./sib_measure_dict["TCMZ"][[end]])) ±
  ↵ $(std(sib_measure_dict["EdU"][[end]]./sib_measure_dict["TCMZ"][[end]])) of
  ↵ sib PCNA+ve are EdU positive")
113
114
115 println("At 10dpf,
  ↵ $(mean(rys_measure_dict["EdU"][[end]]./rys_measure_dict["TCMZ"][[end]])) ±
  ↵ $(std(rys_measure_dict["EdU"][[end]]./rys_measure_dict["TCMZ"][[end]])) of
  ↵ rys PCNA+ve are EdU positive")
116
117 se_login_mts=[fit(MarginalTDist,log.(sib_measure_dict["EdU"][[n])) for n in
  ↵ 1:length(XEdU)]

```

```

118 se_mean=[exp(mean(mt)) for mt in se_logn_mts]
119 se_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in se_logn_mts]
120 se_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in se_logn_mts]
121
122 re_logn_mts=[fit(MarginalTDist,log.(filter(val→!iszero(val),rys_measure_dict["EdU"])[
    ↪ for n in 1:length(XEdU)])
123 re_mean=[exp(mean(mt)) for mt in re_logn_mts]
124 re_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in re_logn_mts]
125 re_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in re_logn_mts]
126
127
128 pcna_plt=plot(XCMZ,sp_mean,ribbon=(sp_lower,sp_upper),color=:green,
    ↪ label="sib mean", xlabel="Age (dpf)", ylabel="PCNA+ve CMZ Cells")
129 plot!(pcna_plt,XCMZ,rp_mean,ribbon=(rp_lower, rp_upper),color=:darkmagenta,
    ↪ label="rys mean", legend=:none)
130 scatter!(pcna_plt,vcat([[XCMZ[n] for i in
    ↪ 1:length(sib_measure_dict["TCMZ"][n])] for n in
    ↪ 1:length(XCMZ)] ... ),vcat(sib_measure_dict["TCMZ"] ... ),marker=:circle,markersize=:
    ↪ data")
131 scatter!(pcna_plt,vcat([[XCMZ[n] for i in
    ↪ 1:length(rys_measure_dict["TCMZ"][n])] for n in
    ↪ 1:length(XCMZ)] ... ),vcat(rys_measure_dict["TCMZ"] ... ),marker=:dtriangle,markersize=:
    ↪ data")
132 annotate!([(4.5,175,Plots.text("A",18))])
133
134 edu_plt=plot(XEdU,se_mean,ribbon=(se_lower,se_upper),color=:green,
    ↪ label="sib mean", xlabel="Age (dpf)", ylabel="EdU+ve CMZ Cells")
135 plot!(edu_plt,XEdU,re_mean,ribbon=(re_lower,re_upper),color=:darkmagenta,
    ↪ label="rys EdU")
136 scatter!(edu_plt,vcat([[XEdU[n] for i in
    ↪ 1:length(sib_measure_dict["EdU"][n])] for n in
    ↪ 1:length(XEdU)] ... ),vcat(sib_measure_dict["EdU"] ... ),marker=:circle,markersize=:3
    ↪ data")
137 scatter!(edu_plt,vcat([[XEdU[n] for i in
    ↪ 1:length(rys_measure_dict["EdU"][n])] for n in
    ↪ 1:length(XEdU)] ... ),vcat(rys_measure_dict["EdU"] ... ),marker=:dtriangle,markersize=:
    ↪ data")
138 annotate!([(4.5,150,Plots.text("B",18))])
139
140
141 combined=plt(plot(pcna_plt,edu_plt,layout=grid(1,2),size=(900,500))
142 savefig(combined_plt,"/bench/PhD/Thesis/images/rys/CMZontogeny.png")

```

---

### 17.8.26 /rys/bg\_models/BBM\_refinement\_analysis.jl

---

```

1 using BioBackgroundModels, Serialization
2
3 hmm_output = "/bench/PhD/NGS_binaries/BBM/refined_chains"
4 sample_output = "/bench/PhD/NGS_binaries/BBM/refinement_samples"
5
6 survey_folders = "/bench/PhD/NGS_binaries/BBM/refined_folders"
7
8 chains=deserialize(hmm_output)
9 sample_dfs = deserialize(sample_output)
10 training_sets, test_sets = split_obs_sets(sample_dfs)
11
12 report_folders=generate_reports(chains, test_sets)
13 serialize(survey_folders, report_folders)

```

---

### 17.8.27 /rys/bg\_models/BBM\_refinement\_prep.jl

---

```

1 #JOB FILEPATHS
2 #GFF3 feature database, FASTA genome and index paths
3 danio_gff_path = "/bench/PhD/seq/GRCz11/Danio rerio.GRCz11.94.gff3"
4 danio_genome_path =
5   ↳ "/bench/PhD/seq/GRCz11/GCA_000002035.4_GRCz11_genomic.fna"
6 danio_gen_index_path =
7   ↳ "/bench/PhD/seq/GRCz11/GCA_000002035.4_GRCz11_genomic.fna.fai"
6 #sample record and hmm serialisation output path
7 sample_output = "/bench/PhD/NGS_binaries/BBM/refinement_samples"
8 #non registered libs
9
10 #GENERAL SETUP
11 @info "Loading libraries ... "
12 using Distributed,Serialization
13
14 #JOB CONSTANTS
15 #CONSTANTS FOR GENOMIC SAMPLING
16 const sample_set_length = Int64(16e6)
17 const sample_window_min = 10
18 const sample_window_max = 3000
19 const perigenic_pad = 500
20 const partitions = 3 #exonic, periexonic, intragenic
21
22 #Setup sampling workers

```

```

23 @info "Spawning workers ... "
24 pool_size = partitions                                #number of workers to use
25 worker_pool = addprocs(pool_size) # add processes up to the worker pool
   ↳ size + 1 control process
26 @everywhere using BioBackgroundModels, Random
27 @everywhere Random.seed!(1)

28
29 #GET SEQUENCE OBSERVATIONS TO TRAIN AND TEST MODELS - PARTITIONING GENOME
   ↳ INTO EXONIC, PERIEXONIC, INTRAGENIC SEQUENCE
30 @info "Building observation db ... "
31 #generate the genomic sample dataframe
32 if isfile(sample_output) #if the sample DB has already been built for the
   ↳ current project, terminate
33   @error "Existing sample dataframe at $sample_output"
34 else #otherwise, build it from scratch
35   @info "Setting up sampling jobs ... "
36   #setup worker channels, input gets the genome partitions
37   channels = setup_sample_jobs(danio_genome_path, danio_gen_index_path,
   ↳ danio_gff_path, sample_set_length, sample_window_min,
   ↳ sample_window_max, perigenic_pad)

38
39   @info "Sampling genome ... "
40   sample_record_dfs=execute_sample_jobs(channels, worker_pool)

41
42   @info "Serializing sample dataframes ... "
43   serialize(sample_output, sample_record_dfs) #write the dataframes to
   ↳ file
44 end

45
46 rmprocs(worker_pool)
47
48 @info "Done sampling jobs!"

```

---

### 17.8.28 /rys/bg\_models/BBM\_sample\_prep.jl

---

```

1 #JOB FILEPATHS
2 #GFF3 feature database, FASTA genome and index paths
3 danio_gff_path = "/bench/PhD/seq/GRCz11/Danio rerio.GRCz11.94.gff3"
4 danio_genome_path =
   ↳ "/bench/PhD/seq/GRCz11/GCA_00002035.4_GRCz11_genomic.fna"
5 danio_gen_index_path =
   ↳ "/bench/PhD/seq/GRCz11/GCA_00002035.4_GRCz11_genomic.fna.fai"

```

```

6 #sample record and hmm serialisation output path
7 sample_output = "/bench/PhD/NGS_binaries/BBM/survey_samples"
8 #non registered libs
9
10 #GENERAL SETUP
11 @info "Loading libraries ... "
12 using Distributed, Serialization
13
14 #JOB CONSTANTS
15 #CONSTANTS FOR GENOMIC SAMPLING
16 const sample_set_length = Int64(4e6)
17 const sample_window_min = 10
18 const sample_window_max = 3000
19 const perigenic_pad = 500
20 const partitions = 3 #exonic, periexonic, intragenic
21
22 #Setup sampling workers
23 @info "Spawning workers ... "
24 pool_size = partitions                                #number of workers to use
25 worker_pool = addprocs(pool_size) # add processes up to the worker pool
   ↳ size
26 @everywhere using BioBackgroundModels, Random
27 @everywhere Random.seed!(1)
28
29 #GET SEQUENCE OBSERVATIONS TO TRAIN AND TEST MODELS - PARTITIONING GENOME
   ↳ INTO EXONIC, PERIEXONIC, INTRAGENIC SEQUENCE
30 @info "Building observation db ... "
31 #generate the genomic sample dataframe
32 if isfile(sample_output) #if the sample DB has already been built for the
   ↳ current project, terminate
   @error "Existing sample dataframe at $sample_output"
33 else #otherwise, build it from scratch
   @info "Setting up sampling jobs ... "
34   #setup worker channels, input gets the genome partitions
35   channels = setup_sample_jobs(danio_genome_path, danio_gen_index_path,
   ↳ danio_gff_path, sample_set_length, sample_window_min,
   ↳ sample_window_max, perigenic_pad)
36
37   @info "Sampling genome ... "
38   sample_record_dfs=execute_sample_jobs(channels, worker_pool)
39
40   @info "Serializing sample dataframes ... "
41
42

```

```

43     serialize(sample_output, sample_record_dfs) #write the dataframes to
        ↵   file
44 end
45
46 rmprocs(worker_pool)
47
48 @info "Done sampling jobs!"

```

---

### 17.8.29 /rys/bg\_models/BBM\_survey.jl

```

1 #JOB FILEPATHS
2 #sample record and hmm serialisation output path
3 sample_output = "/bench/PhD/NGS_binaries/BBM/survey_samples"
4 hmm_output = "/bench/PhD/NGS_binaries/BBM/survey_chains"
5
6 #GENERAL SETUP
7 @info "Loading libraries ... "
8 using BioBackgroundModels, DataFrames, Distributed, Serialization
9 include("/srv/git/rys_nucleosomes/aws/aws_wrangler.jl")
10
11 #JOB CONSTANTS
12 const replicates = 3 #repeat optimisation from this many seperately
        ↵   initialised samples from the prior
13 const Ks = [1,2,4,6] #mosaic class #s to test
14 const order_nos = [0,1,2] #DNA kmer order #s to test
15 const delta_thresh=1e-3 #stopping/convergence criterion (log probability
        ↵   difference btw subsequent EM iterates)
16 const max_iterates=15000
17
18 #LOAD SAMPLES
19 @info "Loading samples from $sample_output ... "
20 sample_dfs = deserialize(sample_output)
21
22 #BUILD TRAINING AND TEST SETS FROM SAMPLES
23 training_sets, test_sets = split_obs_sets(sample_dfs)
24
25 #PROGRAMATICALLY GENERATE Chain_ID Vector
26 job_ids=Vector{Chain_ID}()
27 for (obs_id, obs) in training_sets, K in Ks, order in order_nos, rep in
        ↵   1:replicates
28     push!(job_ids, Chain_ID(obs_id, K, order, rep))
29 end

```

```
30
31 #DISTRIBUTED CLUSTER CONSTANTS
32 load_dict=Dict{Int64,LoadConfig}()
33 local_config=LoadConfig(1:6,2:2)
34 remote_config=LoadConfig(1:6,0:1)
35 aws_instance_config=LoadConfig(1:6,0:2)
36 remote_machine = "10.0.0.3"
37 no_local_processes = 1
38 no_remote_processes = 1
39 #SETUP DISTRIBUTED BAUM WELCH LEARNERS
40 @info "Spawning local cluster workers ... "
41 worker_pool=addprocs(no_local_processes, topology=:master_worker)
42 for worker in worker_pool
43     load_dict[worker]=local_config
44 end
45
46 remote_pool=addprocs([(remote_machine,no_remote_processes)], tunnel=true,
47 ← topology=:master_worker)
48 for worker in remote_pool
49     load_dict[worker]=remote_config
50 end
51
52 worker_pool=vcat(worker_pool, remote_pool)
53
54 #AWS PARAMS
55 @info "Setting up AWS wrangling ... "
56
57 security_group_name="calc1"
58 security_group_desc="calculation group"
59 ami="ami-02df7e1881a08f163"
60 skeys="AWS"
61 instance_type="c5.4xlarge"
62 zone,spot_price=get_cheapest_zone(instance_type)
63 no_instances=1
64 instance_workers=8
65 bid=spot_price+.01
66
67 @assert bid ≥ spot_price
68
69 @info "Wrangling AWS instances ... "
70 aws_ips = spot_wrangle(no_instances, bid, security_group_name,
← security_group_desc, skeys, zone, ami, instance_type)
```

```

71 @info "Giving instances 90s to boot ... "
72 sleep(90)

73
74 @info "Spawning AWS cluster workers ... "
75 for ip in aws_ips
76     instance_pool=addprocs([(ip, instance_workers)], tunnel=true,
    ↪ topology=:master_worker, sshflags="-o StrictHostKeyChecking=no")
77     for worker in instance_pool
78         load_dict[worker]=aws_instance_config
79     end
80     global worker_pool=vcat(worker_pool, instance_pool)
81 end

82
83 @info "Loading worker libraries everywhere ... "
84 @everywhere using BioBackgroundModels
85 #INITIALIZE HMMS
86 @info "Setting up HMMs ... "
87 if isfile(hmm_output) #if some results have already been collected, load
    ↪ them
88     @info "Loading incomplete results ... "
89     hmm_results_dict = deserialize(hmm_output)
90 else #otherwise, pass a new results dict
91     @info "Initialising new HMM results file at $hmm_output"
92     hmm_results_dict = Dict{Chain_ID,Vector{EM_step}}()
93 end

94
95 em_jobset = setup_EM_jobs!(job_ids, training_sets;
    ↪ chains=hmm_results_dict)
96 execute_EM_jobs!(worker_pool, em_jobset..., hmm_output;
    ↪ load_dict=load_dict, delta_thresh=delta_thresh,
    ↪ max_iterates=max_iterates)

```

---

### 17.8.30 /rys/bg\_models/BBM\_survey\_analysis.jl

---

```

1 using BioBackgroundModels, Serialization, Plots, Measures
2 import Measures:mm
3 import Plots:Plot

4
5 hmm_output = "/bench/PhD/NGS_binaries/BBM/survey_chains"
6 sample_output = "/bench/PhD/NGS_binaries/BBM/survey_samples"
7
8 survey_folders = "/bench/PhD/NGS_binaries/BBM/survey_folders"

```

```

9
10 chains=deserialize(hmm_output)
11 sample_dfs = deserialize(sample_output)
12 training_sets, test_sets = split_obs_sets(sample_dfs)
13
14 report_folders=generate_reports(chains, test_sets)
15 serialize(survey_folders, report_folders)
16
17 partplts=Vector{Plot}()
18
19 for o in [0,1,2]
20     for p in ["exon","intergenic","periexonic"]
21         rf=report_folders[p]
22         pr=rf.partition_report
23         od=pr.orddict
24         or=od[o]
25
26         p=="exon" ? (ylbl="Order $o lh"; yfmt=y→y; lm=0mm) : (ylbl="";
27             ↵ yfmt=_→""; lm=-7mm)
28         o==2 ? (xlbl=p; xfmt=x→Int64(x); bm=0mm) : (xlbl=""; xfmt=_→";
29             ↵ bm=-7mm)
30
31         plt=scatter(or.converged_K,or.converged_lh, marker=:cross,
32             ↵ markersize=15, markercolor=:black, label="Ord. $o model lhs",
33             ↵ xlabel=xlbl, ylabel=ylbl, legend=nothing, xformatter=xfmt,
34             ↵ yformatter=yfmt, xticks=[1,2,4,6],
35             ↵ leftmargin=lm,bottommargin=bm)
36         plot!(unique(or.converged_K),[pr.naive_lh for l in
37             ↵ 1:length(unique(or.converged_K))], label="Naive lh",
38             ↵ color=:darkmagenta,linewidth=2.5)
39         push!(partplts,plt)
40     end
41 end
42
43 combined=plot(partplts...,layout=grid(3,3),link=:xy, size=(600,600))
44 savefig(combined,"/bench/PhD/Thesis/images/rys/bghmm.png")

```

---

### 17.8.31 /rys/bg\_models/BBM\_survey\_refinement.jl

---

```

1 #JOB FILEPATHS
2 hmm_output = "/bench/PhD/NGS_binaries/BBM/survey_chains"
3 refined_path = "/bench/PhD/NGS_binaries/BBM/refined_chains"
4 sample_output = "/bench/PhD/NGS_binaries/BBM/refinement_samples"

```

```

5
6 survey_folders = "/bench/PhD/NGS_binaries/BBM/survey_folders"
7
8 #GENERAL SETUP
9 @info "Loading libraries ... "
10 using BioBackgroundModels, DataFrames, Distributed, Serialization
11
12 const delta_thresh=1e-4 #stopping/convergence criterion (log probability
    ↪ difference btw subsequent EM iterates)
13 const max_iterates=15000
14
15 #LOAD SAMPLES
16 @info "Loading samples from $sample_output ... "
17 sample_dfs = deserialize(sample_output)
18
19 #BUILD TRAINING AND TEST SETS FROM SAMPLES
20 training_sets, test_sets = split_obs_sets(sample_dfs)
21
22 #GENERATE Chain_ID Vector AND CHAINS SUBSET
23 @info "Loading reports from $survey_folders"
24 report_folders=deserialize(survey_folders)
25 job_ids=Vector{Chain_ID}()
26 for (partition, folder) in report_folders
    report=folder.partition_report
    for id in report.best_repset
        push!(job_ids,id)
    end
end
32
33 if isfile(refined_path) #if some results have already been collected,
    ↪ load them
34     @info "Loading incomplete results ... "
35     refined_chains=deserialize(refined_path)
36 else
    @info "Initialising new HMM results file at $refined_path"
38     chains = deserialize(hmm_output)
39     refined_chains=Dict{Chain_ID,Vector{EM_step}}()
40     for job_id in job_ids
        refined_chains[job_id]=[EM_step(1, chains[job_id][end].hmm, 0, 0,
            ↪ false)]
    end
43 end
44

```

```

45 #DISTRIBUTED CLUSTERS CONSTANTS
46 remote_machine = "10.0.0.3"
47 no_local_processes = 1
48 no_remote_processes = 1
49 load_dict=Dict{Int64,LoadConfig}()
50 local_config=LoadConfig(1:6,0:2)
51 remote_config=LoadConfig(1:6,0:2)

52
53 #SETUP DISTRIBUTED BAUM WELCH LEARNERS
54 @info "Spawning local cluster workers ..."
55 worker_pool=addprocs(no_local_processes, topology=:master_worker)
56 for worker in worker_pool
57     load_dict[worker]=local_config
58 end

59
60 remote_pool=addprocs([(remote_machine,no_remote_processes)], tunnel=true,
61     ↳ topology=:master_worker)
61 for worker in remote_pool
62     load_dict[worker]=remote_config
63 end

64
65 worker_pool=vcat(worker_pool, remote_pool)

66
67 @info "Loading worker libraries everywhere ..."
68 @everywhere using BioBackgroundModels

69
70 em_jobset = setup_EM_jobs!(job_ids, training_sets;
71     ↳ delta_thresh=delta_thresh, chains=refined_chains)
71 execute_EM_jobs!(worker_pool, em_jobset..., refined_path;
72     ↳ delta_thresh=delta_thresh, max_iterates=max_iterates,
73     ↳ load_dict=load_dict)

```

---

### 17.8.32 /rys/nested\_sampling/dif\_pos\_assembly.jl

---

```

1 @info "Setting up for job ..."
2 #JOB FILEPATHS
3 sib_wms_path =
4     ↳ "/bench/PhD/NGS_binaries/BMI/sib_nuc_position_sequences.fa_wms.tr"
4 rys_wms_path =
5     ↳ "/bench/PhD/NGS_binaries/BMI/rys_nuc_position_sequences.fa_wms.tr"
5
6 sib_df_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_positions"

```

```

7 rys_df_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_positions"
8 combined_df_binary =
9   ↵ "/bench/PhD/NGS_binaries/BMI/combined_diff_positions"
10
10 sib_code_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_codes"
11 rys_code_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_codes"
12 combined_code_binary = "/bench/PhD/NGS_binaries/BMI/combined_diff_codes"
13
14 sib_diff_bg = "/bench/PhD/NGS_binaries/BMI/sib_diff_bg"
15 rys_diff_bg = "/bench/PhD/NGS_binaries/BMI/rys_diff_bg"
16 combined_diff_bg = "/bench/PhD/NGS_binaries/BMI/combined_diff_bg"
17
18 sib_e_pth = "/bench/PhD/NGS_binaries/BMI/sib_e"
19 rys_e_pth = "/bench/PhD/NGS_binaries/BMI/rys_e"
20 combined_e_pth = "/bench/PhD/NGS_binaries/BMI/combined_e"
21
22 #JOB CONSTANTS
23 const ensemble_size = 500
24 const no_sources = 8
25 const source_min_bases = 3
26 const source_max_bases = 10
27 @assert source_min_bases < source_max_bases
28 const source_length_range= source_min_bases:source_max_bases
29 const mixing_prior = .07
30 @assert mixing_prior ≥ 0 & mixing_prior ≤ 1
31
32 @info "Loading master libraries ... "
33 using Distributed, Distributions, Serialization
34
35 @info "Adding workers ... "
36 no_local_procs=2
37 worker_pool=addprocs(no_local_procs, topology=:master_worker)
38
39 @info "Loading libraries everywhere ... "
40 @everywhere using BioMotifInference, Random
41 Random.seed!(myid()*10000)
42
43 @info "Assembling uninformative source priors ... "
44 sib_ui_sp = BioMotifInference.assemble_source_priors(no_sources,
45   ↵ Vector{Matrix{Float64}}())
45 rys_ui_sp = BioMotifInference.assemble_source_priors(no_sources,
46   ↵ Vector{Matrix{Float64}}())

```

```

46 combined_ui_sp = BioMotifInference.assemble_source_priors(no_sources,
    ↵ Vector{Matrix{Float64}}())
47
48 @info "Loading BGHMM likelihood matrix binaries ... "
49 sib_matrix=deserialize(sib_diff_bg)
50 rys_matrix=deserialize(rys_diff_bg)
51 combined_matrix=deserialize(combined_diff_bg)
52
53 @info "Loading coded observation sets ... "
54 sib_obs = deserialize(sib_code_binary)
55 rys_obs = deserialize(rys_code_binary)
56 combined_obs = deserialize(combined_code_binary)
57
58 @info "Assembling sib IPM ensemble on uninformative priors ... "
59 isfile(string(sib_e_pth,'/','ens')) ? (sib_e =
    ↵ deserialize(string(sib_e_pth,'/','ens'))) :
60     (sib_e = BioMotifInference.IPM_Elenseble(worker_pool, sib_e_pth,
    ↵ ensemble_size, sib_ui_sp, (falses(0,0), mixing_prior),
    ↵ sib_matrix, sib_obs, source_length_range, posterior_switch=true);
    ↵ serialize(string(sib_e_pth,'/','ens'),sib_e))
61 sib_e=[]; Base.GC.gc();
62
63 @info "Assembling rys IPM ensemble on uninformative priors ... "
64 isfile(string(rys_e_pth,'/','ens')) ? (rys_e =
    ↵ deserialize(string(rys_e_pth,'/','ens'))) :
65     (rys_e = BioMotifInference.IPM_Elenseble(worker_pool, rys_e_pth,
    ↵ ensemble_size, rys_ui_sp, (falses(0,0), mixing_prior),
    ↵ rys_matrix, rys_obs, source_length_range, posterior_switch=true);
    ↵ serialize(string(rys_e_pth,'/','ens'),rys_e))
66 rys_e=[]; Base.GC.gc();
67
68
69 @info "Assembling combined IPM ensemble on uninformative priors ... "
70 isfile(string(combined_e_pth,'/','ens')) ? (combined_e =
    ↵ deserialize(string(combined_e_pth,'/','ens'))) :
71     (combined_e = BioMotifInference.IPM_Elenseble(worker_pool,
    ↵ combined_e_pth, ensemble_size, combined_ui_sp, (falses(0,0),
    ↵ mixing_prior), combined_matrix, combined_obs,
    ↵ source_length_range, posterior_switch=true);
    ↵ serialize(string(combined_e_pth,'/','ens'),combined_e))
72 combined_e=[]; Base.GC.gc();
73
74 rmprocs(worker_pool)

```

```
75  
76 @info "Job done!"
```

---

### 17.8.33 /rys/nested\_sampling/dif\_pos\_learner.jl

```
1 using Distributed, Distributions, Serialization  
2 include("/srv/git/rys_nucleosomes/aws/aws_wrangler.jl")  
3  
4 sib_e_pth = "/bench/PhD/NGS_binaries/BMI/sib_e"  
5 rys_e_pth = "/bench/PhD/NGS_binaries/BMI/rys_e"  
6 combined_e_pth = "/bench/PhD/NGS_binaries/BMI/combined_e"  
7  
8 @info "Assembling worker pool ... "  
9  
10 #DISTRIBUTED CLUSTERS CONSTANTS  
11 remote_machine = "10.0.0.3"  
12 no_local_processes = 2  
13 no_remote_processes = 6  
14  
15 @info "Spawning local cluster workers ... "  
16 worker_pool=addprocs(no_local_processes, topology=:master_worker)  
17 remote_pool=addprocs([(remote_machine,no_remote_processes)], tunnel=true,  
    ↳ topology=:master_worker)  
18  
19 worker_pool=vcat(worker_pool, remote_pool)  
20  
21 #AWS PARAMS  
22 @info "Setting up AWS wrangling ... "  
23  
24 security_group_name="calc1"  
25 security_group_desc="calculation group"  
26 ami="ami-0bb0f5609b995d39e"  
27 skeys="AWS"  
28 instance_type="c5a.24xlarge"  
29 zone,spot_price=get_cheapest_zone(instance_type)  
30 no_instances=5  
31 instance_workers=48  
32 bid=spot_price+.1  
33  
34 @assert bid ≥ spot_price  
35  
36 @info "Wrangling AWS instances ... "
```

```

37 aws_ips = spot_wrangle(no_instances, bid, security_group_name,
  ↳ security_group_desc, skeys, zone, ami, instance_type)
38 @info "Giving instances 150s to boot ... "
39 sleep(150)

40
41 # aws_ips=["18.223.214.57","18.224.4.81",
  ↳ "3.17.13.150","3.21.113.229","18.222.211.44"]
42
43
44 @info "Spawning AWS cluster workers ... "
45 for ip in aws_ips
46     instance_pool=addprocs([(ip, instance_workers)], tunnel=true,
  ↳ topology=:master_worker, sshflags="-o StrictHostKeyChecking=no")
47     global worker_pool=vcat(worker_pool, instance_pool)
48 end

49
50 @info "Loading worker libraries everywhere ... "
51 @everywhere using BioMotifInference, Random
52 @everywhere Random.seed!(myid() * 100000)

53
54 #JOB CONSTANTS
55 funcvec=full_perm_funcvec
56 models_to_permute=10000
57 func_limit=25
58 push!(funcvec, BioMotifInference.perm_src_fit_mix)
59 push!(funcvec, BioMotifInference.random_decorrelate)

60
61 min_clamps=fill(.01,length(funcvec))
62 min_clamps[2:3] *= .1 #perm_src_fit_mix & permute_mix
63 min_clamps[8] = .1 #difference_merge
64 max_clamps=fill(.5,length(funcvec))
65 max_clamps[6:7] *= .15 #ss and am
66 max_clamps[9] = .15 #sim merge

67
68 initial_weights= ones(length(funcvec))./length(funcvec)
69 # override_weights=fill(.034375,length(funcvec))
70 # override_weights[6:9] *= .1;override_weights[13:14] *= .1625
71 # override_time=20.

72
73 args=[Vector{Tuple{Symbol,Any}}() for i in 1:length(funcvec)]
74 args[end-1]=[:weight_shift_freq,0.],(:length_change_freq,1.),(:length_perm_range,1:1)
75 args[end]=[:iterates,50],(:source_permute_freq,.3),(:mix_move_range,1:10)]

76

```

```

77 instruct = Permute_Instruct(funcvec, initial_weights, models_to_permute,
    ↵ func_limit;min_clmps=min_clamps, max_clmps=max_clamps, args=args)
78
79 display_rotation=[true,10,1,[[:tuning_disp,:lh_disp,:src_disp],[:conv_plot,:liwi_disp
80
81 @info "Beginning nested sampling for sib ensemble ... "
82 sib_e=deserialize(sib_e_pth*"/ens")
83 sib_logZ = converge_ensemble!(sib_e, instruct, worker_pool,
    ↵ converge_criterion="compression", converge_factor=150.,
    ↵ backup=true,25), tuning_disp=true,lh_disp=true,src_disp=true,
    ↵ disp_rotate_inst=display_rotation)
84 sib_e=[]; Base.GC.gc();
85
86 @info "Beginning nested sampling for rys ensemble ... "
87 rys_e=deserialize(rys_e_pth*"/ens")
88 rys_logZ = converge_ensemble!(rys_e, instruct, worker_pool,
    ↵ converge_criterion="compression", converge_factor=150.,
    ↵ backup=true,25), tuning_disp=true,lh_disp=true,src_disp=true,
    ↵ disp_rotate_inst=display_rotation)
89 rys_e=[]; Base.GC.gc();
90
91 @info "Beginning nested sampling for combined ensemble ... "
92 combined_e=deserialize(combined_e_pth*"/ens")
93 combined_logZ = converge_ensemble!(combined_e, instruct, worker_pool,
    ↵ converge_criterion="compression", converge_factor=150.,
    ↵ backup=true,
    ↵ tuning_disp=true,lh_disp=true,src_disp=true,disp_rotate_inst=display_rotation)
94
95 joint_logZ=rys_logZ+sib_logZ
96 evratio=joint_logZ-combined_logZ
97 sig=evratio.val/evratio.err
98
99 println("$((round(sib_logZ,digits=3)) & $(round(rys_logZ,digits=3)) &
    ↵ {\\bf $(round(joint_logZ,digits=3))} &
    ↵ $(round(combined_logZ,digits=3)) & $(round(evratio,digits=3)) &
    ↵ $(round(sig,digits=1)) \\\\ \\hline")
100
101 rmprocs(worker_pool)
102
103 @info "Job done!"

```

---

### 17.8.34 /rys/nested\_sampling/dif\_pos\_sample\_prep.jl

---

```

1 using BioSequences, Distributed, DataFrames, BioBackgroundModels, CSV,
2   ↵ Serialization
3 import StatsBase:sample
4
5 include("/bench/PhD/Thesis/Analyses/rys/position_analysis/position_overlap.jl")
6
7 #JOB PATHS AND CONSTANTS
8
9 sample_rows = 7092 #2 million bases
10
11 sib_pos =
12   ↵ "/bench/PhD/danpos_results/pooled/sib.Fnor.smooth.positions.xls"
13 rys_pos =
14   ↵ "/bench/PhD/danpos_results/pooled/rys.Fnor.smooth.positions.xls"
15
16 danio_genome_path =
17   ↵ "/bench/PhD/seq/GRCz11/GCA_000002035.4_GRCz11_genomic.fna"
18 danio_gen_index_path =
19   ↵ "/bench/PhD/seq/GRCz11/GCA_000002035.4_GRCz11_genomic.fna.fai"
20 danio_gff_path = "/bench/PhD/seq/GRCz11/Danio rerio.GRCz11.94.gff3"
21
22 refined_folders_path = "/bench/PhD/NGS_binaries/BBM/refined_folders"
23 selected_hmms = "/bench/PhD/NGS_binaries/BBM/selected_hmms"
24
25 sib_df_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_positions"
26 rys_df_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_positions"
27 combined_df_binary =
28   ↵ "/bench/PhD/NGS_binaries/BMI/combined_diff_positions"
29
30 sib_code_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_codes"
31 rys_code_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_codes"
32 combined_code_binary = "/bench/PhD/NGS_binaries/BMI/combined_diff_codes"
33
34 sib_diff_bg = "/bench/PhD/NGS_binaries/BMI/sib_diff_bg"
35 rys_diff_bg = "/bench/PhD/NGS_binaries/BMI/rys_diff_bg"
36 combined_diff_bg = "/bench/PhD/NGS_binaries/BMI/combined_diff_bg"
37
38 sib_fa = "/bench/PhD/thicweed_results/sib_nuc_position_sequences.fa"
39 sib_arch =
40   ↵ "/bench/PhD/thicweed_results/sib_nuc_position_sequences.fa_archs.txt"

```

```

35 rys_fa = "/bench/PhD/thicweed_results/rys_nuc_position_sequences.fa"
36 rys_arch =
   ↳ "/bench/PhD/thicweed_results/rys_nuc_position_sequences.fa_archs.txt"
37
38 @info "Reading from position.xls ... "
39 sib_df=CSV.read(sib_pos)
40 rys_df=CSV.read(rys_pos)

41
42 @info "Mapping ... "
43 map_positions!(sib_df, rys_df)
44 map_positions!(rys_df, sib_df)

45
46 @info "Making differential position dataframes ... "
47 sib_diff_df = deepcopy(sib_df[findall(iszero,sib_df.mapped_pos),:])
48 sib_diff_df = sib_diff_df[findall(!isequal("MT"), sib_diff_df.chr),:]
49 rys_diff_df = deepcopy(rys_df[findall(iszero,rys_df.mapped_pos),:])
50 rys_diff_df = rys_diff_df[findall(!isequal("MT"), rys_diff_df.chr),:]

51
52 add_position_sequences!(sib_diff_df, danio_genome_path,
   ↳ danio_gen_index_path)
53 add_position_sequences!(rys_diff_df, danio_genome_path,
   ↳ danio_gen_index_path)

54
55 @info "Filtering ambiguous sequences ... "
56 deleterows!(sib_diff_df, [hasambiguity(seq) for seq in sib_diff_df.seq])
57 deleterows!(rys_diff_df, [hasambiguity(seq) for seq in rys_diff_df.seq])

58
59 @info "Sampling subsets of sequences ... "

60
61 sib_diff_df=sib_diff_df[sample(axes(sib_diff_df, 1), sample_rows;
   ↳ replace=false, ordered=true), :]
62 rys_diff_df=rys_diff_df[sample(axes(rys_diff_df, 1), sample_rows;
   ↳ replace=false, ordered=true), :]

63
64 @info "Masking positions by genome partition and strand ... "
65 BioBackgroundModels.add_partition_masks!(sib_diff_df, danio_gff_path,
   ↳ 500, (:chr,:seq,:start))
66 BioBackgroundModels.add_partition_masks!(rys_diff_df, danio_gff_path,
   ↳ 500, (:chr,:seq,:start))

67
68 @info "Obtaining cluster data ... "
69 get_cluster!(sib_diff_df, sib_fa, sib_arch)
70 get_cluster!(rys_diff_df, rys_fa, rys_arch)

```

```

71
72 @info "Serializing dataframes ... "
73 combined_diff_df = vcat(sib_diff_df, rys_diff_df)
74
75 serialize(sib_df_binary, sib_diff_df)
76 serialize(rys_df_binary, rys_diff_df)
77 serialize(combined_df_binary, combined_diff_df)
78
79 @info "Creating coded observation sets ... "
80 sib_codes = observation_setup(sib_diff_df, order=0, symbol=:seq)
81 rys_codes = observation_setup(rys_diff_df, order=0, symbol=:seq)
82 combined_codes = observation_setup(combined_diff_df, order=0,
83   ↪   symbol=:seq)
84
85 @info "Serializing coded observation sets ... "
86 serialize(sib_code_binary, Matrix(transpose(sib_codes)))
87 serialize(rys_code_binary, Matrix(transpose(rys_codes)))
88 serialize(combined_code_binary, Matrix(transpose(combined_codes)))
89
90 @info "Setting up for BBM likelihood calculations ... "
91 BHMM_dict = Dict{String,BHMM}()
92 refined_folders=deserialize(refined_folders_path)
93 for (part, folder) in refined_folders
94   BHMM_dict[part]=folder.partition_report.best_model[2]
95 end
96
97 serialize(selected_hmms,BHMM_dict)
98
99 @info "Performing calculations ... "
100 sib_lh_matrix = BGHMM_likelihood_calc(sib_diff_df, BHMM_dict,
101   ↪   symbol=:seq)
102 rys_lh_matrix = BGHMM_likelihood_calc(rys_diff_df, BHMM_dict,
103   ↪   symbol=:seq)
104 combined_lh_matrix = hcat(sib_lh_matrix,rys_lh_matrix)
105
106 @info "Serializing background matrices ... "
107 serialize(sib_diff_bg, sib_lh_matrix)
108 serialize(rys_diff_bg, rys_lh_matrix)
109 serialize(combined_diff_bg, combined_lh_matrix)
110
111 @info "Job done!"

```

---

### 17.8.35 /rys/nested\_sampling/local\_test.jl

---

```

1 using Distributed, Distributions, Serialization
2
3 test_e_pth = "/bench/PhD/NGS_binaries/BMI/test_e"
4
5 @info "Assembling worker pool ... "
6
7 #DISTRIBUTED CLUSTERS CONSTANTS
8 remote_machine = "10.0.0.3"
9 no_local_processes = 1
10
11 @info "Spawning local cluster workers ... "
12 worker_pool=addprocs(no_local_processes, topology=:master_worker)
13
14 @info "Loading worker libraries everywhere ... "
15 @everywhere using BioMotifInference, Random
16 @everywhere Random.seed!(myid()*100000)
17
18 #JOB CONSTANTS
19 models_to_permute=5000
20 func_limit=30
21 clamp=.02
22 funcvec=full_perm_funcvec
23 push!(funcvec, BioMotifInference.permute_source)
24 push!(funcvec, BioMotifInference.permute_source)
25 args=[Vector{Tuple{Symbol,Any}}() for i in 1:length(funcvec)]
26 args[end-1]=[:weight_shift_freq,0.],(:length_change_freq,1.),(:length_perm_range,1:1)
27 args[end]=[:weight_shift_freq,.1],(:length_change_freq,0.),(:weight_shift_dist,Uniform(0,1))
28
29 instruct = Permute_Instruct(funcvec,
30   ↳ ones(length(funcvec))./length(funcvec), models_to_permute,
31   ↳ func_limit, clamp; args=args)
32
33 display_rotation=[true,10,1,[[:tuning_disp,:lh_disp,:src_disp],[:conv_plot,:liwi_disp
34
35 @info "Beginning nested sampling for test ensemble ... "
36 test_e=deserialize(test_e_pth*"/ens")
37 converge_ensemble!(test_e, instruct, worker_pool, .001,
38   ↳ backup=(true,1), tuning_disp=true, lh_disp=true, src_disp=true,
39   ↳ disp_rotate_inst=display_rotation)
40
41 rmprocs(worker_pool)

```

```

38
39 @info "Job done!"
```

---

### 17.8.36 /rys/nested\_sampling/overall\_naive.jl

```

1 using BioMotifInference, BioSequences, DataFrames, BioBackgroundModels,
   ↵ CSV, Serialization
2 import BioMotifInference: IPM_likelihood
3
4 include("/bench/PhD/Thesis/Analyses/rys/position_analysis/position_overlap.jl")
5
6 sib_pos =
   ↵ "/bench/PhD/danpos_results/pooled/sib.Fnor.smooth.positions.xls"
7 rys_pos =
   ↵ "/bench/PhD/danpos_results/pooled/rys.Fnor.smooth.positions.xls"
8
9 danio_genome_path =
   ↵ "/bench/PhD/seq/GRCz11/GCA_000002035.4_GRCz11_genomic.fna"
10 danio_gen_index_path =
   ↵ "/bench/PhD/seq/GRCz11/GCA_000002035.4_GRCz11_genomic.fna.fai"
11 danio_gff_path = "/bench/PhD/seq/GRCz11/Danio rerio.GRCz11.94.gff3"
12
13 refined_folders_path = "/bench/PhD/NGS_binaries/BBM/refined_folders"
14 selected_hmms = "/bench/PhD/NGS_binaries/BBM/selected_hmms"
15
16 sib_df_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_positions"
17 rys_df_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_positions"
18 combined_df_binary =
   ↵ "/bench/PhD/NGS_binaries/BMI/combined_diff_positions"
19
20 @info "Reading from position.xls ... "
21 sib_df=CSV.read(sib_pos)
22 rys_df=CSV.read(rys_pos)
23
24 @info "Mapping ... "
25 map_positions!(sib_df, rys_df)
26 map_positions!(rys_df, sib_df)
27
28 @info "Making differential position dataframes ... "
29 sib_diff_df = deepcopy(sib_df[findall(iszero,sib_df.mapped_pos),:])
30 sib_diff_df = sib_diff_df[findall(!isequal("MT"), sib_diff_df.chr),:]
31 rys_diff_df = deepcopy(rys_df[findall(iszero,rys_df.mapped_pos),:])
```

```

32 rys_diff_df = rys_diff_df[findall(!isequal("MT"), rys_diff_df.chr),:]
33 sib_mapped_df = deepcopy(sib_df[findall(!iszero,sib_df.mapped_pos),:])
34 sib_mapped_df = sib_mapped_df[findall(x→x>0, sib_mapped_df.start),:]
35 sib_mapped_df = sib_mapped_df[findall(!isequal("MT"),
36   ↵ sib_mapped_df.chr),:]
37 rys_mapped_df = deepcopy(rys_df[findall(!iszero,rys_df.mapped_pos),:])
38 rys_mapped_df = rys_mapped_df[findall(x→x>0, rys_mapped_df.start),:]
39 rys_mapped_df = rys_mapped_df[findall(!isequal("MT"),
40   ↵ rys_mapped_df.chr),:]
41
42 add_position_sequences!(sib_diff_df, danio_genome_path,
43   ↵ danio_gen_index_path)
44 add_position_sequences!(rys_diff_df, danio_genome_path,
45   ↵ danio_gen_index_path)
46 add_position_sequences!(sib_mapped_df, danio_genome_path,
47   ↵ danio_gen_index_path)
48 add_position_sequences!(rys_mapped_df, danio_genome_path,
49   ↵ danio_gen_index_path)
50
51 @info "Filtering ambiguous sequences ... "
52 deleterows!(sib_diff_df, [hasambiguity(seq) for seq in sib_diff_df.seq])
53 deleterows!(rys_diff_df, [hasambiguity(seq) for seq in rys_diff_df.seq])
54 deleterows!(sib_mapped_df, [hasambiguity(seq) for seq in
55   ↵ sib_mapped_df.seq])
56 deleterows!(rys_mapped_df, [hasambiguity(seq) for seq in
57   ↵ rys_mapped_df.seq])
58
58 @info "Masking positions by genome partition and strand ... "
59 BioBackgroundModels.add_partition_masks!(sib_diff_df, danio_gff_path,
60   ↵ 500, (:chr,:seq,:start))
61 BioBackgroundModels.add_partition_masks!(rys_diff_df, danio_gff_path,
62   ↵ 500, (:chr,:seq,:start))
63 BioBackgroundModels.add_partition_masks!(sib_mapped_df, danio_gff_path,
64   ↵ 500, (:chr,:seq,:start))
65 BioBackgroundModels.add_partition_masks!(rys_mapped_df, danio_gff_path,
66   ↵ 500, (:chr,:seq,:start))
67
67 @info "Creating coded observation sets ... "
68 sib_codes = Matrix(transpose(observation_setup(sib_diff_df, order=0,
69   ↵ symbol=:seq)))
70 rys_codes = Matrix(transpose(observation_setup(rys_diff_df, order=0,
71   ↵ symbol=:seq)))

```

```

60 sib_mapped_codes = Matrix(transpose(observation_setup(sib_mapped_df,
  ↵ order=0, symbol=:seq)))
61 rys_mapped_codes = Matrix(transpose(observation_setup(rys_mapped_df,
  ↵ order=0, symbol=:seq)))
62
63 @info "Setting up for BBM likelihood calculations ... "
64 BHMM_dict = Dict{String,BHMM}()
65 refined_folders=deserialize(refined_folders_path)
66 for (part, folder) in refined_folders
67   BHMM_dict[part]=folder.partition_report.best_model[2]
68 end
69
70 sib_lh_matrix = BGHMM_likelihood_calc(sib_diff_df, BHMM_dict,
  ↵ symbol=:seq)
71 rys_lh_matrix = BGHMM_likelihood_calc(rys_diff_df, BHMM_dict,
  ↵ symbol=:seq)
72 sibm_lh_matrix= BGHMM_likelihood_calc(sib_mapped_df, BHMM_dict,
  ↵ symbol=:seq)
73 rysm_lh_matrix = BGHMM_likelihood_calc(rys_mapped_df, BHMM_dict,
  ↵ symbol=:seq)
74
75 sib_e_pth = "/bench/PhD/NGS_binaries/BMI/sib_e"
76 rys_e_pth = "/bench/PhD/NGS_binaries/BMI/rys_e"
77
78 sibe=deserialize(sib_e_pth*"/ens")
79 ryse=deserialize(rys_e_pth*"/ens")
80
81 sib_MAP=deserialize(sibe.models[findmax([rec.log_Li for rec in
  ↵ sibe.models])[2]].path)
82 rys_MAP=deserialize(ryse.models[findmax([rec.log_Li for rec in
  ↵ ryse.models])[2]].path)
83
84 sib_bg_lh = IPM_likelihood(sib_MAP.sources, sib_codes,
  ↵ [findfirst(iszero,sib_codes[:,o])-1 for o in 1:size(sib_codes,2)],
  ↵ sib_lh_matrix, falses(size(sib_codes,2),length(sib_MAP.sources)))
85
86 sibm_bg_lh = IPM_likelihood(sib_MAP.sources, sib_mapped_codes,
  ↵ [findfirst(iszero,sib_mapped_codes[:,o])-1 for o in
  ↵ 1:size(sib_mapped_codes,2)], sibm_lh_matrix,
  ↵ falses(size(sib_mapped_codes,2),length(sib_MAP.sources)))
87

```

```

88 rys_bg_lh = IPM_likelihood(rys_MAP.sources, rys_codes,
→ [findfirst(iszero, rys_codes[:,o])-1 for o in 1:size(rys_codes,2)],
→ rys_lh_matrix, falses(size(rys_codes,2),length(rys_MAP.sources)))
89
90 rysm_bg_lh = IPM_likelihood(rys_MAP.sources, rys_mapped_codes,
→ [findfirst(iszero, rys_mapped_codes[:,o])-1 for o in
→ 1:size(rys_mapped_codes,2)], rysm_lh_matrix,
→ falses(size(rys_mapped_codes,2),length(rys_MAP.sources)))
91
92 println("\\begin{tabular}{l|l|l|l|l}")
93 println("\\hline")
94 println("{\\bf Sequence set} & {\\bf \\textit{rys} naive LH} & {\\bf sib
→ naive LH} & {\\bf LH ratio}\\hline")
95 println("Differential & $(round(rys_bg_lh, sigdigits=5)) &
→ $(round(sib_bg_lh, sigdigits=5)) &
→ $(round(rys_bg_lh-sib_bg_lh, sigdigits=5))\\hline")
96 println("Mapped & $(round(rys_m_bg_lh, sigdigits=5)) & $(round(sibm_bg_lh,
→ sigdigits=5)) & $(round(rys_m_bg_lh-sibm_bg_lh, sigdigits=5))\\hline
→ \\hline")
97 println("\\end{tabular}")

```

---

### 17.8.37 /rys/nested\_sampling/prior\_test.jl

```

1 @info "Setting up for job ... "
2 #JOB FILEPATHS
3 sib_wms_path =
→ "/bench/PhD/NGS_binaries/BMI/sib_nuc_position_sequences.fa_wms.tr"
4 rys_wms_path =
→ "/bench/PhD/NGS_binaries/BMI/rys_nuc_position_sequences.fa_wms.tr"
5
6 sib_df_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_positions"
7 rys_df_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_positions"
8 combined_df_binary =
→ "/bench/PhD/NGS_binaries/BMI/combined_diff_positions"
9
10 sib_code_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_codes"
11 rys_code_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_codes"
12 combined_code_binary = "/bench/PhD/NGS_binaries/BMI/combined_diff_codes"
13
14 sib_diff_bg = "/bench/PhD/NGS_binaries/BMI/sib_diff_bg"
15 rys_diff_bg = "/bench/PhD/NGS_binaries/BMI/rys_diff_bg"
16 combined_diff_bg = "/bench/PhD/NGS_binaries/BMI/combined_diff_bg"

```

```

17
18 sib_e_ui_pth = "/bench/PhD/NGS_binaries/BMI/sib_e_ui"
19 sib_e_inf_pth = "/bench/PhD/NGS_binaries/BMI/sib_e_inf"
20 rys_e_ui_pth = "/bench/PhD/NGS_binaries/BMI/rys_e_ui"
21 rys_e_inf_pth = "/bench/PhD/NGS_binaries/BMI/rys_e_inf"
22 combined_e_ui_pth = "/bench/PhD/NGS_binaries/BMI/combined_e_ui"
23 combined_e_inf_pth = "/bench/PhD/NGS_binaries/BMI/combined_e_inf"
24
25
26 #JOB CONSTANTS
27 const ensemble_size = 1000
28 const no_sources = 8
29 const source_min_bases = 3
30 const source_max_bases = 10
31 @assert source_min_bases < source_max_bases
32 const source_length_range= source_min_bases:source_max_bases
33 const mixing_prior = .07
34 @assert mixing_prior ≥ 0 & mixing_prior ≤ 1
35 const prior_wt=1.2
36
37 @info "Loading master libraries ... "
38 using Distributed, Distributions, Serialization
39
40 @info "Adding librarians and workers ... "
41 no_local_procs=2
42 no_remote_procs=6
43 remote_machine = "10.0.0.119"
44 remote_pool=addprocs([(remote_machine, no_remote_procs)], tunnel=true,
    ↳ topology=:master_worker)
45 local_pool=addprocs(no_local_procs, topology=:master_worker)
46 worker_pool=vcat(remote_pool,local_pool)
47
48 @info "Loading libraries everywhere ... "
49 @everywhere using BioMotifInference, Random
50 Random.seed!(myid()*10000)
51
52 @info "Loading informative source priors ... "
53 sib_wms = BioMotifInference.read_fa_wms_tr(sib_wms_path)
54 rys_wms = BioMotifInference.read_fa_wms_tr(rys_wms_path)
55 sib_mix_prior =
    ↳ BioMotifInference.cluster_mix_prior!(deserialize(sib_df_binary),
    ↳ sib_wms)

```

```

56 rys_mix_prior =
  ↵ BioMotifInference.cluster_mix_prior!(deserialize(rys_df_binary),
  ↵ rys_wms)

57
58 @info "Filtering informative priors ... "
59 sib_prior_wms = BioMotifInference.filter_priors(Int(floor(no_sources/2)),
  ↵ source_max_bases, sib_wms, sib_mix_prior)
60 rys_prior_wms = BioMotifInference.filter_priors(Int(floor(no_sources/2)),
  ↵ source_max_bases, rys_wms, rys_mix_prior)
61 combined_prior_wms =
  ↵ BioMotifInference.combine_filter_priors(Int(floor(no_sources/2)),
  ↵ source_max_bases, (sib_wms, rys_wms), (sib_mix_prior, rys_mix_prior))

62
63 @info "Assembling informative source priors ... "
64 sib_inf_sp = BioMotifInference.assemble_source_priors(no_sources,
  ↵ sib_prior_wms, prior_wt)
65 rys_inf_sp = BioMotifInference.assemble_source_priors(no_sources,
  ↵ rys_prior_wms, prior_wt)
66 combined_inf_sp = BioMotifInference.assemble_source_priors(no_sources,
  ↵ combined_prior_wms, prior_wt)

67
68 @info "Assembling uninformative source priors ... "
69 sib_ui_sp = BioMotifInference.assemble_source_priors(no_sources,
  ↵ Vector{Matrix{Float64}}())
70 rys_ui_sp = BioMotifInference.assemble_source_priors(no_sources,
  ↵ Vector{Matrix{Float64}}())
71 combined_ui_sp = BioMotifInference.assemble_source_priors(no_sources,
  ↵ Vector{Matrix{Float64}}())

72
73 @info "Loading BGHMM likelihood matrix binaries ... "
74 sib_matrix=deserialize(sib_diff_bg)
75 rys_matrix=deserialize(rys_diff_bg)
76 combined_matrix=deserialize(combined_diff_bg)

77
78 @info "Loading coded observation sets ... "
79 sib_obs = deserialize(sib_code_binary)
80 rys_obs = deserialize(rys_code_binary)
81 combined_obs = deserialize(combined_code_binary)

82
83 @info "Assembling sib IPM ensemble on uninformative priors ... "
84 isfile(string(sib_e_ui_pth,'/','ens')) ? (sib_e_ui =
  ↵ deserialize(string(sib_e_ui_pth,'/','ens'))) :

```

```

85     (sib_e_ui = BioMotifInference.IPM_ElusiveInference(worker_pool, sib_e_ui_pth,
86     ↳ ensemble_size, sib_e_ui_sp, (falses(0,0), mixing_prior),
87     ↳ sib_matrix, sib_obs, source_length_range);
88     ↳ serialize(string(sib_e_ui_pth,'/','ens'),sib_e_ui))
89
90 @info "Assembling sib IPM ensemble on informative priors ... "
91 isfile(string(sib_e_inf_pth,'/','ens')) ? (sib_e_inf =
92     ↳ deserialize(string(sib_e_inf_pth,'/','ens'))) :
93     (sib_e_inf = BioMotifInference.IPM_ElusiveInference(worker_pool,
94     ↳ sib_e_inf_pth, ensemble_size, sib_e_inf_sp, (falses(0,0),
95     ↳ mixing_prior), sib_matrix, sib_obs, source_length_range);
96     ↳ serialize(string(sib_e_inf_pth,'/','ens'),sib_e_inf))
97
98 @info "Assembling rys IPM ensemble on uninformative priors ... "
99 isfile(string(rys_e_ui_pth,'/','ens')) ? (rys_e_ui =
100     ↳ deserialize(string(rys_e_ui_pth,'/','ens'))) :
101     (rys_e_ui = BioMotifInference.IPM_ElusiveInference(worker_pool,
102     ↳ rys_e_ui_pth, ensemble_size, rys_e_ui_sp, (falses(0,0),
103     ↳ mixing_prior), rys_matrix, rys_obs, source_length_range);
104     ↳ serialize(string(rys_e_ui_pth,'/','ens'),rys_e_ui))
105
106 @info "Assembling rys IPM ensemble on informative priors ... "
107 isfile(string(rys_e_inf_pth,'/','ens')) ? (rys_e_inf =
108     ↳ deserialize(string(rys_e_inf_pth,'/','ens'))) :
109     (rys_e_inf = BioMotifInference.IPM_ElusiveInference(worker_pool,
110     ↳ rys_e_inf_pth, ensemble_size, rys_e_inf_sp, (falses(0,0),
111     ↳ mixing_prior), rys_matrix, rys_obs, source_length_range);
112     ↳ serialize(string(rys_e_inf_pth,'/','ens'),rys_e_inf))
113
114 @info "Assembling combined IPM ensemble on uninformative priors ... "
115 isfile(string(combined_e_ui_pth,'/','ens')) ? (combined_e_ui =
116     ↳ deserialize(string(combined_e_ui_pth,'/','ens'))) :
117     (combined_e_ui = BioMotifInference.IPM_ElusiveInference(worker_pool,
118     ↳ combined_e_ui_pth, ensemble_size, combined_e_ui_sp, (falses(0,0),
119     ↳ mixing_prior), combined_matrix, combined_obs,
120     ↳ source_length_range);
121     ↳ serialize(string(combined_e_ui_pth,'/','ens'),combined_e_ui))
122
123 @info "Assembling combined IPM ensemble on informative priors ... "
124 isfile(string(combined_e_inf_pth,'/','ens')) ? (combined_e_inf =
125     ↳ deserialize(string(combined_e_inf_pth,'/','ens'))) :

```

```

105     (combined_e_inf = BioMotifInference.IPM_Elusive(worker_pool,
106      ↵ combined_e_inf_pth, ensemble_size, combined_inf_sp,
107      ↵ (falses(0,0), mixing_prior), combined_matrix, combined_obs,
108      ↵ source_length_range);
109      ↵ serialize(string(combined_e_inf_pth,'/',"ens"),combined_e_inf))
110
111 rmprocs(worker_pool)
112
113 @info "Fitting distributions ... "
114
115 sib_ui_dist=fit_mle(Normal,[model.log_Li for model in sib_e_ui.models])
116 sib_inf_dist=fit_mle(Normal,[model.log_Li for model in sib_e_inf.models])
117 @info "Is informative a better prior for sibs?"
118 println(sib_inf_dist.μ > sib_ui_dist.μ && quantile(sib_inf_dist,.5) >
119   ↵ sib_ui_dist.μ)
120
121 rys_ui_dist=fit_mle(Normal,[model.log_Li for model in rys_e_ui.models])
122 rys_inf_dist=fit_mle(Normal,[model.log_Li for model in rys_e_inf.models])
123 @info "Is informative a better prior for rys?"
124 println(rys_inf_dist.μ > rys_ui_dist.μ && quantile(rys_inf_dist,.5) >
125   ↵ rys_ui_dist.μ)
126
127 @info "Job done!"

```

---

### 17.8.38 /rys/nested\_sampling/spike\_recovery.jl

---

```

1 #Testbed for synthetic spike recovery from example background
2 using BioBackgroundModels, BioMotifInference, Random, Distributed,
2   ↵ Distributions, Serialization
3 Random.seed!(786)
4 #CONSTANTS
5 no_obs=500
6 obsl=100:200
7

```

```

8 const folder_path="/bench/PhD/NGS_binaries/BBM/refined_folders"
9
10 report_folders=deserialize(folder_path)
11
12 bhmm_vec=Vector{BHMM}()
13 push!(bhmm_vec,report_folders["intergenic"].partition_report.best_model[2])
14 push!(bhmm_vec,report_folders["periexonic"].partition_report.best_model[2])
15 push!(bhmm_vec,report_folders["exon"].partition_report.best_model[2])
16
17 bhmm_dist=Categorical([.6,.2,.2])
18
19 struc_sig_1=[.1 .7 .1 .1
20             .1 .1 .1 .7
21             .1 .7 .1 .1]
22 struc_sig_2=[.1 .1 .7 .1
23             .1 .1 .7 .1
24             .7 .1 .1 .1]
25 periodicity=8
26 struc_frac_obs=.75
27
28
29 tata_box=[.05 .05 .05 .85
30             .85 .05 .05 .05
31             .05 .05 .05 .85
32             .85 .05 .05 .05
33             .425 .075 .075 .425
34             .85 .05 .05 .05
35             .425 .075 .075 .425]
36 caat_box=[.15 .15 .55 .15
37             .15 .15 .55 .15
38             .05 .85 .05 .05
39             .05 .85 .05 .05
40             .85 .05 .05 .05
41             .85 .05 .05 .05
42             .05 .05 .05 .85
43             .15 .55 .15 .15
44             .15 .15 .15 .55]
45 motif_frac_obs=.7
46 motif_recur_range=1:4
47
48 @info "Constructing synthetic sample set 1..."
49 obs1, bg_scores1, hmm_truth1, spike_truth1 =
    → synthetic_sample(no_obs,obs1,bhmm_vec,bhmm_dist,[struc_sig_1,tata_box],[(true,(st

```

```

50
51 @info "Constructing synthetic sample set 2 ... "
52 obs2, bg_scores2, hmm_truth2, spike_truth2 =
    ↳ synthetic_sample(no_obs,obs1,bhmm_vec,bhmm_dist,[struc_sig_2,caat_box],[(true,(st
53
54 @info "Assembling combined sample set ... "
55 obs3=hcat(obs1,obs2)
56 bg_scores3=hcat(bg_scores1, bg_scores2)
57
58 @info "Assembling worker pool ... "
59
60 #DISTRIBUTED CLUSTERS CONSTANTS
61 remote_machine = "10.0.0.3"
62 no_local_processes = 2
63 no_remote_processes = 6
64
65 @info "Spawning local cluster workers ... "
66 worker_pool=addprocs(no_local_processes, topology=:master_worker)
67 # remote_pool=addprocs([(remote_machine,no_remote_processes)],
    ↳ tunnel=true, topology=:master_worker)
68
69 # worker_pool=vcat(worker_pool, remote_pool)
70
71 @info "Loading worker libraries everywhere ... "
72 @everywhere using BioMotifInference, Random
73 @everywhere Random.seed!(myid())
74
75 e1 = "/bench/PhD/NGS_binaries/BMI/e1"
76 e2 = "/bench/PhD/NGS_binaries/BMI/e2"
77 e3 = "/bench/PhD/NGS_binaries/BMI/e3"
78
79 #JOB CONSTANTS
80 const ensemble_size = 250
81 const no_sources = 2
82 const source_min_bases = 3
83 const source_max_bases = 12
84 const source_length_range= source_min_bases:source_max_bases
85 const mixing_prior = .5
86 const models_to_permute = ensemble_size * 3
87 funcvec=full_perm_funcvec
88 push!(funcvec, BioMotifInference.permute_source)
89 push!(funcvec, BioMotifInference.permute_source)
90 args=[Vector{Tuple{Symbol,Any}}() for i in 1:length(funcvec)]

```

```

91 args[end-1]=[(:weight_shift_freq,0.),(:length_change_freq,1.),(:length_perm_range,1:1
92 args[end]=[(:weight_shift_freq,.1),(:length_change_freq,0.),(:weight_shift_dist,Unifo
93
94
95 instruct = Permute_Instruct(funcvec,
96     ↵ ones(length(funcvec))/length(funcvec),models_to_permute,100,
97     ↵ min_clmps=.02; args=args)
98
99 @info "Assembling source priors ... "
100 prior_array= Vector{Matrix{Float64}}(){}
101 source_priors = BioMotifInference.assemble_source_priors(no_sources,
102     ↵ prior_array)
103
104 @info "Assembling ensemble 1 ... "
105 ifile(e1*/ens) ? (ens1 = deserialize(e1*/ens)) :
106     (ens1 = IPM_Ensemble(worker_pool, e1, ensemble_size, source_priors,
107         ↵ (falses(0,0), mixing_prior), bg_scores1, obs1,
108         ↵ source_length_range))
109
110 @info "Converging ensemble 1 ... "
111 logZ1 = BioMotifInference.converge_ensemble!(ens1, instruct, worker_pool,
112     ↵ backup=true,250, wk_disp=false, tuning_disp=true, ens_disp=false,
113     ↵ conv_plot=true, src_disp=true, lh_disp=false, liwi_disp=false)
114
115 @info "Assembling ensemble 2 ... "
116 ifile(e2*/ens) ? (ens2 = deserialize(e2*/ens)) :
117     (ens2 = IPM_Ensemble(worker_pool, e2, ensemble_size, source_priors,
118         ↵ (falses(0,0), mixing_prior), bg_scores2, obs2,
119         ↵ source_length_range))
120
121 @info "Converging ensemble 2 ... "
122 logZ2 = BioMotifInference.converge_ensemble!(ens2, instruct, worker_pool,
123     ↵ backup=true,250, wk_disp=false, tuning_disp=true, ens_disp=false,
124     ↵ conv_plot=true, src_disp=true, lh_disp=false, liwi_disp=false)
125
126 @info "Assembling ensemble 3 ... "
127 ifile(e3*/ens) ? (ens3 = deserialize(e3*/ens)) :
128     (ens3 = IPM_Ensemble(worker_pool, e3, ensemble_size, source_priors,
129         ↵ (falses(0,0), mixing_prior), bg_scores3, obs3,
130         ↵ source_length_range))
131
132 @info "Converging ensemble 3 ... "
133
```

```

121 logZ3 = BioMotifInference.converge_ensemble!(ens3, instruct, worker_pool,
    ↵ backup=true,250), wk_disp=false, tuning_disp=true, ens_disp=false,
    ↵ conv_plot=true, src_disp=true, lh_disp=false, liwi_disp=false)
122
123 @info "logZR of joint 1&2 / 3:"
124 ratio=logZ1+logZ2-logZ3
125 println(ratio)
126
127 rmprocs(worker_pool)

```

---

### 17.8.39 /rys/position\_analysis/position\_overlap.jl

```

1 using DataFrames, BioSequences, BioBackgroundModels, CSV, FASTX,
    ↵ Statistics
2
3 function add_position_sequences!(df::DataFrame, genome_path::String,
    ↵ genome_idx_path::String)
4     scaffold_seq_record_dict::Dict{String,BioSequences.LongSequence} =
    ↵ BioBackgroundModels.build_scaffold_seq_dict(genome_path,
    ↵ genome_idx_path)
5
6     seqs=[BioSequences.LongSequence{DNAAlphabet{4}}() for i in
    ↵ 1:size(df,1)]
7     df[!, :seq] *= seqs
8
9     Threads.@threads for entry in eachrow(df)
10         entry.start > 0 ? (estart=entry.start) : (estart=1)
11         entry.seq=BioBackgroundModels.fetch_sequence(entry.chr,
    ↵ scaffold_seq_record_dict ,estart, entry.end, '+')
12     end
13
14 end
15
16 function get_cluster!(df::DataFrame, fasta::String, arch::String)
17     df[!, :cluster] *= zeros(Int64, size(df,1))
18     art=CSV.read(arch, header=0)
19
20     position_vec=Vector{Vector{Int64}}()
21     reader=FASTA.Reader(open(fasta))
22     for (n, entry) in enumerate(reader)
23         scaffold = FASTA.identifier(entry)
24         desc_array = split(FASTA.description(entry))

```

```

25     pos_start = parse(Int64, desc_array[2])
26     pos_end = parse(Int64, desc_array[4])
27     seq = FASTA.sequence(entry)
28
29     idx = filter(in(findall(chr→chr=scaffold, df.chr)),
30                  ↳ findall(start→start=pos_start, df.start))
31
32     if length(idx)==1
33         match=df[idx[1],:]
34         @assert match.end == pos_end
35         @assert match.seq==seq
36         match.cluster=art.Column1[n]
37     end
38 end
39
40 function make_position_df(position_fasta::String)
41     position_reader = FASTA.Reader(open((position_fasta), "r"))
42     position_df = DataFrame(SeqID = String[], Start=Int64[], End=Int64[],
43                             ↳ Seq = LongSequence[])
44
45     for entry in position_reader
46         scaffold = FASTA.identifier(entry)
47
48         if scaffold ≠ "MT"
49             desc_array = split(FASTA.description(entry))
50             pos_start = parse(Int64, desc_array[2])
51             pos_end = parse(Int64, desc_array[4])
52             seq = FASTA.sequence(entry)
53
54             if !hasambiguity(seq)
55                 push!(position_df, [scaffold, pos_start, pos_end, seq])
56             end
57         end
58     end
59
60     close(position_reader)
61     return position_df
62 end
63
64 function observation_setup(position_df :: DataFrame; order :: Int64=0,
65                           ↳ symbol :: Symbol=:Seq)

```

```

64     order_seqs =
65       ↳ BioBackgroundModels.get_order_n_seqs(Vector{LongSequence{DNAAlphabet{2}}})(pos
66       ↳ symbol],order)
67   coded_seqs = BioBackgroundModels.code_seqs(order_seqs)
68
69
70   return coded_seqs
71 end
72
73
74
75
76
77
78   function map_positions!(base_df, map_df)
79     df_rows = size(base_df,1)
80     base_df[!, :mapped_pos] = zeros(Int64, df_rows)
81     base_df[!, :overlap_bp] = zeros(Int64, df_rows)
82     base_df[!, :rel_overlap] = zeros(df_rows)
83
84
85     chrgroups=groupby(base_df, :chr)
86
87
88     for chr_df in chrgroups #split the df by scaffold so base numbers are
89       ↳ local to scaffold
90       base_chr = chr_df.chr[1]
91       matched=false
92       for map_chr_df in groupby(map_df, :chr)
93         base_chr == map_chr_df.chr[1] && subDFmap!(chr_df,
94           ↳ map_chr_df)
95         matched=true
96       end
97       matched=false &&
98         ↳ (chr_df.mapped_pos=zeros(Int64,length(chr_df.start));
99             ↳ chr_df.overlap_bp=zeros(Int64,length(chr_df.start));
100            ↳ chr_df.rel_overlap=zeros(length(chr_df.start)))
101
102
103     return base_df
104   end
105
106
107   function subDFmap!(chr_df::SubDataFrame,
108     ↳ map_chr_df::SubDataFrame)
109     Threads.@threads for position in eachrow(chr_df)
110       search_start = position.start;
111       ↳ search_end=position.end
112
113       ↳ start_idxs=findall(in(findall(ende→search_start≤ende,ma
114       ↳ findall(start→search_start≥start,map_chr_df.start)))

```

```

97
    ↵ end_idxs=findall(in(findall(ende→search_end≤ende,map_ch
    ↵ findall(start→search_end≥start,map_chr_df.start))
98     mapped_idxs = unique(vcat(start_idxs, end_idxs))

99
100    overlaps=Vector{Int64}()
101    for idx in mapped_idxs
102        #println(idx)
103        mapped_start=map_chr_df.start[idx];
104        ↵ mapped_end=map_chr_df.end[idx]
105        if (mapped_start==search_start &&
106            ↵ mapped_end==search_end) #complete
107            ↵ position overlap (all positions are the
108            ↵ same size)
109
110            ↵ push!(overlaps,(mapped_end-mapped_start+1))
111            #println("same")
112        elseif (mapped_start < search_start ≤
113            ↵ mapped_end) #5' end of the searched
114            ↵ position overlaps with the mapped
115            ↵ position
116
117            ↵ push!(overlaps,(mapped_end-search_start+1))
118            #println("5' overlap")
119        elseif (mapped_start ≤ search_end <
120            ↵ mapped_end) #3' end of searched position
121            ↵ overlaps with the mapped position
122
123            ↵ push!(overlaps,(search_end-mapped_start+1))
124            #println("3' overlap")
125        end
126    end

127    if length(mapped_idxs)>0 #1 or more mapped
128        ↵ candidates: take the one with the most
129        ↵ overlap
130        overlap, oidx = findmax(overlaps)
131        map_idx = mapped_idxs[oidx]
132        position.mapped_pos=map_idx
133        position.overlap_bp=overlap
134        position.rel_overlap=overlap /
135        ↵ (search_end-search_start+1)
136
137    end

```

```

123           end
124       end

```

---

#### 17.8.40 /test/eggbox\_plots.jl

```

1 using GMC_NS, Distributions, Plots
2
3 gr()
4 default(legendfont = (10), guidefont = (12), tickfont = (10))
5
6 prior=Uniform(0,1)
7 priors=[prior,prior]
8
9 box=[-1. 1.
10      -1. 1.]
11
12 gmc=copy(GMC_DEFAULTS)
13 gmc[2]=1e-3
14
15 e=Eggbox_Ensemble("Eggbox_test", 400, priors, box, gmc ... )
16
17 x1=0.:01:1.
18 x2=0.:01:1.
19
20 f(x1, x2) = begin
21     (2 + cos(5π * x1) * cos(5π * x2))^5
22 end
23
24 function harvest_xs(e)
25     x1m=[m.pos[1] for m in e.models]
26     x2m=[m.pos[2] for m in e.models]
27     x1m.=+1.; x2m.=+1.
28     x1m./=2.; x2m./=2
29     return x1m, x2m
30 end
31
32 plt=contour(x1,x2,f, xlabel="x1", ylabel="x2", colorbar=:none)
33
34 init=deepcopy(plt)
35 plot!(init, title="Initialisation")
36 scatter!(init, harvest_xs(e), marker=:cross, markercolor=:black,
→   legend=:none)

```

```
37
38 converge_ensemble!(e,backup=(true,100), max_iterates=1000,
  ↵  converge_factor=1e-6)
39
40 thousand=deepcopy(plt)
41 plot!(thousand, title="1000 iterates")
42 scatter!(thousand, harvest_xs(e), marker=:cross, markercolor=:black,
  ↵  legend=:none)
43
44 converge_ensemble!(e,backup=(true,100), max_iterates=5000,
  ↵  converge_factor=1e-6)
45
46 tenthousand=deepcopy(plt)
47 plot!(tenthousand, title="5000 iterates")
48 scatter!(tenthousand, harvest_xs(e), marker=:cross, markercolor=:black,
  ↵  legend=:none)
49
50 converge_ensemble!(e,backup=(true,100), converge_factor=1e-6)
51
52 converged=deepcopy(plt)
53 plot!(converged, title="Converged ($(length(e.log_Li)) iterates)")
54 scatter!(converged, harvest_xs(e), marker=:cross, markercolor=:black,
  ↵  legend=:none)
55
56 cmb=plot(init, thousand, tenthousand, converged, layout=grid(2,2),
  ↵  size=(900,900))
57 savefig(cmb, "/bench/PhD/Thesis/images/GMC/eggbox.png")
```

---

# Bibliography

- [ABS<sup>+</sup>10] W. Ted Allison, Linda K. Barthel, Kristina M. Skebo, Masaki Takechi, Shoji Kawamura, and Pamela A. Raymond. Ontogeny of cone photoreceptor mosaics in zebrafish. *The Journal of Comparative Neurology*, 518(20):4182–4195, October 2010.
- [AC96] Macrene R. Alexiades and Constance Cepko. Quantitative analysis of proliferation and cell cycle length during development of the rat retina. *Developmental Dynamics*, 205(3):293–307, March 1996.
- [ADH10] Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle Markov chain Monte Carlo methods: Particle Markov Chain Monte Carlo Methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, June 2010.
- [AH09] Michalis Agathocleous and William A. Harris. From progenitors to differentiated cells in the vertebrate retina. *Annual Review of Cell and Developmental Biology*, 25:45–69, 2009.
- [AHW<sup>+</sup>20] Afnan Azizi, Anne Herrmann, Yinan Wan, Salvador JRP Buse, Philipp J Keller, Raymond E Goldstein, and William A Harris. Nuclear crowding and nonlinear diffusion during interkinetic nuclear migration in the zebrafish retina. 9(58635):31, 2020.
- [ALHP07] Michalis Agathocleous, Morgane Locker, William A. Harris, and Muriel Perron. A General Role of Hedgehog in the Regulation of Proliferation. *Cell Cycle*, 6(2):156–159, January 2007.
- [AlQ19] Mohammed AlQuraishi. AlphaFold at CASP13. *Bioinformatics*, 35(22):4862–4865, November 2019.
- [And18] Simon Andrews. FastQC A Quality Control tool for High Throughput Sequence Data. <https://www.bioinformatics.babraham.ac.uk/projects/fastqc/>, 04-10-18.
- [And18] Simon Andrews. Trim Galore! [https://www.bioinformatics.babraham.ac.uk/projects/trim\\_galore/](https://www.bioinformatics.babraham.ac.uk/projects/trim_galore/), 28-06-18.
- [AR08] Ruben Adler and Pamela A. Raymond. Have we achieved a unified model of photoreceptor cell fate specification in vertebrates? *Brain Research*, 1192:134–150, February 2008.
- [ASNS17] Ankit Agrawal, Snehal V Sambare, Leelavati Narlikar, and Rahul Siddharthan. THiCweed: Fast, sensitive detection of sequence features by clustering big data sets. *bioRxiv*, November 2017.

- [BA02] Kenneth P. Burnham and David Raymond Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer, New York, 2nd ed edition, 2002.
- [Bar14] Marcello Barbieri. *Code Biology: A New Science of Life*. Springer Berlin Heidelberg, New York, NY, 2014.
- [BB01] Thomas Becker and Catherina G. Becker. Regenerating descending axons preferentially reroute to the gray matter in the presence of a general macrophage/microglial reaction caudal to a spinal transection in adult zebrafish. *The Journal of Comparative Neurology*, 433(1):131–147, April 2001.
- [BB20] M.J. Brewer and A. Butler. Model selection and the cult of AIC. In *Departmental Seminar*, University of Wollongong, Australia, February 2020.
- [BEKS15] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *arXiv:1411.1607 [cs]*, July 2015.
- [BGL<sup>+</sup>10] Klaus A. Becker, Prachi N. Ghule, Jane B. Lian, Janet L. Stein, Andre J. van Wijnen, and Gary S. Stein. Cyclin D2 and the CDK substrate p220<sup>NPAT</sup> are required for self-renewal of human embryonic stem cells. *Journal of Cellular Physiology*, 222(2):456–464, February 2010.
- [Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, 2006.
- [BJ79] D. H. Beach and Marcus Jacobson. Influences of thyroxine on cell proliferation in the retina of the clawed frog at different ages. *The Journal of Comparative Neurology*, 183(3):615–623, February 1979.
- [BNL<sup>+</sup>96] M. Burmeister, J. Novak, M. Y. Liang, S. Basu, L. Ploder, N. L. Hawes, D. Vidgen, F. Hoover, D. Goldman, V. I. Kalnins, T. H. Roderick, B. A. Taylor, M. H. Hankin, and R. R. McInnes. Ocular retardation mouse caused by Chx10 homeobox null allele: Impaired retinal progenitor proliferation and bipolar cell differentiation. *Nature Genetics*, 12(4):376–384, April 1996.
- [BP66] Leonard E. Baum and Ted Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, December 1966.
- [BRD<sup>+</sup>15] Henrik Boije, Steffen Rulands, Stefanie Dudczig, Benjamin D. Simons, and William A. Harris. The Independent Probabilistic Firing of Transcription Factors: A Paradigm for Clonal Variability in the Zebrafish Retina. *Developmental Cell*, 34(5):532–543, September 2015.
- [Bri10] Ingo Brigandt. Beyond reduction and pluralism: Toward an epistemology of explanatory integration in biology. *Erkenntnis*, 73(3):295–311, 2010.
- [BSB00] José M. Bernardo, Adrian F. M. Smith, and Thomas Bayes. *Bayesian Theory*. Wiley Series in Probability and Mathematical Statistics. Wiley, Chichester, 2000.

- [Buc16] Johannes Buchner. A statistical test for Nested Sampling algorithms. *Statistics and Computing*, 26(1-2):383–392, January 2016.
- [CAH<sup>+</sup>14] L. Centanin, J.-J. Ander, B. Hoeckendorf, K. Lust, T. Kellner, I. Kraemer, C. Urbany, E. Hasel, W. A. Harris, B. D. Simons, and J. Wittbrodt. Exclusive multipotency and preferential asymmetric divisions in post-embryonic neural stem cells of the fish retina. *Development*, 141(18):3472–3482, September 2014.
- [CAI<sup>+</sup>04] Brenda LK Coles, Brigitte Angénieux, Tomoyuki Inoue, Katia Del Rio-Tsonis, Jason R. Spence, Roderick R. McInnes, Yvan Arsenijevic, and Derek van der Kooy. Facile isolation and the characterization of human retinal stem cells. *Proceedings of the National Academy of Sciences of the United States of America*, 101(44):15772–15777, 2004.
- [CAR<sup>+</sup>15] Renee W. Chow, Alexandra D. Almeida, Owen Randlett, Caren Norden, and William A. Harris. Inhibitory neuron migration and IPL formation in the developing zebrafish retina. *Development*, 142(15):2665–2677, August 2015.
- [Cav18] Florencia Cavodeassi. Dynamic Tissue Rearrangements during Vertebrate Eye Morphogenesis: Insights from Fish Models. *Journal of Developmental Biology*, 6(1):4, February 2018.
- [CAY<sup>+</sup>96] Constance L. Cepko, Christopher P. Austin, Xianjie Yang, Macrene Alexiades, and Diala Ezzeddine. Cell fate determination in the vertebrate retina. *Proceedings of the National Academy of Sciences*, 93(2):589–595, 1996.
- [CBR03] Michel Cayouette, Ben A. Barres, and Martin Raff. Importance of intrinsic mechanisms in cell fate decisions in the developing rat retina. *Neuron*, 40(5):897–904, December 2003.
- [CC07] Bo Chen and Constance L Cepko. Requirement of histone deacetylase activity for the expression of critical photoreceptor genes. *BMC Developmental Biology*, 7(1):78, 2007.
- [CCP09] J. M. Catchen, J. S. Conery, and J. H. Postlethwait. Automated identification of conserved synteny after whole-genome duplication. *Genome Research*, 19(8):1497–1505, August 2009.
- [CCY<sup>+</sup>05] Florencia Cavodeassi, Filipa Carreira-Barbosa, Rodrigo M. Young, Miguel L. Concha, Miguel L. Allende, Corinne Houart, Masazumi Tada, and Stephen W. Wilson. Early Stages of Zebrafish Eye Formation Require the Coordinated Activity of Wnt11, Fz5, and the Wnt/β-Catenin Pathway. *Neuron*, 47(1):43–56, July 2005.
- [CDMR01] Ian J. Conlon, Graham A. Dunn, Anne W. Mudge, and Martin C. Raff. Extracellular control of cell size. *Nature Cell Biology*, 3(10):918–921, October 2001.
- [Cep14] Connie Cepko. Intrinsically different retinal progenitor cells produce specific types of progeny. *Nature Reviews Neuroscience*, 15(9):615–627, September 2014.
- [CHB<sup>+</sup>08] Hannah H. Chang, Martin Hemberg, Mauricio Barahona, Donald E. Ingber, and Sui Huang. Transcriptome-wide noise controls lineage choice in mammalian progenitor cells. *Nature*, 453(7194):544–547, May 2008.

- [Chi97] D. C. Chin. Comparative study of stochastic algorithms for system optimization based on gradient approximations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):244–249, April 1997.
- [CHW11] Lázaro Centanin, Burkhard Hoeckendorf, and Joachim Wittbrodt. Fate Restriction and Multipotency in Retinal Stem Cells. *Cell Stem Cell*, 9(6):553–562, December 2011.
- [CM18] Peter R Cook and Davide Marenduzzo. Transcription-driven genome organization: A model for chromosome structure and the regulation of gene expression tested through simulations. *Nucleic Acids Research*, 1:12, 2018.
- [CMD<sup>+</sup>09] Fatemah Chehrehasa, Adrian C.B. Meedeniya, Patrick Dwyer, Greger Abrahamsen, and Alan Mackay-Sim. EdU, a new thymidine analogue for labelling proliferating cells in the nervous system. *Journal of Neuroscience Methods*, 177(1):122–130, February 2009.
- [CW08] Alexander Churbanov and Stephen Winters-Hilt. Implementing EM and Viterbi algorithms for Hidden Markov Model in linear memory. *BMC Bioinformatics*, 9(1):224, December 2008.
- [CXP<sup>+</sup>13] K. Chen, Y. Xi, X. Pan, Z. Li, K. Kaestner, J. Tyler, S. Dent, X. He, and W. Li. DANPOS: Dynamic analysis of nucleosome position and occupancy by sequencing. *Genome Research*, 23(2):341–351, February 2013.
- [CYV<sup>+</sup>08] Anna M. Clark, Sanghee Yun, Eric S. Veien, Yuan Y. Wu, Robert L. Chow, Richard I. Dorsky, and Edward M. Levine. Negative regulation of Vsx1 by its paralog Chx10/Vsx2 is conserved in the vertebrate retina. *Brain Research*, 1192:99–113, February 2008.
- [CZ12] Kairong Cui and Keji Zhao. Genome-Wide Approaches to Determining Nucleosome Occupancy in Metazoans Using MNase-Seq. In Randall H. Morse, editor, *Chromatin Remodeling*, volume 833, pages 413–419. Humana Press, Totowa, NJ, 2012.
- [Dal15] Stephen Dalton. Linking the Cell Cycle to Cell Fate Decisions. *Trends in Cell Biology*, 25(10):592–600, October 2015.
- [Dar88] Charles Darwin. *The Origin of Species by Means of Natural Selection, or, The Preservation of Favoured Races in the Struggle for Life*. J. Murray, London :, 1888.
- [DB16] Christian Dietz and Michael R. Berthold. KNIME for Open-Source Bioimage Analysis: A Tutorial. In Winnok H. De Vos, Sebastian Munck, and Jean-Pierre Timmermans, editors, *Focus on Bio-Image Informatics*, volume 219, pages 179–197. Springer International Publishing, Cham, 2016.
- [DC00] Michael A. Dyer and Constance L. Cepko. Control of Müller glial cell proliferation and activation following retinal injury. *Nature Neuroscience*, 3(9):873–880, September 2000.
- [DCRH97] R. I. Dorsky, W. S. Chang, D. H. Rapaport, and W. A. Harris. Regulation of neuronal diversity in the Xenopus retina by Delta signalling. *Nature*, 385(6611):67–70, January 1997.
- [DD17] Jacqueline Deschamps and Denis Duboule. Embryonic timing, axial stem cells, chromatin dynamics, and the Hox clock. *Genes & Development*, 31(14):1406–1416, July 2017.

- [DFWV<sup>+</sup>97] MARCO Di Frusco, Hans Weiher, Barbara C. Vanderhyden, Takashi Imai, Tadahiro Shiomii, T. A. Hori, Rudolf Jaenisch, and Douglas A. Gray. Proviral inactivation of the Npat gene of Mpv 20 mice results in early embryonic arrest. *Molecular and cellular biology*, 17(7):4080–4086, 1997.
- [DH05] T. A. Down and Tim J.P. Hubbard. NestedMICA: Sensitive inference of over-represented motifs in nucleic acid sequence. *Nucleic Acids Research*, 33(5):1445–1453, March 2005.
- [DHM07] Persi Diaconis, Susan Holmes, and Richard Montgomery. Dynamical Bias in the Coin Toss. *SIAM Review*, 49(2):211–235, January 2007.
- [DPCH03] Tilak Das, Bernhard Payer, Michel Cayouette, and William A. Harris. In vivo time-lapse imaging of cell divisions during neurogenesis in the developing zebrafish retina. *Neuron*, 37(4):597–609, 2003.
- [DRH95] Richard I Dorsky, David H Rapaport, and William A Harris. Xotch inhibits cell differentiation in the xenopus retina. *Neuron*, 14(3):487–496, March 1995.
- [Eag18] Antony Eagle. Chance versus Randomness. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2018 edition, 2018.
- [EHML09] Ted Erclik, Volker Hartenstein, Roderick R. McInnes, and Howard D. Lipshitz. Eye evolution at high resolution: The neuron as a unit of homology. *Developmental Biology*, 332(1):70–79, August 2009.
- [ESY<sup>+</sup>17] Peter Engerer, Sachihiro C Suzuki, Takeshi Yoshimatsu, Prisca Chapouton, Nancy Obeng, Benjamin Odermatt, Philip R Williams, Thomas Misgeld, and Leanne Godinho. Uncoupling of neurogenesis and differentiation during retinal development. *The EMBO Journal*, 36(9):1134–1146, May 2017.
- [Fag12] Melinda Bonnie Fagan. The Joint Account of Mechanistic Explanation. *Philosophy of Science*, 79(4):448–472, October 2012.
- [Fag13] Melinda Bonnie Fagan. *Philosophy of Stem Cell Biology: Knowledge in Flesh and Blood*. New Directions in the Philosophy of Science. Palgrave Macmillan, Hounds mills, Basingstoke, Hampshire, 2013.
- [Fag15] Melinda Bonnie Fagan. Collaborative explanation and biological mechanisms. *Studies in History and Philosophy of Science Part A*, 52:67–78, August 2015.
- [Fav15] Donald Favareau. Why this now? The conceptual and historical rationale behind the development of biosemiotics. *Green Letters*, 19(3):227–242, September 2015.
- [FEF<sup>+</sup>09] Curtis R. French, Timothy Erickson, Danielle V. French, David B. Pilgrim, and Andrew J. Waskiewicz. Gdf6a is required for the initiation of dorsal–ventral retinal patterning and lens development. *Developmental Biology*, 333(1):37–47, September 2009.
- [Fey75] Paul Feyerabend. How to Defend Society Against Science. *Radical Philosophy*, (11):3–8, 1975.

- [Fey81] Paul Feyerabend. *Philosophical Papers*. Cambridge University Press, Cambridge ; New York, 1981.
- [Fey93] Paul Feyerabend. *Against Method*. Verso, London ; New York, 3rd ed edition, 1993.
- [FH08] Farhan Feroz and M. P. Hobson. Multimodal nested sampling: An efficient and robust alternative to MCMC methods for astronomical data analysis. *Monthly Notices of the Royal Astronomical Society*, 384(2):449–463, January 2008.
- [FHB09] F. Feroz, M. P. Hobson, and M. Bridges. MultiNest: An efficient and robust Bayesian inference tool for cosmology and particle physics. *Monthly Notices of the Royal Astronomical Society*, 398(4):1601–1614, October 2009.
- [FK12] Chikara Furusawa and Kunihiko Kaneko. A Dynamical-Systems View of Stem Cell Biology. *Science*, 338(6104):215–217, October 2012.
- [FR00] Andy J. Fischer and Thomas A. Reh. Identification of a Proliferating Marginal Zone of Retinal Progenitors in Postnatal Chickens. *Developmental Biology*, 220(2):197–210, April 2000.
- [FR03] Andy J. Fischer and Thomas A. Reh. Potential of Müller glia to become neurogenic retinal progenitor cells: Retinal Müller Glia as a Source of Stem Cells. *Glia*, 43(1):70–76, July 2003.
- [GBB<sup>+</sup>03] G. Gao, A. P. Bracken, K. Burkard, D. Pasini, M. Classon, C. Attwooll, M. Sagara, T. Imai, K. Helin, and J. Zhao. NPAT Expression Is Regulated by E2F and Is Essential for Cell Cycle Progression. *Molecular and Cellular Biology*, 23(8):2821–2833, April 2003.
- [GDL<sup>+</sup>09] Prachi N. Ghule, Zbigniew Dominski, Jane B. Lian, Janet L. Stein, Andre J. van Wijnen, and Gary S. Stein. The subnuclear organization of histone gene regulatory proteins and 3' end processing factors of normal somatic and embryonic stem cells is compromised in selected human cancer cell types. *Journal of Cellular Physiology*, 220(1):129–135, July 2009.
- [Geh96] Walter J. Gehring. The master control gene for morphogenesis and evolution of the eye. *Genes to Cells*, 1(1):11–15, January 1996.
- [GJH<sup>+</sup>17] Mahdi Golkaram, Jiwon Jang, Stefan Hellander, Kenneth S. Kosik, and Linda R. Petzold. The Role of Chromatin Density in Cell Population Heterogeneity during Stem Cell Differentiation. *Scientific Reports*, 7(1):13307, October 2017.
- [GNB08] Nathan J. Gosse, Linda M. Nevin, and Herwig Baier. Retinotopic order in the absence of axon competition. *Nature*, 452(7189):892–895, April 2008.
- [Gol17] Sheldon Goldstein. Bohmian Mechanics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2017 edition, 2017.
- [Gre17] Michael Green. The truth about Bayesian priors and overfitting. <https://towardsdatascience.com/the-truth-about-bayesian-priors-and-overfitting-84e24d3a1153>, August 2017.

- [GS07] Frédéric Gaillard and Yves Sauvé. Cell-based therapy for retina degeneration: The promise of a cure. *Vision Research*, 47(22):2815–2824, October 2007.
- [GTR<sup>+</sup>10] D. M. Gilbert, S.-I. Takebayashi, T. Ryba, J. Lu, B. D. Pope, K. A. Wilson, and I. Hiratani. Space and Time in the Nucleus: Developmental Control of Replication Timing and Chromosome Architecture. *Cold Spring Harbor Symposia on Quantitative Biology*, 75(0):143–153, January 2010.
- [GWC<sup>+</sup>07] Leanne Godinho, Philip R. Williams, Yvonne Claassen, Elayne Provost, Steven D. Leach, Maarten Kamermans, and Rachel O.L. Wong. Nonapical Symmetric Divisions Underlie Horizontal Cell Layer Formation in the Developing Retina In Vivo. *Neuron*, 56(4):597–603, November 2007.
- [GZC<sup>+</sup>11] Francisco L. A. F. Gomes, Gen Zhang, Felix Carbonell, José A. Correa, William A. Harris, Benjamin D. Simons, and Michel Cayouette. Reconstruction of rat retinal progenitor cell lineages in vitro reveals a surprising degree of stochasticity in cell fate decisions. *Development (Cambridge, England)*, 138(2):227–235, January 2011.
- [Has03] Samiul Hasan. *Abstract.Pdf*. PhD thesis, University of Cambridge, Cambridge, 2003.
- [HBEH88] Christine E. Holt, Thomas W. Bertsch, Hillary M. Ellis, and William A. Harris. Cellular Determination in the Xenopus Retina is Independent of Lineage and Birth Date. *Neuron*, 1(1):15–26, March 1988.
- [HCG95] G Halder, P Callaerts, and W. Gehring. Induction of ectopic eyes by targeted expression of the eyeless gene in *Drosophila*. *Science*, 267(5205):1788–1792, March 1995.
- [HE99] Minjie Hu and Stephen S. Easter. Retinal Neurogenesis: The Formation of the Initial Central Patch of Postmitotic Cells. *Developmental Biology*, 207(2):309–321, March 1999.
- [Hea67] D.F. Heath. Normal or Log-normal: Appropriate Distributions. *Nature*, 213:1159–1160, March 1967.
- [HG12] David I. Hastie and Peter J. Green. Model choice using reversible jump Markov chain Monte Carlo: *Model choice using reversible jump MCMC*. *Statistica Neerlandica*, 66(3):309–338, August 2012.
- [HH91] William A. Harris and Volker Hartenstein. Neuronal determination without cell division in *xenopus* embryos. *Neuron*, 6:499–515, 1991.
- [HHHL18] Edward Higson, Will Handley, Mike Hobson, and Anthony Lasenby. Sampling Errors in Nested Sampling Parameter Estimation. *Bayesian Analysis*, 13(3):873–896, September 2018.
- [HHHL19] Edward Higson, Will Handley, Michael Hobson, and Anthony Lasenby. Dynamic nested sampling: An improved algorithm for parameter estimation and evidence calculation. *Statistics and Computing*, 29(5):891–913, September 2019.

- [HJH<sup>+</sup>17] Maximilian Hastreiter, Tim Jeske, Jonathan Hoser, Michael Kluge, Kaarin Ahomaa, Marie-Sophie Friedl, Sebastian J. Kopetzky, Jan-Dominik Quell, H. Werner Mewes, and Robert Küffner. KNIME4NGS: A comprehensive toolbox for Next Generation Sequencing analysis. *Bioinformatics*, page btx003, January 2017.
- [HLZQ17] Sui Huang, Fangting Li, Joseph X. Zhou, and Hong Qian. Processes on the emergent landscapes of biochemical reaction networks and heterogeneous cell population dynamics: Differentiation in living matters. *Journal of the Royal Society Interface*, 14(130), May 2017.
- [Hof08] Jesper Hoffmeyer, editor. *A Legacy for Living Systems: Gregory Bateson as Precursor to Biosemiotics*. Number volume 2 in Biosemiotics. Springer, New York, 2008.
- [Hof15] Jesper Hoffmeyer. Semiotic Scaffolding of Multicellularity. *Biosemiotics*, 8(2):159–171, August 2015.
- [Hor04] D. J. Horsford. Chx10 repression of Mitf is required for the maintenance of mammalian neuroretinal identity. *Development*, 132(1):177–187, December 2004.
- [HP98] William A. Harris and Muriel Perron. Molecular recapitulation: The growth of the vertebrate retina. *International Journal of Developmental Biology*, 42:299–304, 1998.
- [HPG07] J. Herisson, G. Payen, and R. Gherbi. A 3D pattern matching algorithm for DNA sequences. *Bioinformatics*, 23(6):680–686, March 2007.
- [HSF97] John Henderson, Steven Salzberg, and Kenneth H. Fasman. Finding Genes in DNA with a Hidden Markov Model. *Journal of Computational Biology*, 4(2):127–141, January 1997.
- [HSK<sup>+</sup>13] Steven A. Harvey, Ian Sealy, Ross Kettleborough, Fruzsina Fenyes, Richard White, Derek Stemple, and James C. Smith. Identification of the zebrafish maternal and paternal transcriptomes. *Development*, 140(13):2703–2710, July 2013.
- [HW81] Philip Heidelberger and Peter D. Welch. A spectral method for confidence interval generation and run length control in simulations. *Communications of the ACM*, 24(4):233–245, April 1981.
- [HYSL11] Hongpeng He, Fa-Xing Yu, Chi Sun, and Yan Luo. CBP/p300 and SIRT1 Are Involved in Transcriptional Regulation of S-Phase Specific Histone Genes. *PLoS ONE*, 6(7):e22088, July 2011.
- [HZA<sup>+</sup>12] Jie He, Gen Zhang, Alexandra D. Almeida, Michel Cayouette, Benjamin D. Simons, and William A. Harris. How Variable Clones Build an Invariant Retina. *Neuron*, 75(5):786–798, September 2012.
- [HZP18] Lachlan Harris, Oressia Zalucki, and Michael Piper. BrdU/EdU dual labeling to determine the cell-cycle dynamics of defined cellular subpopulations. *Journal of Molecular Histology*, 49(3):229–234, June 2018.
- [ICW13] Kenzo Ivanovitch, Florencia Cavodeassi, and Stephen W. Wilson. Precocious Acquisition of Neuroepithelial Character in the Eye Field Underlies the Onset of Eye Morphogenesis. *Developmental Cell*, 27(3):293–305, November 2013.

- [IKRN16] Jaroslav Icha, Christiane Kunath, Mauricio Rocha-Martins, and Caren Norden. Independent modes of ganglion cell translocation ensure correct lamination of the zebrafish retina. *The Journal of Cell Biology*, 215(2):259–275, October 2016.
- [ILU<sup>+</sup>14] Helgi I. Ingólfsson, Cesar A. Lopez, Jaakkko J. Uusitalo, Djurre H. de Jong, Srinivasa M. Gopal, Xavier Periole, and Siewert J. Marrink. The power of coarse graining in biomolecular simulations: The power of coarse graining in biomolecular simulations. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 4(3):225–248, May 2014.
- [Ioa05] John P. A. Ioannidis. Why Most Published Research Findings Are False. *PLoS Medicine*, 2(8):e124, August 2005.
- [IYS<sup>+</sup>96] T Imai, M Yamauchi, N Seki, T Sugawara, T Saito, Y Matsuda, H Ito, T Nagase, N Nomura, and T Hori. Identification and characterization of a new gene physically linked to the ATM gene. *Genome Research*, 6(5):439–447, May 1996.
- [JBE03] Edwin T Jaynes, G. Larry Bretthorst, and EBSCO Publishing. *Probability Theory: The Logic of Science*. Cambridge University Press, Cambridge, 2003.
- [JHTS00] Jong-Hwan Lee, Ho-Young Jung, Te-Won Lee, and Soo-Young Lee. Speech feature extraction using independent component analysis. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, volume 3, pages 1631–1634, Istanbul, Turkey, 2000. IEEE.
- [JO04] Jerald B. Johnson and Kristian S. Omland. Model selection in ecology and evolution. *Trends in Ecology & Evolution*, 19(2):101–108, February 2004.
- [Kan06] Kunihiiko Kaneko. *Life: An Introduction to Complex Systems Biology*. Understanding Complex Systems. Springer, Berlin ; New York, 2006.
- [Kes04] David Kestenbaum. The Not So Random Coin Toss. <https://www.npr.org/templates/story/story.php?storyId=1697475>, 2004.
- [KFG<sup>+</sup>07] Kristen M. Kwan, Esther Fujimoto, Clemens Grabher, Benjamin D. Mangum, Melissa E. Hardy, Douglas S. Campbell, John M. Parant, H. Joseph Yost, John P. Kanki, and Chi-Bin Chien. The Tol2kit: A multisite gateway-based construction kit for Tol2 transposon transgenesis constructs. *Developmental Dynamics*, 236(11):3088–3099, November 2007.
- [KHI97] N.L. Kleinman, S.D. Hill, and V.A. Ilenda. SPSA/SIMMOD optimization of air traffic delay cost. In *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)*, pages 1121–1125 vol.2, Albuquerque, NM, USA, 1997. IEEE.
- [KHM<sup>+</sup>15] Kevin H. Knuth, Michael Habeck, Nabin K. Malakar, Asim M. Mubeen, and Ben Placek. Bayesian evidence and model selection. *Digital Signal Processing*, 47:50–67, December 2015.
- [KKT<sup>+</sup>13] Vijayalakshmi Kari, Oleksandra Karpiuk, Bettina Tieg, Malte Kriegs, Ekkehard Dikomey, Heike Krebber, Yvonne Begus-Nahrmann, and Steven A. Johnsen. A Subset of Histone H2B Genes Produces Polyadenylated mRNAs under a Variety of Cellular Conditions. *PLoS ONE*, 8(5):e63745, May 2013.

- [KKZ<sup>+</sup>18] Naeh L. Klages-Mundt, Ashok Kumar, Yuexuan Zhang, Prabodh Kapoor, and Xuetong Shen. The Nature of Actin-Family Proteins in Chromatin-Modifying Complexes. *Frontiers in Genetics*, 9, September 2018.
- [KMF<sup>+</sup>09] Noam Kaplan, Irene K. Moore, Yvonne Fondufe-Mittendorf, Andrea J. Gossett, Desiree Tillo, Yair Field, Emily M. LeProust, Timothy R. Hughes, Jason D. Lieb, Jonathan Widom, and Eran Segal. The DNA-encoded nucleosome organization of a eukaryotic genome. *Nature*, 458(7236):362–366, March 2009.
- [Kon06] Toru Kondo. Epigenetic alchemy for cell fate conversion. *Current Opinion in Genetics & Development*, 16(5):502–507, October 2006.
- [Kor05] Alfred Korzybski. *Science and Sanity: An Introduction to Non-Aristotelian Systems and General Semantics*. Inst. of General Semantics, Brooklyn, N.Y, 5. ed., 3. print edition, 2005.
- [Koz08] Zbynek Kozmik. The role of Pax genes in eye evolution. *Brain Research Bulletin*, 75(2-4):335–339, March 2008.
- [KP01] Walter G. Kelley and Allan C. Peterson. *Difference Equations: An Introduction with Applications*. Harcourt Academic Press, San Diego, CA, USA, second edition, 2001.
- [KS12] Kevin H. Knuth and John Skilling. Foundations of Inference. *Axioms*, 1(1):38–73, June 2012.
- [KSN99] Nathan L. Kleinman, James C. Spall, and Daniel Q. Naiman. Simulation-Based Optimization with Stochastic Approximation Using Common Random Numbers. *Management Science*, 45(11):1570–1578, 1999.
- [LAA<sup>+</sup>06] M. Locker, M. Agathocleous, M. A. Amato, K. Parain, W. A. Harris, and M. Perron. Hedgehog signaling and the retina: Insights into the mechanisms controlling the proliferative properties of neural precursors. *Genes & Development*, 20(21):3036–3048, November 2006.
- [Lak76] Imre Lakatos. Falsification and the Methodology of Scientific Research Programmes. In Sandra G. Harding, editor, *Can Theories Be Refuted? Essays on the Duhem-Quine Thesis*, Synthese Library, pages 205–259. Springer Netherlands, Dordrecht, 1976.
- [Lan12] Hans Petter Langtangen. *A Primer on Scientific Programming with Python*. Number 6 in Texts in Computational Science and Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg, third edition, 2012.
- [Lar92] Ellen W. Larsen. Tissue strategies as developmental constraints: Implications for animal evolution. *Trends in Ecology & Evolution*, 7(12):414–417, December 1992.
- [LD09] Enhu Li and Eric H. Davidson. Building developmental gene regulatory networks. *Birth Defects Research Part C: Embryo Today: Reviews*, 87(2):123–130, June 2009.
- [LDT12] Benjamin W. Lindsey, Audrey Darabie, and Vincent Tropepe. The cellular composition of neurogenic periventricular zones in the adult zebrafish forebrain. *The Journal of Comparative Neurology*, 520(10):2275–2316, July 2012.

- [LGBS00] Te-Won Lee, M. Girolami, A.J. Bell, and T.J. Sejnowski. A unifying information-theoretic framework for independent component analysis. *Computers & Mathematics with Applications*, 39(11):1–21, June 2000.
- [LHKR08] Deepak A. Lamba, Susan Hayes, Mike O. Karl, and Thomas Reh. Baf60c is a component of the neural progenitor-specific BAF complex in developing retina. *Developmental Dynamics*, 237(10):3016–3023, September 2008.
- [LHO<sup>+</sup>00] Zheng Li, Minjie Hu, Małgorzata J. Ochocinska, Nancy M. Joseph, and Stephen S. Easter. Modulation of cell proliferation in the embryonic retina of zebrafish (*Danio rerio*). *Developmental Dynamics*, 219(3):391–401, 2000.
- [LS12] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–359, April 2012.
- [LV08] Ming Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Texts in Computer Science. Springer, New York, 3rd ed edition, 2008.
- [LWR<sup>+</sup>07] Julie Lessard, Jiang I. Wu, Jeffrey A. Ranish, Mimi Wan, Monte M. Winslow, Brett T. Staahl, Hai Wu, Ruedi Aebersold, Isabella A. Graef, and Gerald R. Crabtree. An Essential Switch in Subunit Composition of a Chromatin Remodeling Complex during Neural Development. *Neuron*, 55(2):201–215, July 2007.
- [Ma00] T. Ma. Cell cycle-regulated phosphorylation of p220NPAT by cyclin E/Cdk2 in Cajal bodies promotes histone gene transcription. *Genes & Development*, 14(18):2298–2313, September 2000.
- [MAA<sup>+</sup>01] Till Marquardt, Ruth Ashery-Padan, Nicole Andrejewski, Raffaella Scardigli, Francois Guillemot, and Peter Gruss. Pax6 Is Required for the Multipotent State of Retinal Progenitor Cells. *Trends in Biochemical Sciences*, 30(3):i, 2001.
- [MAB<sup>+</sup>13] Gary R. Mirams, Christopher J. Arthurs, Miguel O. Bernabeu, Rafel Bordas, Jonathan Cooper, Alberto Corrias, Yohan Davit, Sara-Jane Dunn, Alexander G. Fletcher, Daniel G. Harvey, Megan E. Marsh, James M. Osborne, Pras Pathmanathan, Joe Pitt-Francis, James Southern, Nejib Zemzemi, and David J. Gavaghan. Chaste: An Open Source C++ Library for Computational Physiology and Biology. *PLoS Computational Biology*, 9(3):e1002970, March 2013.
- [MBE<sup>+</sup>07] Karine Massé, Surinder Bhamra, Robert Eason, Nicholas Dale, and Elizabeth A. Jones. Purine-mediated signalling triggers eye development. *Nature*, 449(7165):1058–1062, October 2007.
- [MDBN<sup>+</sup>05] Juan-Ramon Martinez-Morales, Filippo Del Bene, Gabriela Nica, Matthias Hammerschmidt, Paola Bovolenta, and Joachim Wittbrodt. Differentiation of the Vertebrate Retina Is Coordinated by an FGF Signaling Center. *Developmental Cell*, 8(4):565–574, April 2005.
- [MFKM12] Kiran Mahajan, Bin Fang, John M Koomen, and Nupam P Mahajan. H2B Tyr37 phosphorylation suppresses expression of replication-dependent core histone genes. *Nature Structural & Molecular Biology*, 19(9):930–937, August 2012.

- [MGv<sup>+</sup>09] Partha Mitra, Prachi N. Ghule, Margaretha van der Deen, Ricardo Medina, Rong-lin Xie, William F. Holmes, Xin Ye, Keiichi I. Nakayama, J. Wade Harper, Janet L. Stein, Gary S. Stein, and Andre J. van Wijnen. CDK inhibitors selectively diminish cell cycle controlled activation of the histone H4 gene promoter by p220<sup>NPAT</sup> and HiNF-P. *Journal of Cellular Physiology*, 219(2):438–448, May 2009.
- [MHR<sup>+</sup>99] Nicholas Marsh-Armstrong, Haochu Huang, Benjamin F Remo, Tong Tong Liu, and Donald D Brown. Asymmetric Growth and Development of the Xenopus laevis Retina during Metamorphosis Is Controlled by Type III Deiodinase. *Neuron*, 24(4):871–878, December 1999.
- [Min00] T.P. Minka. Estimating a Dirichlet Distribution. Technical Report, MIT, 2000.
- [MMDM04] Kathryn B. Moore, Kathleen Mood, Ira O. Daar, and Sally A. Moody. Morphogenetic Movements Underlying Eye Field Formation Require Interactions between the FGF and ephrinB1 Signaling Pathways. *Developmental Cell*, 6(1):55–67, January 2004.
- [Mor03] Michel Morange. *La Vie Expliquée: 50 Ans Après La Double Hélice*. Sciences. O. Jacob, Paris, 2003.
- [Mor08] Michel Morange. What history tells us XIII. Fifty years of the Central Dogma. *Journal of biosciences*, 33(2):171–175, 2008.
- [Mor09] Michel Morange. A new revolution? The place of systems biology and synthetic biology in the history of biology. *EMBO Reports*, 10(Suppl 1):S50–S53, August 2009.
- [Mor11] Michel Morange. Recent opportunities for an increasing role for physical explanations in biology. *Studies in History and Philosophy of Biol & Biomed Sci*, 42(2):139–144, 2011.
- [Mun19] Bernard Munos. 2018 New Drugs Approvals: An All-Time Record, And A Watershed. *Forbes*, January 2019.
- [Mur07] Kevin P Murphy. Conjugate Bayesian analysis of the Gaussian distribution. Technical Report, University of British Columbia, October 2007.
- [MXM<sup>+</sup>03] Partha Mitra, Rong-Lin Xie, Ricardo Medina, Hayk Hovhannisyan, S. Kaleem Zaidi, Yue Wei, J. Wade Harper, Janet L. Stein, André J. van Wijnen, and Gary S. Stein. Identification of HiNF-P, a Key Activator of Cell Cycle-Controlled Histone H4 Genes at the Onset of S Phase. *Molecular and Cellular Biology*, 23(22):8110–8123, November 2003.
- [MZLF02] A Murciano, J Zamora, J Lopez Sanchez, and J Frade. Interkinetic Nuclear Movement May Provide Spatial Clues to the Regulation of Neurogenesis. *Molecular and Cellular Neuroscience*, 21(2):285–300, October 2002.
- [Neu00] C. J. Neumann. Patterning of the Zebrafish Retina by a Wave of Sonic Hedgehog Activity. *Science*, 289(5487):2137–2139, September 2000.
- [NLM89] R. S. Nowakowski, S. B. Lewin, and M. W. Miller. Bromodeoxyuridine immunohistochemical determination of the lengths of the cell cycle and the DNA-synthetic phase for an anatomically defined population. *Journal of Neurocytology*, 18(3):311–318, June 1989.

- [NNM<sup>+</sup>19] Hayden Nunley, Mikiko Nagashima, Kamirah Martin, Alcides Lorenzo Gonzalez, Sachihiro C. Suzuki, Declan Norton, Rachel O. L. Wong, Pamela A. Raymond, and David K. Lubensky. Defect patterns on the curved surface of fish retinae suggest mechanism of cone mosaic formation. Preprint, Biophysics, October 2019.
- [NRN20] Elisa Nerli, Mauricio Rocha-Martins, and Caren Norden. Asymmetric neurogenic commitment of retinal progenitors involves Notch through the endocytic pathway. *eLife*, 9:e60462, November 2020.
- [NYLH09] Caren Norden, Stephen Young, Brian A. Link, and William A. Harris. Actomyosin Is the Main Driver of Interkinetic Nuclear Migration in the Retina. *Cell*, 138(6):1195–1208, September 2009.
- [PB04] David Posada and Thomas R. Buckley. Model Selection and Model Averaging in Phylogenetics: Advantages of Akaike Information Criterion and Bayesian Approaches Over Likelihood Ratio Tests. *Systematic Biology*, 53(5):793–808, October 2004.
- [PEM<sup>+</sup>09] David M. Parichy, Michael R. Elizondo, Margaret G. Mills, Tiffany N. Gordon, and Raymond E. Engeszer. Normal table of postembryonic zebrafish development: Staging by externally visible anatomy of the living fish. *Developmental Dynamics*, 238(12):2975–3015, December 2009.
- [Pfi20] Cameron Pfiffer. TuringLang/MCMCChains.jl. The Turing Language, December 2020.
- [PJ10] J. Pirngruber and S. A. Johnsen. Induced G1 cell-cycle arrest controls replication-dependent histone mRNA 3' end processing through p21, NPAT and CDK9. *Oncogene*, 29(19):2853–2863, 2010.
- [PKVH98] Muriel Perron, Shami Kanekar, Monica L. Vetter, and William A Harris. The genetic sequence of retinal development in the ciliary margin of the xenopus eye. *Developmental Biology*, (199):185–200, 1998.
- [PLM<sup>+</sup>17] Tao Peng, Linan Liu, Adam L MacLean, Chi Wut Wong, Weian Zhao, and Qing Nie. A mathematical model of mechanotransduction reveals how mechanical memory regulates mesenchymal stem cell fate decisions. *BMC Systems Biology*, 11, May 2017.
- [Poi13] Julia Pointner. *Genome-wide identification of nucleosome positioning determinants in Schizosaccharomyces pombe*. PhD thesis, Ludwig-Maximilians-Universität, München, July 2013.
- [PSS<sup>+</sup>09] Judith Pirngruber, Andrei Shchebet, Lisa Schreiber, Efrat Shema, Neri Minsky, Rob D Chapman, Dirk Eick, Yael Aylon, Moshe Oren, and Steven A Johnsen. CDK9 directs H2B monoubiquitination and controls replication-dependent histone mRNA 3'-end processing. *EMBO reports*, 10(8):894–900, August 2009.
- [Rab89] Lawrence R Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *PROCEEDINGS OF THE IEEE*, 77(2):30, 1989.

- [Rah] Ali Rahimi. An Erratum for “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition”. [https://alumni.media.mit.edu/~rahimi/rabiner/rabiner-errata/rabiner-errata.html#complex\\_xi](https://alumni.media.mit.edu/~rahimi/rabiner/rabiner-errata/rabiner-errata.html#complex_xi).
- [RBBP06] Pamela A. Raymond, Linda K. Barthel, Rebecca L. Bernardos, and John J. Perkowski. Molecular characterization of retinal stem cells and their niches in adult zebrafish. *BMC developmental biology*, 6(1):36, 2006.
- [RDT<sup>+</sup>01] J. T. Rasmussen, M. A. Deardorff, C. Tan, M. S. Rao, P. S. Klein, and M. L. Vetter. Regulation of eye development by frizzled signaling in Xenopus. *Proceedings of the National Academy of Sciences*, 98(7):3861–3866, March 2001.
- [Res00] Nicholas Rescher. *Nature and Understanding*. Oxford University Press, New York, 2000.
- [Res05] Nicholas Rescher. *Cognitive Harmony*. University of Pittsburgh Press, Pittsburgh, PA, 2005.
- [RO04] Jonathan M. Raser and Erin K. O’Shea. Control of Stochasticity in Eukaryotic Gene Expression. *Science; Washington*, 304(5678):1811–4, June 2004.
- [RTL<sup>+</sup>18] Jiong Ren, Cai-zhi Tang, Xu-Dong Li, Zhi-Bin Niu, Bo-Yang Zhang, Tao Zhang, Mei-Jiao Gao, Xin-Ze Ran, Yong-Ping Su, and Feng-Chao Wang. Identification of G2/M phase transition by sequential nuclear and cytoplasmic changes and molecular markers in mice intestinal epithelial cells. *Cell Cycle*, 17(6):780–791, March 2018.
- [Rv08] Arjun Raj and Alexander van Oudenaarden. Stochastic gene expression and its consequences. *Cell*, 135(2):216–226, October 2008.
- [Ryd08] Tobias Rydén. EM versus Markov chain Monte Carlo for estimation of hidden Markov models: A computational perspective. *Bayesian Analysis*, 3(4):659–688, December 2008.
- [Sad97] Payman Sadegh. Constrained Optimization via Stochastic Approximation with a Simultaneous Perturbation Gradient Approximation. *Automatica*, 33(5):889–892, May 1997.
- [Sch93] Kenneth F. Schaffner. *Discovery and Explanation in Biology and Medicine*. Science and Its Conceptual Foundations. University of Chicago Press, Chicago, 1993.
- [SF03] Deborah L Stenkamp and Ruth A Frey. Extraretinal and retinal hedgehog signaling sequentially regulate retinal differentiation in zebrafish. *Developmental Biology*, 258(2):349–363, June 2003.
- [SFM89] J. Sambrook, E. F. Fritsch, and T. Maniatis. Molecular cloning: A laboratory manual. *Molecular cloning: a laboratory manual.*, (Ed. 2), 1989.
- [SH14] Corinna Singleman and Nathalia G. Holtzman. Growth and Maturation in the Zebrafish, *Danio Rerio*: A Staging Tool for Teaching and Research. *Zebrafish*, 11(4):396–406, August 2014.
- [SK15] Zheng Sun and Natalia L. Komarova. Stochastic control of proliferation and differentiation in stem cell dynamics. *Journal of Mathematical Biology; Heidelberg*, 71(4):883–901, October 2015.

- [Ski06] John Skilling. Nested Sampling for Bayesian Computations. In *Proc. Valencia*, Benidorm (Alicante, Spain), June 2006. inference.org.uk.
- [Ski12] John Skilling. Bayesian computation in big spaces-nested sampling and Galilean Monte Carlo. In *BAYESIAN INFERENCE AND MAXIMUM ENTROPY METHODS IN SCIENCE AND ENGINEERING: 31st International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, pages 145–156, Waterloo, Ontario, Canada, 2012.
- [Ski19] John Skilling. Galilean and Hamiltonian Monte Carlo. In *Proceedings*, volume 33, page 8, Garching, Germany, 2019. MDPI.
- [SMFW10] Erik B Suderth, Michael I Mandel, William T Freeman, and Alan S Willsky. Distributed Occlusion Reasoning for Tracking with Nonparametric Belief Propagation. *Communications of the ACM*, 53(10):95–103, 2010.
- [Spa98] J. C. Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):817–823, July 1998.
- [Spe19] Joshua S. Speagle. Dynesty: A Dynamic Nested Sampling Package for Estimating Bayesian Posteriors and Evidences. *arXiv:1904.02180 [astro-ph, stat]*, April 2019.
- [SSGE82] G. D. Stormo, T. D. Schneider, L. Gold, and A. Ehrenfeucht. Use of the 'Perceptron' algorithm to distinguish translational initiation sites in *E. coli*. *Nucleic Acids Research*, 10(9):2997–3011, May 1982.
- [Ste18] Jacob Stegenga. *Medical Nihilism*. Oxford University Press, Oxford, United Kingdom, first edition edition, 2018.
- [STN<sup>+</sup>02] Masashi Sagara, Eri Takeda, Akiyo Nishiyama, Syunsaku Utsumi, Yoshiroh Toyama, Shigeki Yuasa, Yasuharu Ninomiya, and Takashi Imai. Characterization of functional regions for nuclear localization of NPAT. *The Journal of Biochemistry*, 132(6):875–879, 2002.
- [SVT13] Lourdes Serrano, Berta N Vazquez, and Jay Tischfield. Chromatin structure, pluripotency and differentiation. *Experimental Biology and Medicine*, 238(3):259–270, March 2013.
- [SZDG14] Fernando Soler-Toscano, Hector Zenil, Jean-Paul Delahaye, and Nicolas Gauvrit. Calculating Kolmogorov Complexity from the Output Frequency Distributions of Small Turing Machines. *PLoS ONE*, 9(5):e96223, May 2014.
- [TC87] David L. Turner and Constance L. Cepko. A common progenitor for neurons and glia persists in rat retina late in development. *Nature*, 328(9), July 1987.
- [TCM<sup>+</sup>07] Michael Y. Tolstorukov, Andrew V. Colasanti, David M. McCandlish, Wilma K. Olson, and Victor B. Zhurkin. A Novel Roll-and-Slide Mechanism of DNA Folding in Chromatin: Implications for Nucleosome Positioning. *Journal of Molecular Biology*, 371(3):725–738, August 2007.

- [TFC13] Vuong Tran, Lijuan Feng, and Xin Chen. Asymmetric distribution of histones during Drosophila male germline stem cell asymmetric divisions. *Chromosome Research*, 21(3):255–269, May 2013.
- [TK06] Dmitry N. Tsigankov and Alexei A. Koulakov. A unifying model for activity-dependent and activity-independent mechanisms predicts complete structure of topographic maps in ephrin-A deficient mice. *Journal of Computational Neuroscience*, 21(1):101–114, August 2006.
- [TMS64] J. E. Till, E. A. Mcculloch, and L. Siminovitch. A stochastic model of stem cell proliferation, based on the growth of spleen colony-forming cells. *Proceedings of the National Academy of Sciences of the United States of America*, 51:29–36, January 1964.
- [TR86] Sally Temple and Martin C. Raff. Clonal analysis of oligodendrocyte development in culture: Evidence for a developmental clock that counts cell divisions. *Cell*, 44(5):773–779, March 1986.
- [TR14] W.-W. Tee and D. Reinberg. Chromatin features and the epigenetic regulation of pluripotency states in ESCs. *Development*, 141(12):2376–2390, June 2014.
- [Tro00] V. Tropepe. Retinal Stem Cells in the Adult Mammalian Eye. *Science*, 287(5460):2032–2036, March 2000.
- [Tro08] Roberto Trotta. Bayes in the sky: Bayesian inference and model selection in cosmology. *Contemporary Physics*, 49(2):71–104, March 2008.
- [TSC90] D. L. Turner, E. Y. Snyder, and C. L. Cepko. Lineage-independent determination of cell type in the embryonic mouse retina. *Neuron*, 4(6):833–845, June 1990.
- [TSC08] Jeffrey M. Trimarchi, Michael B. Stadler, and Constance L. Cepko. Individual Retinal Progenitor Cells Display Extensive Heterogeneity of Gene Expression. *PLOS ONE*, 3(2):e1588, February 2008.
- [VGV<sup>+</sup>18] Jessie Van houcke, Emiel Geeraerts, Sophie Vanhunsel, An Beckers, Lut Noterdaeme, Marianne Christiaens, Ilse Bollaerts, Lies De Groef, and Lieve Moons. Extensive growth is followed by neurodegenerative pathology in the continuously expanding adult zebrafish retina. *Biogerontology*, October 2018.
- [VJM<sup>+</sup>09] Marta Vitorino, Patricia R Jusuf, Daniel Maurus, Yukiko Kimura, Shin-ichi Higashijima, and William A Harris. Vsx2 in the zebrafish retina: Restricted lineages through derepression. *Neural Development*, 4(1):14, 2009.
- [VL54] V. Viliter and L. Lewis. [Existence and distribution of mitoses in the retina of the deepsea fish Bathylagus benedicti]. - PubMed - NCBI. *C R Seances Soc Biol Fil.*, 148(21-22):1771–5, 1954.
- [vM77] L. v. Salvini-Plawen and Ernst Mayr. On the Evolution of Photoreceptors and Eyes. In Max K. Hecht, William C. Steere, and Bruce Wallace, editors, *Evolutionary Biology*, pages 207–263. Springer US, Boston, MA, 1977.

- [VV12] Antonio Visioli and Ramon Vilanova, editors. *PID Control in the Third Millennium: Lessons Learned and New Approaches*. Advances in Industrial Control. Springer, London New York, 2012.
- [Wad57] C H Waddington. *The Strategy of the Genes : A Discussion of Some Aspects of Theoretical Biology*. Routledge Taylor & Francis Group, London New York, 1957.
- [Wag07] Günter P. Wagner. The developmental genetics of homology. *Nature Reviews Genetics*, 8(6):473–479, 2007.
- [WAR<sup>+</sup>16] Y. Wan, A. D. Almeida, S. Rulands, N. Chalour, L. Muresan, Y. Wu, B. D. Simons, J. He, and W. A. Harris. The ciliary marginal zone of the zebrafish retina: Clonal and time-lapse analysis of a continuously growing tissue. *Development*, 143(7):1099–1107, April 2016.
- [WC17] Gregory M. Wright and Feng Cui. Nucleosome rotational positioning is influenced by cis- and trans-acting factors. *bioRxiv*, page 35, 2017.
- [WCG<sup>+</sup>20] Kim. J Westerich, K.S. Chandrasekaran, T. Gross-Thebing, N. Kuck, E. Raz, and A. Rentmeister. Bioorthogonal mRNA labeling at the poly(A) tail for imaging localization and dynamics in live zebrafish embryos. *Chemical Science*, 2020.
- [WD99] C. S. Wallace and D. L. Dowe. Minimum Message Length and Kolmogorov Complexity. *The Computer Journal*, 42(4):270–283, January 1999.
- [Wes00] M. Westerfield. The Zebrafish Book : A Guide for the Laboratory Use of Zebrafish. [http://zfin.org/zf\\_info/zbook/zfbk.html](http://zfin.org/zf_info/zbook/zfbk.html), 2000.
- [WF88] R. Wetts and S. E. Fraser. Multipotent precursors can give rise to all major cell types of the frog retina. *Science*, 239(4844):1142–1145, March 1988.
- [WG92] Robert W Williams and Dan Goldowitz. Lineage versus environment in embryonic retina: A revisionist perspective. *TINS*, 15(10):6, 1992.
- [WIEO04] Aiyan Wang, Tsuyoshi Ikura, Kazuhiro Eto, and Masato S. Ota. Dynamic interaction of p220NPAT and CBP/p300 promotes S-phase entry. *Biochemical and Biophysical Research Communications*, 325(4):1509–1516, December 2004.
- [Win15] R. Winklbauer. Cell adhesion strength from cortical tension - an integration of concepts. *Journal of Cell Science*, 128(20):3687–3693, October 2015.
- [WJH03] Y. Wei, J. Jin, and J. W. Harper. The Cyclin E/Cdk2 Substrate and Cajal Body Component p220NPAT Activates Histone Transcription through a Novel LisH-Like Domain. *Molecular and Cellular Biology*, 23(10):3669–3680, May 2003.
- [WPMT15] Loksum Wong, Namita Power, Amanda Miles, and Vincent Tropepe. Mutual antagonism of the paired-type homeobox genes, vsx2 and dmbx1, regulates retinal progenitor cell cycle exit upstream of ccnd1 expression. *Developmental Biology*, 402(2):216–228, June 2015.
- [WR90] Takashi Watanabe and Martin C. Raff. Rod photoreceptor development in vitro: Intrinsic properties of proliferating neuroepithelial cells change as development proceeds in the rat retina. *Neuron*, 4(3):461–467, March 1990.

- [WR09] L. L. Wong and D. H. Rapaport. Defining retinal progenitor cell competence in *Xenopus laevis* by clonal analysis. *Development*, 136(10):1707–1715, May 2009.
- [WSM<sup>+</sup>05] Ann M. Wehman, Wendy Staub, Jason R. Meyers, Pamela A. Raymond, and Herwig Baier. Genetic dissection of the zebrafish retinal stem-cell compartment. *Developmental Biology*, 281(1):53–65, May 2005.
- [WSP<sup>+</sup>09] Minde I. Willardsen, Arminda Suli, Yi Pan, Nicholas Marsh-Armstrong, Chi-Bin Chien, Heithem El-Hodiri, Nadean L. Brown, Kathryn B. Moore, and Monica L. Vetter. Temporal regulation of Ath5 gene expression during eye development. *Developmental Biology*, 326(2):471–481, February 2009.
- [Yam05] M. Yamaguchi. Histone deacetylase 1 regulates retinal neurogenesis in zebrafish by suppressing Wnt and Notch signaling pathways. *Development*, 132(13):3027–3043, July 2005.
- [YBW15] Fanny Yang, Sivaraman Balakrishnan, and Martin J. Wainwright. Statistical and computational guarantees for the Baum-Welch algorithm. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 658–665, Monticello, IL, September 2015. IEEE.
- [YDH<sup>+</sup>11] Jingye Yang, Huzefa Dungrawala, Hui Hua, Arkadi Manukyan, Lesley Abraham, Wesley Lane, Holly Mead, Jill Wright, and Brandt L. Schneider. Cell size and growth rate are major determinants of replicative lifespan. *Cell Cycle*, 10(1):144–155, January 2011.
- [You85] Richard W. Young. Cell differentiation in the retina of the mouse. *The Anatomical Record*, 212(2):199–205, June 1985.
- [YQW<sup>+</sup>18] Kai Yao, Suo Qiu, Yanbin V. Wang, Silvia J. H. Park, Ethan J. Mohns, Bhupesh Mehta, Xinran Liu, Bo Chang, David Zenisek, Michael C. Crair, Jonathan B. Demb, and Bo Chen. Restoration of vision after de novo genesis of rod photoreceptors in mammalian retinas. *Nature*, August 2018.
- [YSK15] Jienian Yang, Zheng Sun, and Natalia L. Komarova. Analysis of stochastic stem cell models with control. *Mathematical Biosciences*, 266(Supplement C):93–107, August 2015.
- [YWNH03] X. Ye, Y. Wei, G. Nalepa, and J. W. Harper. The Cyclin E/Cdk2 Substrate p220NPAT Is Required for S-Phase Entry, Histone Gene Expression, and Cajal Body Maintenance in Human Somatic Cells. *Molecular and Cellular Biology*, 23(23):8586–8600, December 2003.
- [ŽCC<sup>+</sup>05] Mihaela Žigman, Michel Cayouette, Christoforos Charalambous, Alexander Schleiffer, Oliver Hoeller, Dara Dunican, Christopher R. McCudden, Nicole Firnberg, Ben A. Barres, David P. Siderovski, and Juergen A. Knoblich. Mammalian Inscuteable Regulates Spindle Orientation and Cell Fate in the Developing Retina. *Neuron*, 48(4):539–545, November 2005.
- [ZDI<sup>+</sup>98] Jiyong Zhao, Brian Dynlacht, Takashi Imai, Tada-aki Hori, and Ed Harlow. Expression of NPAT, a novel substrate of cyclin E–CDK2, promotes S-phase entry. *Genes & development*, 12(4):456–461, 1998.

- [ZKL<sup>+</sup>00] Jiyong Zhao, Brian K. Kennedy, Brandon D. Lawrence, David A. Barbie, A. Gregory Matera, Jonathan A. Fletcher, and Ed Harlow. NPAT links cyclin E–Cdk2 to the regulation of replication-dependent histone gene transcription. *Genes & development*, 14(18):2283–2297, 2000.
- [ZMR<sup>+</sup>09] Yong Zhang, Zarmik Moqtaderi, Barbara P Rattner, Ghia Euskirchen, Michael Snyder, James T Kadonaga, X Shirley Liu, and Kevin Struhl. Intrinsic histone-DNA interactions are not the major determinant of nucleosome positions in vivo. *Nature Structural & Molecular Biology*, 16(8):847–852, August 2009.
- [Zub03] M. E. Zuber. Specification of the vertebrate eye by a network of eye field transcription factors. *Development*, 130(21):5155–5167, August 2003.
- [ZZLG08] Ya-Li Zhou, Qi-Zhi Zhang, Xiao-Dong Li, and Woon-Seng Gan. On the use of an SPSA-based model-free feedback controller in active noise control for periodic disturbances in a duct. *Journal of Sound and Vibration*, 317(3-5):456–472, November 2008.