

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 sophisticated statistical and computational techniques into the analysis of cell and molecular biological data has created many 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 defensible. 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 two distinct phases of proliferative activity with similar timing across morphological axes, despite asymmetrical population dynamics. Changes to RPC lineage commitment across the two phases suggest an explanation for the changing photoreceptor mosaic. These studies establish that CMZ RPCs are a dynamic population, 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 prove 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 advanced.

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 . I am finally indebted to Dr. John Skilling, both for delivering me to a coherent statistical understanding in his published work, and for a generous response to an 11th-hour 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>Précis of Modelling Studies</b>	<b>3</b>
<b>1 Canonical retinal progenitor cell phenomena and their explanations</b>	<b>5</b>
1.1 The Harris Stochastic Mitotic Mode Explanation (SMME) . . . . .	5
1.2 Explanations for RPC function in 2009 and the drive to unification . . . . .	6
1.3 Canonical vertebrate RPC phenomena: the RPC “morphogenetic alphabet” . . . . .	7
1.3.1 Proliferative phenomena . . . . .	8
1.3.2 Specificative phenomena . . . . .	9
1.3.3 Other morphogenetic phenomena . . . . .	11
1.4 Macromolecular mechanistic explanations for RPC phenomena . . . . .	12
1.4.1 Transcription factor networks . . . . .	12
1.4.2 Intercellular signalling networks . . . . .	13
1.4.3 Patterning mechanisms . . . . .	14
1.4.4 Chromatin dynamics . . . . .	15
1.5 A unified theory of RPC function? “Blurring” to order . . . . .	15
1.6 Explanatory Strategy and Intent of the SMME . . . . .	17
<b>2 “Stochastic mitotic mode” models do not explain zebrafish retinal progenitor lineage outcomes</b>	<b>19</b>
2.1 Introduction . . . . .	19
2.2 Results and Discussion . . . . .	21
2.2.1 Gomes SSM: Ancestral Model of the SMME SSMs . . . . .	22
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 . . . . .	25
2.2.4 Model selection demonstrates the SMME is not the best available explanation for RPC lineage outcomes . . . . .	26
2.2.5 SMME SSMs cannot explain the post-embryonic phase of CMZ-driven zebrafish retinal formation . . . . .	29
2.3 Conclusion . . . . .	33

<b>3 Toward a computational CMZ model comparison framework</b>	<b>36</b>
3.1 SMME Postmortem: a wrong turn at Gomes . . . . .	36
3.2 Implications of the SMME’s failure for modelling CMZ RPCs . . . . .	38
3.3 Desiridata for spatial CMZ models in a putative model comparison framework . . . . .	40
3.3.1 Spatial dimension of the models: the ”slice model” . . . . .	41
3.3.2 Temporal resolution of the models . . . . .	42
3.4 Bayesian decisionmaking for structuring models under uncertainty . . . . .	42
<b>4 Bayesian periodization of postembryonic CMZ activity</b>	<b>44</b>
4.1 Preliminaries: Calculation of Bayesian evidence supports independent Log-Normal modelling of CMZ parameters . . . . .	44
4.2 Survey of CMZ population and gross retinal contribution . . . . .	48
4.3 Two-phase periodization of postembryonic CMZ activity by phased difference equation modelling . . . . .	50
4.4 Slice-model characterisation of asymmetrical CMZ population dynamics demonstrates anatomical homogeneity of proliferative schedule . . . . .	55
4.4.1 Cumulative thymidine labelling supports Galilean Monte Carlo Nested Sampling parameter estimate . . . . .	59
4.5 The CMZ contributes stably to each cellular layer with time-variable lineage composition . . . . .	60
4.6 Early retinal cohorts of the <i>D. rerio</i> retina are turned over at a low rate by 4C4-positive microglia . . . . .	65
4.7 Summary: Two-phase periodisation of postembryonic CMZ activity & functional significance of CMZ activity . . . . .	67
<b>5 Mutant npat results in nucleosome positioning defects in <i>D. rerio</i> CMZ progenitors, blocking specification but not proliferation</b>	<b>69</b>
5.1 Introduction . . . . .	69
5.2 Results . . . . .	71
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 . . . . .	71
5.2.2 The microphthalmic zebrafish line <i>rys</i> is an npat mutant . . . . .	78
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 . . . . .	84
5.3 Discussion . . . . .	92
<b>6 Inferring and modelling specification dynamics in retinal progenitors</b>	<b>96</b>
6.1 Specification as a function of non-proliferative chromatin dynamics . . . . .	96
6.2 Future directions . . . . .	97
<b>II Software Technical Reports</b>	<b>99</b>
<b>7 GMC_NS.jl: Nested sampling by Galilean Monte Carlo for biological models</b>	<b>100</b>
7.1 Implementation notes . . . . .	100

7.2	Ensemble, Model, and Model Record interfaces . . . . .	101
7.3	Usage notes . . . . .	102
7.3.1	Setting up for a run . . . . .	102
7.3.2	PID tuner settings for GMC . . . . .	102
7.3.3	Parallelization . . . . .	103
7.3.4	Displays . . . . .	103
7.3.5	Example use . . . . .	104
<b>8</b>	<b>CMZNicheSims.jl: <i>D. rerio</i> CMZ RPC niche simulators</b>	<b>107</b>
8.1	Introduction . . . . .	107
8.2	CMZ_Ensemble model and usage . . . . .	108
8.3	Slice_Ensemble usage . . . . .	109
<b>9</b>	<b>BioBackgroundModels.jl: Parallel training of Hidden Markov Model zoos of genomic background noise</b>	<b>111</b>
9.1	Genome partitioning and sampling . . . . .	112
9.2	Optimizing BHMMs by EM algorithms . . . . .	113
9.3	BHMM analysis and display . . . . .	115
<b>10</b>	<b>BioMotifInference.jl: Independent component analysis motif inference by nested sampling</b>	<b>121</b>
10.1	Introduction . . . . .	121
10.2	Implementation of the nested sampling algorithm . . . . .	121
10.3	Usage notes . . . . .	123
10.3.1	Preparation of observation set . . . . .	123
10.3.2	IPM_Ensemble assembly . . . . .	124
10.3.3	Permute routine setup . . . . .	125
10.3.4	Nested sampling of IPM_Ensembles . . . . .	128
<b>III</b>	<b>Supplementary Materials</b>	<b>129</b>
<b>11</b>	<b>Supplementary materials for Chapter 2</b>	<b>130</b>
11.1	Materials and methods . . . . .	130
11.1.1	Zebrafish husbandry . . . . .	130
11.1.2	Proliferative RPC Histochemistry . . . . .	130
11.1.2.1	Anti-PCNA histochemistry . . . . .	130
11.1.2.2	Cumulative thymidine analogue labelling for estimation of CMZ RPC cell cycle length . . . . .	131
11.1.2.3	Whole retina thymidine analogue labelling of post-embryonic CMZ contributions . . . . .	131
11.1.3	Confocal microscopy and image analysis . . . . .	132
11.1.4	Estimation of CMZ annular population size . . . . .	132
11.1.5	Modelling lens growth . . . . .	132
11.1.6	Estimation of 3dpf CMZ cell cycle length . . . . .	132

11.1.7 CHASTE Simulations . . . . .	133
11.1.7.1 Project code . . . . .	133
11.1.7.2 SPSA optimisation of models . . . . .	133
11.2 Supporting figures . . . . .	135
<b>12 Supplementary materials for Chapter 4</b>	<b>140</b>
12.1 Materials and methods . . . . .	140
12.1.1 Zebrafish husbandry . . . . .	140
12.1.2 PCNA Immunohistochemistry . . . . .	140
12.1.3 Thymidine analogue histochemistry . . . . .	140
12.1.4 Lineage tracing immunohistochemistry . . . . .	140
12.1.5 4C4 immunohistochemistry . . . . .	141
12.1.6 Confocal micrograph acquisition and processing . . . . .	141
12.1.7 Estimation of overall CMZ population and retinal volume . . . . .	141
12.1.8 Monte Carlo estimation of CMZ population and retinal volume rates of change . . . . .	141
12.1.9 Simulation of phased CMZ activity by systems of difference equations . . . . .	141
12.1.10 Slice model simulations of phased CMZ activity . . . . .	141
12.1.11 Evidence estimation by Galilean Monte Carlo nested sampling . . . . .	141
12.1.12 Evidence estimation by Empirical Bayes linear regression . . . . .	142
12.1.13 Estimation of posterior distributions of phase model parameters . . . . .	142
12.2 Supplementary Figures . . . . .	142
12.3 Evidence calculations for models of layer and lineage contribution . . . . .	148
12.4 Likelihood ratio calculations for Normal and Log-Normal models of layer and lineage contribution . . . . .	148
<b>13 Supplementary material for Chapter 5</b>	<b>150</b>
13.1 Materials and methods . . . . .	150
13.1.1 Zebrafish husbandry . . . . .	150
13.1.2 Generation of transgenic vsx2:eGFP <i>rys</i> line . . . . .	150
13.1.3 Morpholino injections . . . . .	150
13.1.4 Morphological photography . . . . .	150
13.1.5 PCNA and EdU proliferative histochemistry . . . . .	150
13.1.6 BrdU pulse-chase assays . . . . .	151
13.1.7 Analysis of <i>rys</i> nuclear parameters by Galilean Monte Carlo Nested Sampling . . . . .	151
13.1.8 Progenitor identity marker immunohistochemistry . . . . .	151
13.1.9 Caspase-3 immunohistochemistry . . . . .	151
13.1.10 Pax6 immunohistochemistry . . . . .	151
13.1.11 In situ hybridization . . . . .	151
13.1.12 Electron microscopy . . . . .	151
13.1.13 RT-PCR analysis of <i>rys</i> npat expression . . . . .	151
13.1.14 <i>rys</i> MNase digestions and NGS . . . . .	151
13.1.15 Nucleosome position calling . . . . .	151
13.1.16 Analyses of <i>rys</i> nucleosome position disposition . . . . .	151

13.1.16.1	Background Hidden Markov modelling of <i>D. rerio</i> genomic sequence emission . . . . .	152
13.1.17	Evidence and maximum a posteriori estimation of ICA models of <i>rys</i> nucleosome position sequence emission . . . . .	152
13.2	Supplementary figures . . . . .	152
13.3	Supplementary tables . . . . .	160
<b>14</b>	<b>Theoretical Appendix A: Model theory and statistical methods</b>	<b>161</b>
14.1	Model Theory . . . . .	161
14.1.1	Bayesian Epistemological View on Model Comparison . . . . .	162
14.1.2	Model sampling and optimization . . . . .	164
14.1.3	Overfitting and underfitting . . . . .	165
14.1.4	Monte Carlo simulations . . . . .	166
14.1.5	Simple Stochastic Models . . . . .	166
14.1.6	Systems of difference equations . . . . .	167
14.1.7	Independent Component Analysis models of sequence emission . . . . .	168
14.1.8	Position Weight Matrices . . . . .	168
14.1.9	Hidden Markov Models . . . . .	169
14.2	Statistical Methods . . . . .	169
14.2.1	Simultaneous Perturbation Stochastic Approximation (SPSA) . . . . .	169
14.2.2	Bayesian parameter estimation . . . . .	169
14.2.2.1	Problems with frequentist inference using normal models of sample data .	169
14.2.2.2	The Bayesian approach to normal models of unknown mean and variance	169
14.2.3	Empirical Bayes linear regression . . . . .	170
14.2.4	Expectation Maximization optimization of HMMs . . . . .	170
14.2.5	Galilean Monte Carlo . . . . .	170
14.2.6	Nested sampling . . . . .	170
<b>15</b>	<b>Theoretical Appendix B: Metaphysical Arguments</b>	<b>171</b>
15.1	Chance is not a valid explanation for biological phenomena; Randomness is a measure of property of sequences and not an explanation . . . . .	171
15.1.0.1	Chance versus Randomness . . . . .	172
15.1.0.2	Chance in molecular mechanisms . . . . .	173
15.1.0.3	Can macromolecular chanciness be rooted in quantum indeterminacy? .	175
15.1.0.4	Randomness in RPC fate specification . . . . .	176
15.1.0.5	Summary: “Stochastic” or “variable”? . . . . .	176
15.2	Macromolecular mechanistic explanations in the Systems era . . . . .	177
15.2.1	The Feyerabendian modeller . . . . .	179
<b>16</b>	<b>Code Appendix</b>	<b>183</b>
16.1	Code and Output Archive . . . . .	183
16.2	SMME . . . . .	183
16.2.1	<code>setup_project.py</code> . . . . .	183
16.2.2	<code>/apps/src/BoijeSimulator.cpp</code> . . . . .	186

16.2.3	/apps/src/GomesSimulator.cpp . . . . .	193
16.2.4	/apps/src/HeSimulator.cpp . . . . .	199
16.2.5	/apps/src/WanSimDebug.cpp . . . . .	211
16.2.6	/apps/src/WanSimulator.cpp . . . . .	215
16.2.7	/python_fixtures/He_output_fixture.py . . . . .	222
16.2.8	/python_fixtures/Kolmogorov_fixture.py . . . . .	230
16.2.9	/python_fixtures/SPSA_fixture.py . . . . .	236
16.2.10	/python_fixtures/Wan_output_fixture.py . . . . .	253
16.2.11	/python_fixtures/diagram_utility_scripts/Gomes_He_cycle_plots.py . . .	257
16.2.12	/python_fixtures/diagram_utility_scripts/He_Boije_signal_plots.py . .	258
16.2.13	/python_fixtures/figure_plots/Cumulative_EdU.py . . . . .	260
16.2.14	/python_fixtures/figure_plots/He_output_plot.py . . . . .	262
16.2.15	/python_fixtures/figure_plots/Kolmogorov_plot.py . . . . .	281
16.2.16	/python_fixtures/figure_plots/Mitotic_rate_plot.py . . . . .	282
16.2.17	/python_fixtures/figure_plots/Wan_output_plot.py . . . . .	286
16.2.18	/src/BoijeCellCycleModel.cpp . . . . .	287
16.2.19	/src/BoijeCellCycleModel.hpp . . . . .	296
16.2.20	/src/BoijeRetinalNeuralFates.cpp . . . . .	302
16.2.21	/src/BoijeRetinalNeuralFates.hpp . . . . .	303
16.2.22	/src/GomesCellCycleModel.cpp . . . . .	307
16.2.23	/src/GomesCellCycleModel.hpp . . . . .	316
16.2.24	/src/GomesRetinalNeuralFates.cpp . . . . .	321
16.2.25	/src/GomesRetinalNeuralFates.hpp . . . . .	323
16.2.26	/src/HeAth5Mo.cpp . . . . .	328
16.2.27	/src/HeAth5Mo.hpp . . . . .	328
16.2.28	/src/HeCellCycleModel.cpp . . . . .	330
16.2.29	/src/HeCellCycleModel.hpp . . . . .	344
16.2.30	/src/OffLatticeSimulationPropertyStop.cpp . . . . .	350
16.2.31	/src/OffLatticeSimulationPropertyStop.hpp . . . . .	358
16.2.32	/src/WanStemCellCycleModel.cpp . . . . .	364
16.2.33	/src/WanStemCellCycleModel.hpp . . . . .	372
16.3	NGRefTools . . . . .	377
16.3.1	/src/LogNormalUtils.jl . . . . .	377
16.3.2	/src/MarginalTDist.jl . . . . .	377
16.3.3	/src/NGRef.jl . . . . .	381
16.3.4	/src/NGRefTools.jl . . . . .	382
16.3.5	/src/NIGRef.jl . . . . .	383
16.4	GMC_NS . . . . .	384
16.4.1	/src/GMC_NS.jl . . . . .	384
16.4.2	/src/GMC_NS_Model.jl . . . . .	386
16.4.3	/src/GMC/galilean_trajectory.jl . . . . .	387
16.4.4	/src/ensemble/GMC_NS_Eensemle.jl . . . . .	390
16.4.5	/src/nested_sampler/converge_ensemble.jl . . . . .	392

16.4.6 /src/nested_sampler/nested_step.jl . . . . .	395
16.4.7 /src/nested_sampler/search_patterns.jl . . . . .	396
16.4.8 /src/normal/LogNormal_Ensemble.jl . . . . .	400
16.4.9 /src/normal/LogNormal_Model.jl . . . . .	403
16.4.10 /src/normal/Normal_Ensemble.jl . . . . .	404
16.4.11 /src/normal/Normal_Model.jl . . . . .	407
16.4.12 /src/utilities/coordinate_utils.jl . . . . .	408
16.4.13 /src/utilities/ellipsoid.jl . . . . .	409
16.4.14 /src/utilities/ensemble_utilities.jl . . . . .	411
16.4.15 /src/utilities/ns_progressmeter.jl . . . . .	416
16.4.16 /src/utilities/progress_displays.jl . . . . .	421
16.4.17 /src/utilities/stats.jl . . . . .	424
16.4.18 /src/utilities/t_Tuner.jl . . . . .	424
16.4.19 /test/normal_demo.jl . . . . .	429
16.5 CMZNicheSims . . . . .	429
16.5.1 /src/BioSimpleStochastic.jl . . . . .	429
16.5.2 /src/CMZ_sim/CMZ_ensemble.jl . . . . .	430
16.5.3 /src/CMZ_sim/CMZ_lh.jl . . . . .	433
16.5.4 /src/CMZ_sim/CMZ_model.jl . . . . .	435
16.5.5 /src/slice_pop_sim/lens_model.jl . . . . .	436
16.5.6 /src/slice_pop_sim/slice_ensemble.jl . . . . .	437
16.5.7 /src/slice_pop_sim/slice_lh.jl . . . . .	440
16.5.8 /src/slice_pop_sim/slice_model.jl . . . . .	442
16.5.9 /src/thymidine_sim/thymidine_cell.jl . . . . .	443
16.5.10 /src/thymidine_sim/thymidine_ensemble.jl . . . . .	443
16.5.11 /src/thymidine_sim/thymidine_lh.jl . . . . .	446
16.5.12 /src/thymidine_sim/thymidine_model.jl . . . . .	448
16.5.13 /src/thymidine_sim/thymidine_sim.jl . . . . .	449
16.6 BioBackgroundModels . . . . .	452
16.6.1 /src/BioBackgroundModels.jl . . . . .	452
16.6.2 /src/API/EM_master.jl . . . . .	453
16.6.3 /src/API/genome_sampling.jl . . . . .	457
16.6.4 /src/API/reports.jl . . . . .	460
16.6.5 /src/BHMM/BHMM.jl . . . . .	461
16.6.6 /src/EM/EM_converge.jl . . . . .	463
16.6.7 /src/EM/baum-welch.jl . . . . .	464
16.6.8 /src/EM/chain.jl . . . . .	468
16.6.9 /src/EM/churbanov.jl . . . . .	469
16.6.10 /src/genome_sampling/partition_masker.jl . . . . .	472
16.6.11 /src/genome_sampling/sequence_sampler.jl . . . . .	480
16.6.12 /src/likelihood_funcs/bg_lh_matrix.jl . . . . .	490
16.6.13 /src/likelihood_funcs/hmm.jl . . . . .	493
16.6.14 /src/reports/chain_report.jl . . . . .	494

16.6.15 /src/reports/partition_report.jl . . . . .	499
16.6.16 /src/reports/replicate_convergence.jl . . . . .	501
16.6.17 /src/utilities/BBG_analysis.jl . . . . .	505
16.6.18 /src/utilities/BBG_progressmeter.jl . . . . .	506
16.6.19 /src/utilities/HMM_init.jl . . . . .	509
16.6.20 /src/utilities/load_balancer.jl . . . . .	510
16.6.21 /src/utilities/log_prob_sum.jl . . . . .	511
16.6.22 /src/utilities/model_display.jl . . . . .	511
16.6.23 /src/utilities/observation_coding.jl . . . . .	512
16.6.24 /src/utilities/utilities.jl . . . . .	515
16.6.25 /test/ref_fns.jl . . . . .	516
16.6.26 /test/runitests.jl . . . . .	520
16.6.27 /test/synthetic_sequence_gen.jl . . . . .	545
16.7 BioMotifInference . . . . .	548
16.7.1 /src/BioMotifInference.jl . . . . .	548
16.7.2 /src/IPM/ICA_PWM_Model.jl . . . . .	550
16.7.3 /src/IPM/IPM_likelihood.jl . . . . .	555
16.7.4 /src/IPM/IPM_prior_utilities.jl . . . . .	561
16.7.5 /src/ensemble/IPM_Eensemle.jl . . . . .	564
16.7.6 /src/ensemble/ensemble_utilities.jl . . . . .	569
16.7.7 /src/nested_sampler/converge_ensemble.jl . . . . .	573
16.7.8 /src/nested_sampler/nested_step.jl . . . . .	577
16.7.9 /src/permuation/Permute_Tuner.jl . . . . .	580
16.7.10 /src/permuation/orthogonality_helper.jl . . . . .	584
16.7.11 /src/permuation/permute_control.jl . . . . .	586
16.7.12 /src/permuation/permute_functions.jl . . . . .	591
16.7.13 /src/permuation/permute_utilities.jl . . . . .	606
16.7.14 /src/utilities/model_display.jl . . . . .	612
16.7.15 /src/utilities/ns_progressmeter.jl . . . . .	616
16.7.16 /src/utilities/synthetic_genome.jl . . . . .	623
16.7.17 /src/utilities/worker_diagnostics.jl . . . . .	625
16.7.18 /src/utilities/worker_sequencer.jl . . . . .	626
16.7.19 /test/consolidate_unit_tests.jl . . . . .	626
16.7.20 /test/ensemble_tests.jl . . . . .	628
16.7.21 /test/lh_perf.jl . . . . .	632
16.7.22 /test/likelihood_unit_tests.jl . . . . .	633
16.7.23 /test/mix_matrix_unit_tests.jl . . . . .	638
16.7.24 /test/permute_func_tests.jl . . . . .	639
16.7.25 /test/permute_tuner_tests.jl . . . . .	646
16.7.26 /test/pwm_unit_tests.jl . . . . .	647
16.7.27 /test/runitests.jl . . . . .	650
16.7.28 /test/spike_recovery.jl . . . . .	651
16.8 AWSWrangler . . . . .	656

16.8.1 /src/AWSWrangler.jl . . . . .	656
<b>16.9 Analyses . . . . .</b>	<b>659</b>
16.9.1 /cmz/a10correlation.jl . . . . .	659
16.9.2 /cmz/a10dvratio.jl . . . . .	662
16.9.3 /cmz/a10dvslice.jl . . . . .	664
16.9.4 /cmz/a10nvln.jl . . . . .	670
16.9.5 /cmz/a10periodisation.jl . . . . .	674
16.9.6 /cmz/a10popsurvey.jl . . . . .	680
16.9.7 /cmz/a19lineagetrace.jl . . . . .	687
16.9.8 /cmz/a25GMC_NS.jl . . . . .	700
16.9.9 /cmz/a25dvlinsreg.jl . . . . .	703
16.9.10 /cmz/a25thymidinesim.jl . . . . .	706
16.9.11 /cmz/a27GMC_NS.jl . . . . .	708
16.9.12 /cmz/a27linreg.jl . . . . .	710
16.9.13 /rys/a37.jl . . . . .	712
16.9.14 /rys/a38.jl . . . . .	717
16.9.15 /rys/a38GMC_NS.jl . . . . .	719
16.9.16 /rys/caspase.jl . . . . .	723
16.9.17 /rys/em.jl . . . . .	725
16.9.18 /rys/qPCR.jl . . . . .	726
16.9.19 /rys/rysp_GMC_NS.jl . . . . .	732
16.9.20 /rys/ryspont.jl . . . . .	736
16.9.21 /rys/bg_models/BBM_refinement_analysis.jl . . . . .	740
16.9.22 /rys/bg_models/BBM_refinement_prep.jl . . . . .	741
16.9.23 /rys/bg_models/BBM_sample_prep.jl . . . . .	742
16.9.24 /rys/bg_models/BBM_survey.jl . . . . .	744
16.9.25 /rys/bg_models/BBM_survey_analysis.jl . . . . .	746
16.9.26 /rys/bg_models/BBM_survey_refinement.jl . . . . .	747
16.9.27 /rys/nested_sampling/dif_pos_assembly.jl . . . . .	749
16.9.28 /rys/nested_sampling/dif_pos_learner.jl . . . . .	751
16.9.29 /rys/nested_sampling/dif_pos_sample_prep.jl . . . . .	754
16.9.30 /rys/nested_sampling/local_test.jl . . . . .	757
16.9.31 /rys/nested_sampling/prior_test.jl . . . . .	758
16.9.32 /rys/nested_sampling/spike_recovery.jl . . . . .	763
16.9.33 /rys/position_analysis/position_overlap.jl . . . . .	766

# List of Tables

2.1	AIC values for models assessed against training and test datasets . . . . .	29
4.1	Likelihood ratio comparison between Normal and Log-Normal models of retinal population parameters . . . . .	45
4.2	Evidence favours Log-Normal models of retinal population parameters . . . . .	45
4.3	Evidence favours uncorrelated linear models of CMZ-population and retinal volume over time . . . . .	47
4.4	Evidence favours a 2-phase periodization of CMZ activity . . . . .	51
4.5	Maximum a posteriori parameter estimates for periodization models . . . . .	53
4.6	Evidence favours a combined slice model over separate dorsal and ventral models	58
4.7	Maximum a posteriori parameter estimates for slice models . . . . .	58
4.8	Evidence favours whole-CMZ linear cycle models over separate D/V models .	60
4.9	Evidence supports stable layer contributions with time-varying lineage contributions . . . . .	63
4.10	Evidence for linear regression models supports early cohort population stability	65
4.11	GMC-NS evidence estimates confirm Empirical Bayes analysis of early cohort stability . . . . .	66
5.1	Evidence-based ranking of phenomenal contributors to <i>ryst</i> phenotype . . . . .	75
5.2	Standard deviations of significance for <i>ryst</i> mutant transcript > sib or WT . .	83
5.3	Evidence favours separate emission models for the <i>ryst</i> mutant and sibling differential position sets . . . . .	88
12.1	Evidence for Normal vs. Log-Normal models of layer and lineage contribution . . . . .	148
12.2	Evidence for combined vs. split models of layer and lineage contribution across the dorso-ventral axis . . . . .	148
13.1	Evidence for Normal vs. Log-Normal models of layer and lineage contribution . . . . .	160

# List of Figures

1.1	Histogenetic birth order of retinal neurons . . . . .	10
2.1	Structure of Gomes SSM . . . . .	23
2.2	Structure of He SSM . . . . .	24
2.3	Structure of Boije SSM . . . . .	25
2.4	Model comparison: the SPSA-optimised He SSM and a deterministic alternative	28
2.5	Most zebrafish retinal neurons are contributed by the CMZ between one and three months of age . . . . .	30
2.6	Per-lineage probabilities of mitoses, He et al. observations compared to model output . . . . .	31
2.7	CMZ population of proliferating RPCs: estimates from observations and simulated Wan-type CMZs . . . . .	33
4.1	CMZ population and retinal volume estimates are uncorrelated at 3dpf . . . . .	47
4.2	Population and activity of the CMZ over the first year of <i>D.rerio</i> life . . . . .	49
4.3	Maximum a posteriori output of periodization models . . . . .	52
4.4	Kernel density estimates of marginal posterior parameter distributions, 2-phase model . . . . .	54
4.5	Developmental progression of dorso-ventral population asymmetry in the CMZ.	56
4.6	Maximum a posteriori output of total, dorsal, and ventral CMZ slice models .	57
4.7	Kernel density estimates of marginal posterior parameter distributions, total slice model . . . . .	59
4.8	Representative 23dpf lineage marker confocal micrographs . . . . .	61
4.9	Second-phase declines in CMZ-contributed IslPax6+ RGCs, PKC $\beta$ + bipolar neurons, and Zpr1+ double cones . . . . .	64
4.10	4C4+ microglia associate with and engulf EdU-labelled CMZ contributions in the specified neural retina . . . . .	66
5.1	<i>rjs</i> mutants exhibit a small-eye phenotype . . . . .	70
5.2	<i>rjs</i> CMZ populations start relatively small and quiescent, end abberantly large and proliferative . . . . .	72
5.3	<i>rjs</i> CMZ RPCs fail to contribute to the neural retina . . . . .	73
5.4	7dpf <i>rjs</i> CMZ RPCs display enhanced asymmetry, increased volume, decreased sphericity, and relative but not absolute enlargement . . . . .	74
5.5	RPC nuclei of the <i>rjs</i> CMZ display disorganized, loosely packed chromatin .	76

5.6	<i>rys</i> mutant CMZs have increased caspase-3 positive nuclei . . . . .	77
5.7	Mutant <i>rys</i> RPCs display expanded expression of early progenitor markers . . . . .	78
5.8	RT-PCR analysis reveals two aberrant intron retention variants in <i>rys</i> mutant npat transcripts . . . . .	79
5.9	Functional domains of Human NPAT compared to predicted wild-type and <i>rys</i> <i>Danio</i> npat . . . . .	80
5.10	In situ hybridization reveals progressive restriction of npat expression to the CMZ . . . . .	81
5.11	<i>rys</i> overexpress total and polyadenylated core histone transcripts . . . . .	82
5.12	48hpf embryos injected with an npat ATG-targeted morpholino display a small-eye phenotype . . . . .	84
5.13	ATG morpholino perturbation of npat recapitulates decline in mean central retinal population relative to CMZ . . . . .	85
5.14	<i>rys</i> chromosomes are differentially enriched and depleted of nucleosome position density and occupancy . . . . .	86
5.15	PWM sources detected in sibling differential nucleosome positions . . . . .	87
5.16	PWM sources detected in sibling differential nucleosome positions . . . . .	90
5.17	PWM sources detected in <i>rys</i> mutant differential sources . . . . .	91
7.1	Example BMI.jl Output . . . . .	106
9.1	Example BBM.jl Partition Report . . . . .	117
9.2	Example BBM.jl Replicate Report . . . . .	118
9.3	Example BBM.jl Chain Report - Part 1 . . . . .	119
9.4	Example BBM.jl Chain Report - Part 2 . . . . .	120
9.5	Example BBM.jl Chain Report - Part 3 . . . . .	120
12.1	Developmental progression of naso-temporal population asymmetry in the CMZ .	143
12.2	Developmental progression of naso-temporal population asymmetry in the CMZ .	144
12.3	Linear regressions of temporally correlated and uncorrelated models of central retinal and 30dpf CMZ-contributed cohorts . . . . .	145
12.4	Linear regressions performed on cumulative labelling data from dorsal, ventral, and combined CMZ sectional populations . . . . .	146
12.5	4C4+ microglia are associated with the CMZ . . . . .	147
13.1	Synteny Database output for the syntenic region containing <i>D. rerio</i> npat . . . . .	153
13.2	npat is overexpressed in <i>rys</i> . . . . .	154
13.3	10dpf mutant <i>rys</i> RPCs are mitotic . . . . .	155
13.4	4C4-positive microglia engulf <i>rys</i> mutant RPCs . . . . .	156
13.5	Morpholinos directed to npat result in histone transcript overexpression similar to <i>rys</i> . . . . .	157
13.6	Novel nucleosome positions in <i>rys</i> occur in similar numbers to those lost from sibs . . . . .	157
13.7	Test likelihoods for background HMMs trained on <i>D. rerio</i> genome partitions	158
13.8	PWM sources detected in combined sib and rys differential sources . . . . .	159

14.1 Simple stochastic stem cell model, representing probabilities of cell division events, excerpted from Fagan 2013 pg. 61. 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. . . . .	166
15.1 Cellular systems model-construction, excerpted from [Fag15, p.7]. The system of equations and subsequent steps are based on a 2-element wiring diagram. . . . .	178

## 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. 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 a variety of 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.

Readers of the .pdf document will find some text unobtrusively highlighted in plum throughout the thesis. Section indices highlighted in this way link to the specified section. Technical terms have also been highlighted when pertinent material is available in Part III to explain them for the reader who may be unfamiliar.

### 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 the Raff 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.

I have occasionally used two useful terms from explanation theory, when discussing models, for concision. These were unfamiliar to me from my biological training and so are defined here for others: explanandum - the fact to be explained explanans - the explanation itself

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

# **Part I**

# **Modelling Studies**

## Précis of Modelling Studies

The primary concern of this thesis is developing and assessing computational models of retinal progenitor cell (RPC) phenomena. While many different, well substantiated explanations for various aspects of eye development exist, these explanations have usually been supported by assessing the frequentist significance of observed effects, not by building formally testable models of the phenomena themselves. Of the explanations which could be subject to model comparison, virtually all of these are derivatives of the Simple Stochastic Model (SSM), dating to the earliest work, performed by Till and McCulloch [TMS64].

The explosion of molecular biological information has massively increased the number of parameters that might be included in RPC models. The question of how RPC activity might be successfully predicted and controlled *in vivo* may revolve around finding adequate models that explain retinogenetic processes. The development and assessment of such models is, therefore, of fundamental basic and applied interest. This is particularly so, in light of increasing interest in entraining RPCs for the purpose of retinal regenerative medicine. Our group is particularly interested in the possibility that RPCs located in the peripheral retinal annulus of the circumferential marginal zone (CMZ), which are present in zebrafish and mammals alike, could be harnessed for retinal repair.

It is surprising that, despite the use of SSMs, the literature on neural stem cells contains virtually no systematic statistical approaches to the construction, optimization, or comparison of models. This thesis is an attempt to address this problem. It proceeds in three basic strokes. Firstly, Chapters 1 and 2 evaluate the existing explanations of RPC behaviour, and find that those with formal model components are deficient, and cannot explain the activities of RPCs in the CMZ. Secondly, Chapters 3 and 4 propose a general CMZ modelling framework under which the Bayesian evidence for different model-hypotheses might be assessed, and, by testing a variety of population-level hypotheses, provide guidance and develop the methods required to test cell-based hypotheses. Lastly, Chapters ??, 5, and 6 introduce the zebrafish CMZ mutant *rys*, and apply some of the methods developed in the second stroke to explain aspects of the aberrant morphology and behaviour of *rys* RPCs.

Chapter 1 begins by summarizing the range of explanations that have been offered for RPC activities, and introduces the primary set of formal models that have recently been used to favour an explanation centered around a supposed stochastic mitotic mode of RPCs. Chapter 2 elucidates the structure and development of the stochastic mitotic mode explanation (SMME) models, finding fundamental logical errors in their interpretation of "stochasticity". Moreover, by pursuing a basic information theoretical model selection approach, it is demonstrated that the SMME is not even the best available explanation relying on this flawed notion of stochasticity.

Chapter 3 discusses the implications of Chapter 2 for models of RPCs generally, and in the CMZ particularly; it also recommends a better model selection approach from Bayesian theory, nested sampling. Chapter 4 surveys the activity of RPCs in the post-embryonic zebrafish CMZ, developing Bayesian methods for doing so. These methods include the use of nested sampling to solve the long-standing problem of estimating cell cycle parameters of subpopulations of cells from cumulative thymidine labelling measurements of the entire super-population. It concludes with recommendations for future modelling approaches to the CMZ, structured by the foregoing Bayesian analysis.

Chapter 5 identifies the causative mutation in *rys* and shows that the zebrafish CMZ mutant *rys* CMZ RPCs have disorganized chromatin, likely blocking differentiation but not proliferation. Nested sampling is used to demonstrate that the nuclear phenotype is likely the consequence of a shift in

nucleosome histone composition in *rys* progenitors. Chapter 6 discusses some of the theoretical issues raised by the *rys* analysis, and their implications for modelling retinogenesis more broadly, especially in integrating nuclear dynamics into models of RPCs.

# Chapter 1

## Canonical retinal progenitor cell phenomena and their explanations

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

The work presented in [Chapter 2](#) comprises a critical examination of the best-developed theory of *D. rerio* retinal progenitor cell (RPC) function, and a broader, global general modelling approach motivated by the examination’s findings. The theory in question originates with 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 the stochastic effects of two particular transcription factors.

The SMME for RPC function is of fundamental theoretical and practical interest. It arises from a concerted effort by the Harris group to make sense of the difficult problem of explaining 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, the Harris group would go on to promulgate a [simple stochastic model](#) of zebrafish RPC function invoking only two named macromolecules.

Harris thus seems to be making a bold attempt to cut the Gordian knot of conflicting evidential threads and advance a simple, comprehensible, “mind-sized” model of RPC function. In effect, Harris’ explanation for zebrafish RPC function 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 preeminence in the

---

<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...”

field, we have good reason to take seriously the possibility that he has succeeded. Examining whether this is the case is our first priority. In order to appreciate the scope of his theoretical maneuver, and possible alternatives to it, it is necessary to 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

In many ways, molecular biologists have been attempting to explain the same remarkable features of retinal progenitor cells for decades. Animal retinas, having relatively well-understood functions and highly stereotypical structures, seem like highly tractable tissues for typical molecular biological explanations. With well defined cell types present in tightly regulated, neurotopologically limited proportions and organisations, it is no surprise that both theoretically-inclined molecular biologists and clinically-inclined regenerative medical practitioners have been keenly interested in the retina<sup>2</sup>. The high level of 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. Retinal biologists have long offered explanations suggesting that RPCs are more-or-less identical and go through highly stereotypical macromolecular processes. It is, however, the persistently observed departures from this (by now, obviously naïve) conception that have occupied most of our attention.

We may crudely gather the RPC-related phenomena studied by biologists under the headings of proliferation, specification, and organisation. A molecular biologist guided by a preference for cognitive simplicity (that is, Occam’s Razor) will reasonably suppose that the simplest explanation for the regularity of retinal development is that any given RPC is executing the same strict “developmental program” as its neighbours. In other words, any one RPC lineages’ proliferative and specificative outcomes are the same as any other RPC lineage. If every progenitor produces a similar number of cells, and the progeny are specified in similar proportions, well-understood principles of cellular adhesion could understandably give rise to the characteristic organisation we observe in animal retinas. However, by the 1980s, vertebrate lineage tracing experiments were routinely revealing 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 a wide range of proliferative and specificative 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), 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

---

<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.

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. This is only one example of a general phenomenon experienced by molecular biologists, in which enumerationist research programs, directed at producing more and more facts about macromolecular involvement in cellular phenomena, have failed to generate additional theoretical understanding [Kan06]. 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 [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 “morphic-genetic alphabet”

The majority 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, specification, migration, growth, death, and extracellular matrix formation<sup>3</sup>. Since the vast majority of reported phenomena fall under the first two categories, those belonging to the last four are described collectively.

### 1.3.1 Proliferative phenomena

Clonal lineage tracing experiments have reliably found that vertebrate RPCs give rise to highly variable numbers of offspring over the collective proliferative "lifetime" of their descendants. The most dramatic of these findings found that rat RPC lineage sizes vary across two orders of magnitude *in vivo*, from 1 to over 200 [TSC90]<sup>4</sup>. 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 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 remarkable statement casually introducing a study of unusual mitoses in the retina of a deepsea fish:

It is conventional<sup>5</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

---

<sup>3</sup>I have renamed Larsen's categories to clarify them for the retinal context, but retained her conceptual scheme, which is discussed in more detail in the Theoretical Appendix.

<sup>4</sup>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, it is generally accepted that vertebrate RPC lineages tend to vary in size by at least one order of magnitude, from less than ten to tens of neurons.

<sup>5</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.

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 have generally, and in particular by Harris, been 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].

The length of the RPC cell cycle has been 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 has generally been inferred from clonal lineage size, although it has also been occasionally assayed directly in cumulative thymidine analogue labelling experiments. Vertebrate RPCs have generally been found to 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>+</sup>00]. After the central retina is specified, the RPC cell cycle once again lengthens, and is presumed to continue to slow until RPCs have completed differentiation<sup>6</sup>.

Finally, the orientation of the RPC division plane in mitosis has also been implicated in retinal organisation. The orientation of divisions has been associated both with the proliferative and differentiative fate 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[ŽCC<sup>+</sup>05]. 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 Specificative phenomena

Irrespective of their location in the retina, vertebrate RPC lineages have offspring which 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 said to be of different “types” on the basis of lineage fate outcomes [HBEH88, TSC90, WF88]. While some specified progenitors have propensities to give rise to similar cell types, these relations appear to be species-specific, and do not seem to define separate progenitor pools [AR08]. In general, then, RPCs are taken to be totipotent with respect to the neural retina- all of the cell types<sup>7</sup> of the differentiated retina are derived from similar RPC lineages. Very 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 the like [BRD<sup>+</sup>15].

<sup>6</sup>This presumption is demonstrably incorrect in the /textit{D. rerio} eye, see Figure 2.7

<sup>7</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.

Of particular note here 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 specificative behaviour of RPCs depended 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 former.

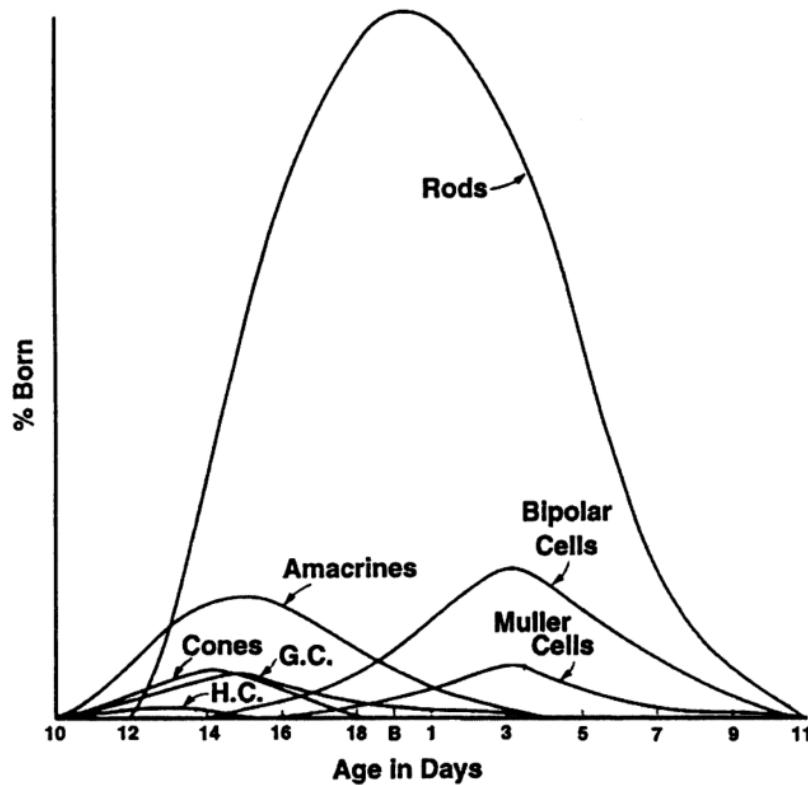


Figure 1.1: **Histogenetic birth order of retinal neurons**

Adapted from [You85] by [CAY<sup>+</sup>96]. 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>8</sup>, vertebrate retinal development displays a temporal 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 1.1. As noted above, RPCs also exit the cell cycle in a spatiotemporally defined

<sup>8</sup>Even in papers arguing for a strict, linear sequence of specificative outcomes in all RPC lineages, the actual data show 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.

order (from central to peripheral over time), and specification follows the same pattern<sup>9</sup>. This naturally gave rise to some question 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 specificative 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 the recent results of 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 identical (or even vaguely similar) specificative programs.

### 1.3.3 Other morphogenetic phenomena

The outcomes of RPC proliferation and lineage commitment have generally been taken to be sufficient to explain 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, 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>10</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 proliferative and differentiative outcomes, but is dissociable from them- both cell cycle and differentiation proceed (albeit in a precocious manner) when INM is disrupted[MZLF02]. The environment provided by the apical retinal surface appears to be required for RPC mitosis to proceed<sup>11</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 to what extent this short-range migration is related to 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 layers (consisting of axons and other neural processes) have begun to divide the cellular layers into bounded compartments.

Notable lacunae 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 hardly been studied at all. While cell size and growth are known to be tightly linked to proliferative behaviour in yeast [YDH<sup>+</sup>11], any related

<sup>9</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>10</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>11</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.

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, let us examine a selection of the many macromolecular mechanistic explanations (MEx) that have been offered to explain aspects of RPC behaviour. Unsurprisingly, given the field’s focus on the early events of eye development, these MEx are intended mainly to explain the phenomena associated with this period. Thus, the majority of MEx offered have been concerned to 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 that 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 MEx are found in the Pax6 “master gene” explanation popularised in the 1990s [Geh96]. This explanation revolved around the observation of 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 suggestion of Mayr and Salvini-Plawen, that differences in eye structure and function across clades suggested the independent appearance of eyes in more than 40 clades [vM77]. Pax6 was thus taken to provide 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 seems to be 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 morpholog-

ical 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>12</sup>. Moreover, the expansion of the moncausal “master gene” explanation, to include a network of TFs with broad gene regulatory effects, clearly highlighted the problems of complexity in offering MEx for RPC function. The components of this network interact in complex, context-dependent ways, and while the EFTFN as a whole is taken to promote RPC proliferation and to delay specification<sup>13</sup>, its individual 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 taken 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 MEx involves Chx10 (aka vsx2), a transcription factor which was found to be important for normal proliferation of RPCs, its loss causing microphthalmia in the mouse [BNL<sup>+</sup>96]. Chx10 was subsequently found to repress Mitf, which is 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 MEx 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.

### 1.4.2 Intercellular signalling networks

Virtually every developmentally significant class of signal has been implicated in RPC function, so that Harris, by 2009, simply glosses over a majority of these pathways by briefly summarising them in tabular form and not otherwise mentioning them [AH09]. These include BMP, CNTF, FGF, Glucagon,

---

<sup>12</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>13</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.

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 eye field transcription factor network (EFTFN), another well developed MEx described below [MBE<sup>+</sup>07]. 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].

Developmentally 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 advanced by Harris 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 (mainly advanced by Harris) 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]. 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 robust 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]. This code seems to be established, in part by the action of Gdf6a, 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, pharmacological inhibition of HDAC results in decreased proliferation and specification [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 clearly see Harris' 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, Harris 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, Harris effectively had 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. I have referred to this as the Enumerationist approach and explained why it has failed, and continues to fail, in the Theoretical Appendix.

The second option, the one actually chosen by Harris, is more theoretically sophisticated. As Nicholas Rescher has noted regarding the 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 actually be necessary to produce an ordered explanation that is useful with respect to some objective. Given the number of apparently contradictory MEx for RPC behaviour, we have exactly the kind of situation Rescher is describing- more sophistication, and more detail, has dissolved order, not revealed it<sup>14</sup>. The inability to assemble an unified explanation for RPC function has left us without a good fundamental understanding of how highly ordered neural tissues like eyes can be 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, the theoretical problem here 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].

---

<sup>14</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.

In a situation of this kind, it may well be that this type of “blurring” is required, and it seems that is what Harris is attempting in advancing his SMME. Harris is asking, in some sense, whether most of the MEx offered for RPC behaviour are extraneous to an adequate understanding of how RPCs work. By cutting down to the simplest possible explanation, Harris 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 Harris’ 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, which I summarise as follows:

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 provide support for the contention that zebrafish RPCs are a group of equipotent cells which give rise to variable outcomes that depend on independent “stochastic” processes within each of these cells. This support is to be provided by demonstrating that a suitably configured 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>15</sup> Secondly, the internal structure of the SSM should provide good reason to believe that one of the Noise explanations is a better explanatory option 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. With that said, let us turn to the SMME and determine how well these manuevers have succeeded.

---

<sup>15</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

*The text of this chapter has been adapted from a manuscript originally prepared for submission to PLOS One and is formatted accordingly; the methods and supplementary materials are available in ??*

### 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 naturally 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 dispute 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 cellular behaviours with unpredictably variable outcomes. Stem cell biologists have traditionally resorted to Simple Stochastic Models (SSMs) in order to explain the observed unpredictable variability in clonal outcomes of putative stem cells[TMS64, 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. 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 random variables with defined probability distribu-

tions. 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] 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 speciative outcomes of progenitor lineages.

In this report, we evaluate one of the best-developed of these explanations, proffered by William Harris’ retinal biology group. We have dubbed this 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 explain histogenetic ordering (the specification of some retinal neural types before others, notably the early appearance of retinal ganglion cells (RGCs)) in vertebrates, without reference to 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 typical model selection procedures, nor has their output been examined using modern information theoretic methods, as advocated by mainstream model selection theorists[BA02].

In order to facilitate the critical evaluation 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 strongly suggested that the introduction of an unexplained progression of temporal “phases” 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 behaviours. 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 the CMZ. 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 Results and Discussion

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 is concerned primarily to explain 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 biological explanations developing in time remains understudied. Perhaps most refined tool for global evaluation of biological theories (Schaffner's "Extended Theories"), treats biological explanations as hierarchically organised logical structures, after the fashion of Imre Lakatos, and proposes the use of Bayesian logic to distinguish between them[Sch93]. However, biologists rarely offer explanations in this form; we rather prefer MEx, expressed in diagrammatic form or as mathematical model-objects.

In this report, we accept Fagan's view 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[Fag15]. 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 often operate as epistemological anarchists; the development of our explanatory logic is not bound by an identifiable 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 the traditional 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 Faganian 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 theoretical options to explain observed variability in RPC lineage outcomes have historically been considered. Harris has previously 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 empirically-derived probability distributions for the time each cell takes to divide, 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. The model thus represents a scenario where the processes governing proliferation, leading to cell cycle exit, and governing cell fate commitment are, at every stage, causally independent of each other and of their foregoing history of outcomes. 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 behaviour of these cells.

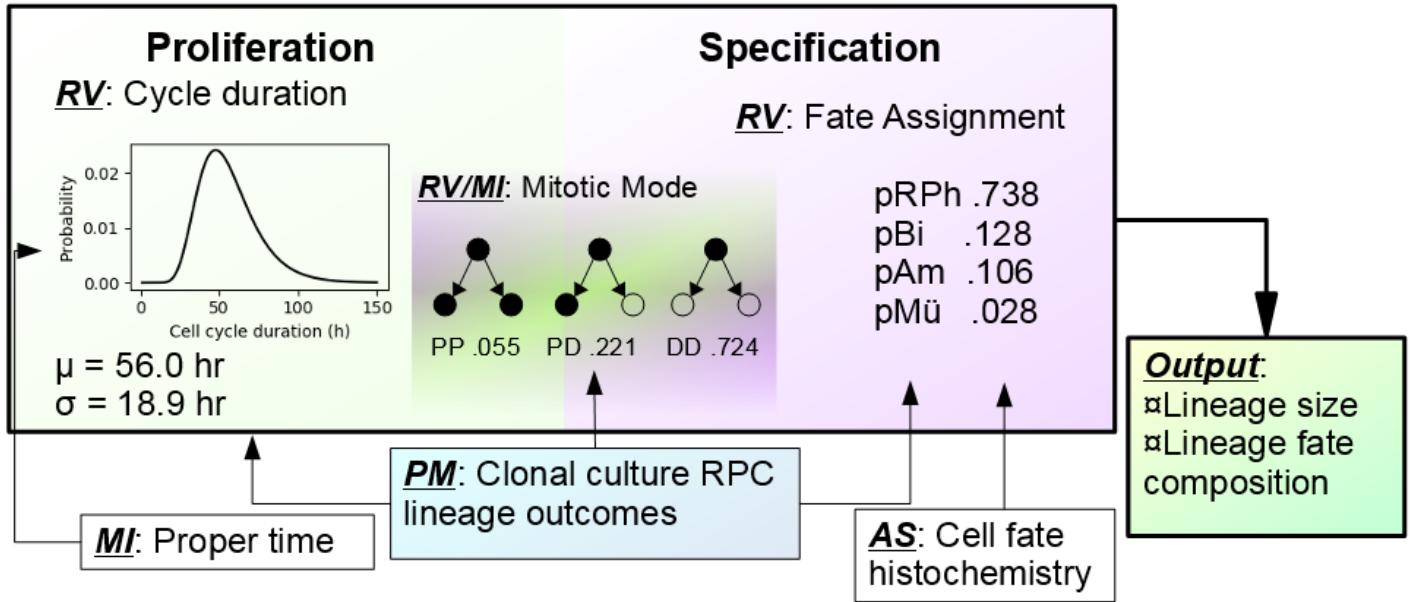


Figure 2.1: Structure of Gomes SSM

Structure of the Gomes SSM. 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. The close synchrony of zebrafish RPC divisions is modelled by assigning sister RPCs the same cycle length, shifted by a normal distribution of one hour width, 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, 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.

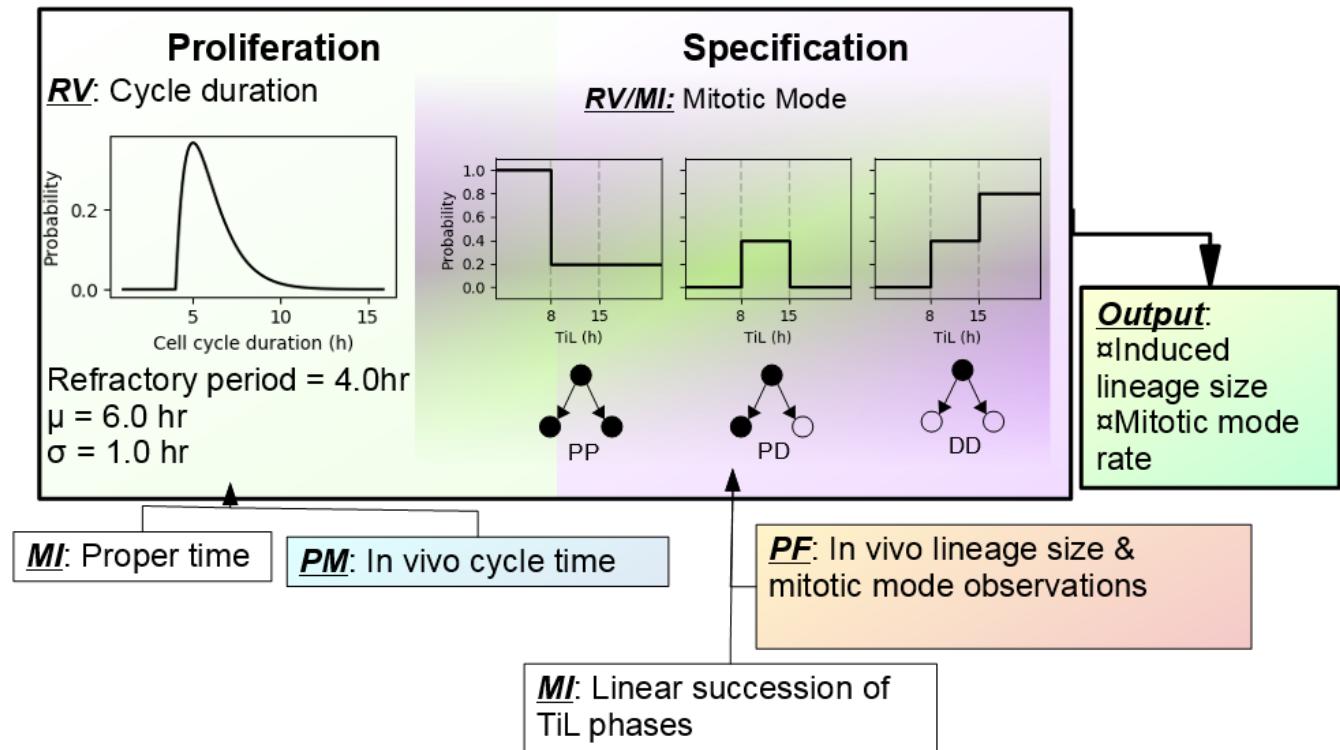


Figure 2.2: **Structure of He SSM**  
Structure of the He SSM. TiL, Time in Lineage.

The He model is, therefore, called on to explain the temporal structure of the proliferative and speciative 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 values the mitotic mode RV may take on is 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 effort is made 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 Ptfla, 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.

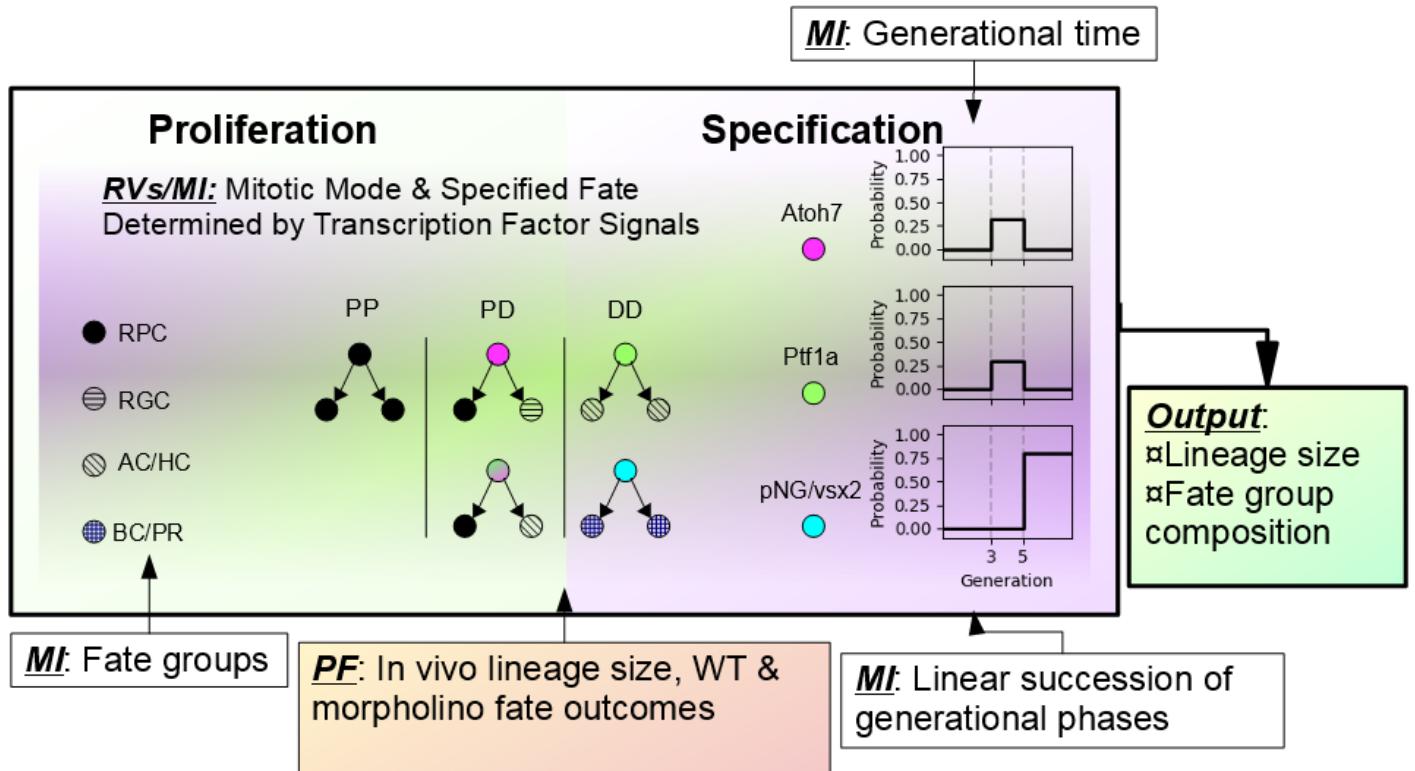


Figure 2.3: Structure of Boije SSM

Structure of the Boije SSM. 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 Ptfla 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, once again, emphasized throughout.

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

From a modeller’s perspective, 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 in any detail their fitting procedures, nor do any of the above report any statistical measures of goodness-of-fit. Additionally, no models representing the alternative “theoretical options” available to explain variability in RPC lineage outcomes 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, no effort is made to quantify their relative influence on model output, 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 this stochastic mitotic mode concept. Finally, the He and Boije SSMs contain more parameters, which are determined by fewer empirical measurements than the Gomes SSM. Clearly, then, the SMME SSMs are at significant risk of representing trivial overfits to their data, the modeller’s equivalent of the “just-so” story, not representing any actually-existing macromolecular or cellular process.

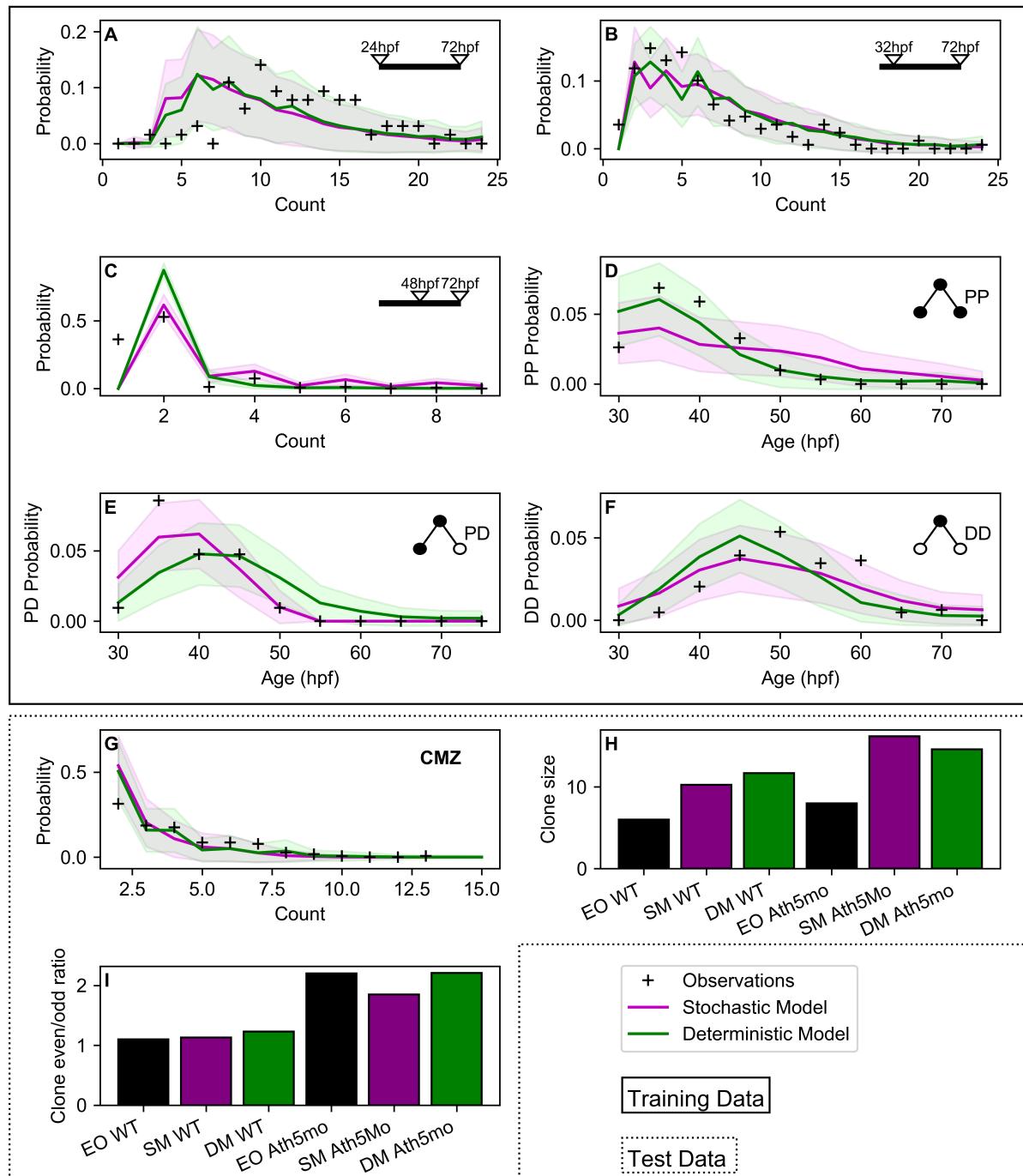
Fortunately, we may employ standard statistical model optimisation and selection techniques to adjudicate this possibility. The most straightforward way to do so is to reexamine the He SSM alongside a competing alternative model. The data to which the He SSM has been applied is particularly amenable to a scheme in which the models are fit to a “training” dataset, followed by a “test” dataset. In this case, the training data are the induced lineage size and mitotic mode rate data from He et al., and the test dataset are the Atoh7 morpholino observations from He et al., and the CMZ lineage size data from Wan et al. This allows us to test how well the He SSM, and an alternative, hold up under novel experimental conditions, without relying on a trivial ordering of goodness-of-fit to training data. Since the Boije SSM draws straightforwardly 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 differs only in its construal of the process governing the RPC mitotic mode. 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.

We take this to be a reasonable representation of the classic suggestion that RPCs step through linear succession of competency phases, given the conceptual tools that the He SSM uses to model mitotic phenomena. If we suppose that this temporal program is governed by RPC lineages 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 entirely plausible to suggest that lineages differ in the lengths of time they occupy each state. This is particularly true if we concede that an SSM, forego any spatial modelling whatsoever, must necessarily abstract extracellular and spatial influences on these processes. 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 SSMs mentioned above has not been published, the relevant reports provide enough detail to reconstruct these models in full, which we did using the CHASTE cell-based simulation framework, in order to provide transparent and reproducible implementations of the SSMs. Because the values selected for the He SSMs’ parameters seem to have been selected as a series of rough estimates, 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 any reasonable 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 idiosyncrasies. 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. Unsurprisingly, when we rebuilt 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 clearly not the best fit available for the He SSM, and we wish to directly compare the best fits (i.e. the parameterisations at minima of some loss function) for the He SSM and our putative alternative model, we used the simultaneous perturbation stochastic approximation (SPSA) algorithm[[Spa98](#)] to optimise both model fits. SPSA is particularly 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 rigorous comparison between the two differently-parameterised models (the deterministic alternative has two fewer parameters than the He SSM).



**Figure 2.4: 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 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.

After (re)-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. The combined results are displayed in Fig 2.4, with the output of the two models being presented separately in S2 Fig and S3 Fig. Remarkably, the DM closely recapitulates the output of the SM for both datasets. Moreover, while the more highly-parameterised SM permits a better fit to the training data, the DM proves to be a better fit to the test dataset, as summarised in Table 2.1. This is, therefore, a classic case of model overfitting: a higher-parameter model fits some training dataset better than a simpler model, but fails upon challenge with a new dataset. Given this model selection scheme, it is plain that we should choose the DM over the SM as a superior explanatory model, both on the basis of explanatory power and of Occam’s Razor.

Table 2.1: **AIC values for models assessed against training and test datasets**

Model	Training AIC	Test AIC
Stochastic (He fit)	-93.26	255.64
Stochastic (SPSA fit)	<b>-93.80</b>	92.24
Deterministic (SPSA fit)	-69.33	<b>86.60</b>

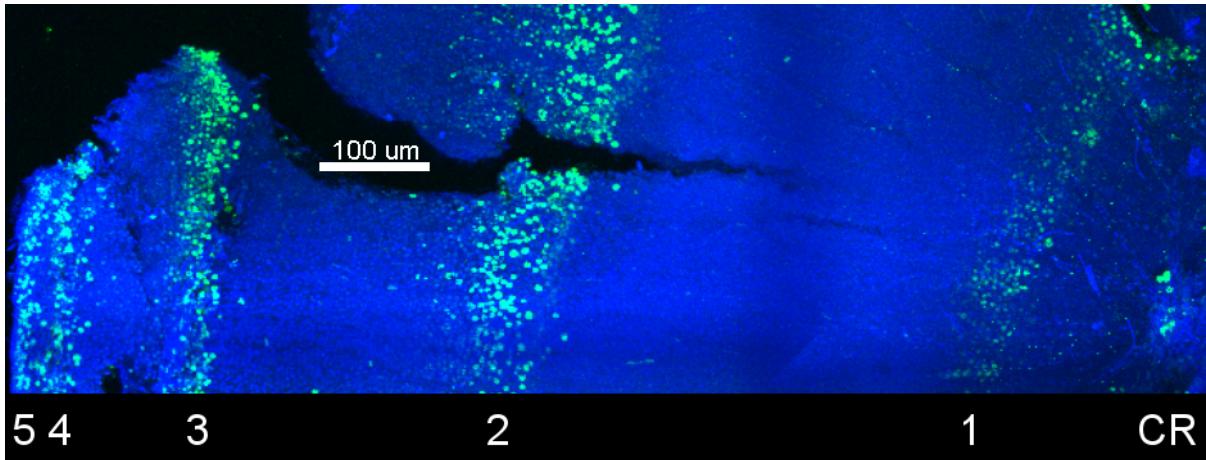
AIC: Akaike’s information criterion. Lower values reflect better model explanations of the noted dataset. Lowest values are noted in bold.

We therefore conclude that, had the Harris group employed standard model selection procedures, comparing plausible alternatives to their favoured explanation, they would have been forced to conclude that the SMME is not the best available explanation for variability in zebrafish RPC lineage outcomes. It is clear from this analysis that the assumed, but unexplained, linear succession of mitotic mode phases is what provides the overall structure of the model output. Variability in lineage outcomes may be supplied by entirely different model ingredients without any loss of explanatory power (indeed, with some improvement, in our case). The rhetorical emphasis on a stochastic process governing mitotic mode, ostensibly 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 conclude that the SSM model type is simply inadequate for the task of locating the source of variability in RPC lineage outcomes.

### 2.2.5 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>+09</sup>], 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 plate out the retina in annular cohorts. A typical “tree-ring” analysis from our studies, marking the DNA of cohorts of cells contributed to the retina at particular times with indelible thymidine analogues, is

shown in Fig 2.5. It is immediately obvious from such experiments that the structure of the zebrafish retina is quantitatively 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 2.5: 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.

Indeed, the SMME SSMs were originally brought to our attention when 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 2.4, 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 2.6.

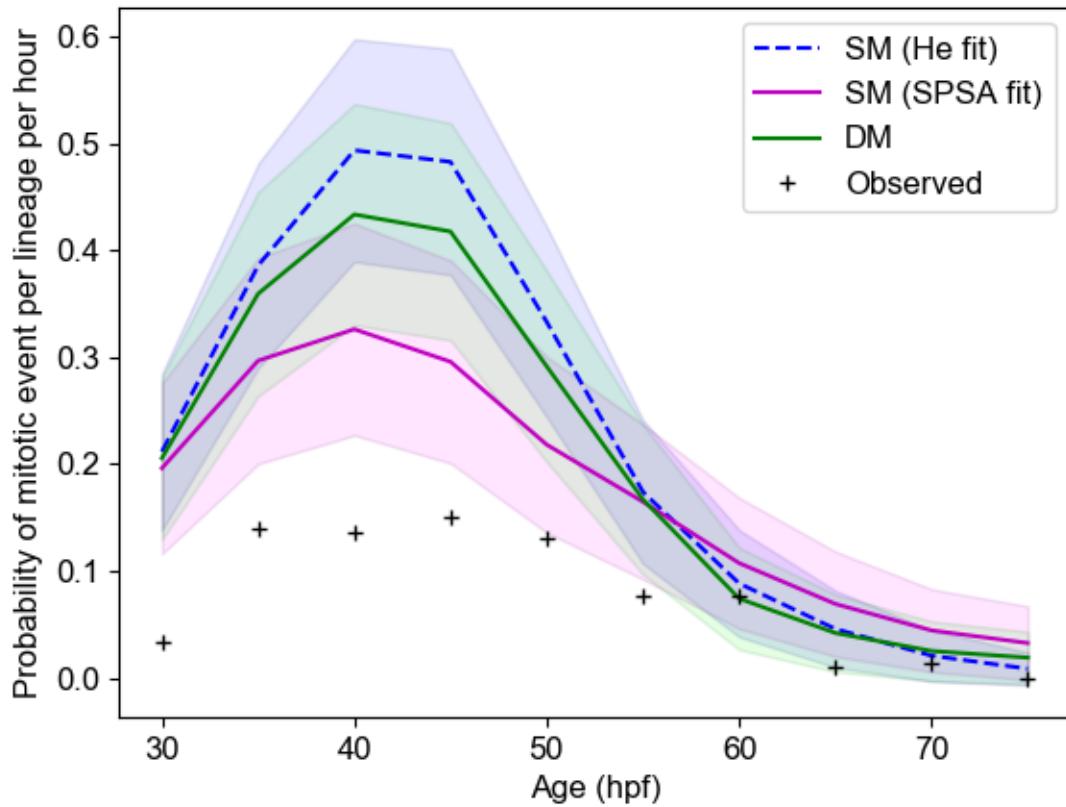


Figure 2.6: Per-lineage probabilities of mitoses, He et al. observations compared to model output

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.

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. Still, because we observed a massive build-up of proliferating RPCs between two the first few weeks of post-embryonic CMZ activity involve a massive build-up of proliferating RPCs between two and four weeks post-fertilisation, we thought this might actually suit this later context better.

In order to produce estimates of 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. An additional number of immortal stem cells amounting to one tenth of this total was added. 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 2.7, 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 strongly 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 are far more proliferatively active than in early development. It is likely that models that closely associate proliferative behaviour with fate specification, like those of the SMME, will necessarily 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 also possible that RPCs are substantially heterogenous with respect to their proliferative behaviour, and that this heterogeneity is mainly apparent later in development.

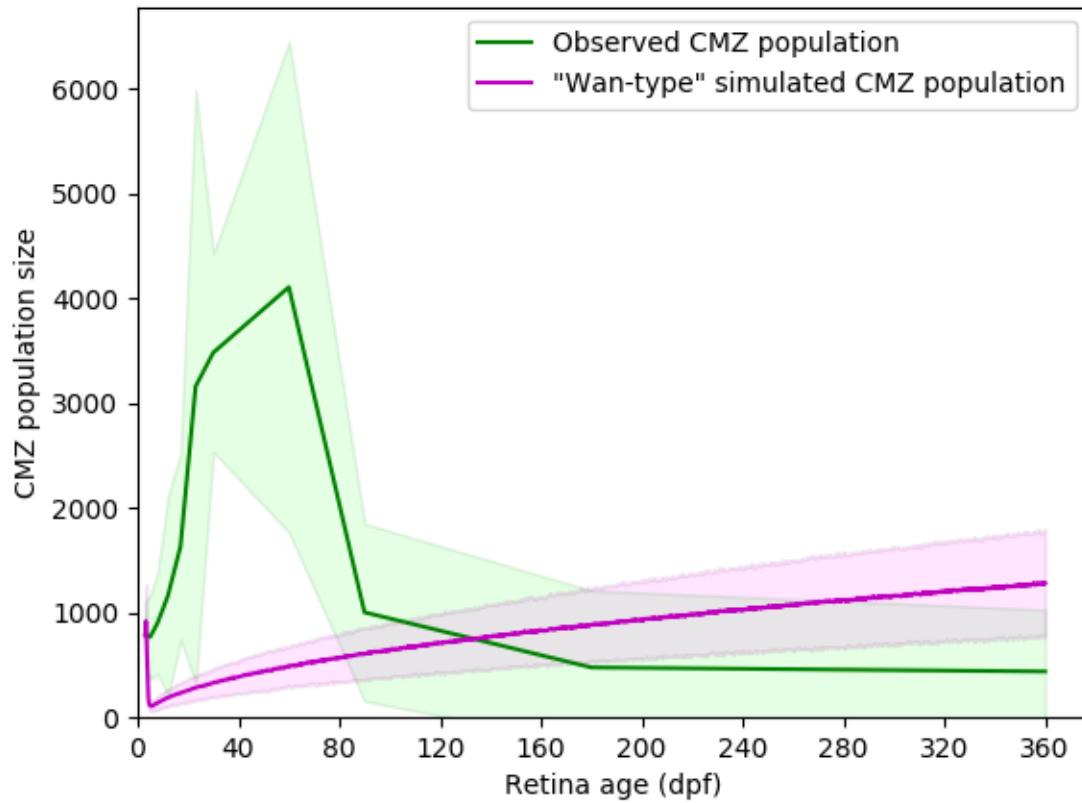


Figure 2.7: CMZ population of proliferating RPCs: estimates from observations and simulated Wan-type CMZs

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.

## 2.3 Conclusion

Simple stochastic models are familiar tools for stem cell biologists. Introduced by Till, McCulloch, and Siminovitch in 1964, 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 necessarily emphasizes an aspatial, lineage-centric view of tissue development, which cannot account for the generation of structurally complex tissues like retinas beyond cell numbers, and, perhaps, fate composition. As a result of this relatively loose relationship to the complex morphogenetic environment, there are few

constraints on the model configurations that may produce similar outputs. As we hope has been made plain by the analyses herein, this can result in modellers being led astray by their apparently good fits to observations. This is one reason why it has long been unacceptable in other biological modelling communities (particularly, among ecologists) to present the fit of one highly parameterised model 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 particular macromolecular or cellular referents that particular model constructs represent. The “mitotic mode” construct present in all SSMs is a particularly ambiguous and problematic 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 straightforwardly 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 variability may be responsible for the observed variability in RPC lineage outcomes. Since the extent of transcription noise may be “tuned” to a degree by e.g. promoter sequences[RO04], parameters related to this noise could plausibly be under selective pressure. There are two significant problems with identifying the SMME’s stochastic mitotic mode with this type of process. The first we have 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 therefore impossible to determine what sort of process might give rise to variability in RPC lineage outcomes by using SSMs in this fashion. The second, more fundamental problem is that a causal explanation of the presence of the signal, for which noise is a property, is elided entirely 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, in order to produce good explanations for both the order and unpredictable variability exhibited in RPC behaviours, more sophisticated models and more rigorous modelling practices are required. 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

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

Let us 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 that it has compounded several modelling failures to obscure the basic reality of the matter: the SMME models do not support an explanation based on the noise of some macromolecular process, they rather 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]. This is the fundamental 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 data better than its stochastic counterpart. Only by the process of counterinduction, the comparison of models with different propositional structures about reality, does this become clear.

The critical failing of the SMME models is where they depart from the original Gomes SSM: the assumption of the linear structure of temporal phases. This has to be done in order to model the early contribution of RGCs in zebrafish neurons, an essential explanandum for any explanation purporting to explain to retinogenesis in these animals; to the extent that it is left without some biological rationale, retinogenesis remains unexplained. It follows that the SMME does not achieve its lofty aim of being a complete description of the aspects of retinogenesis an SSM]SSM is capable of explaining, and this is confirmed by our observation that the Wan model [WAR<sup>+</sup>16] has no explanatory power outside the first few days of life, which makes up a tiny portion of the total retinal contribution to the zebrafish retina.

We must, therefore, assess that Harris' first theoretical maneuver, explaining the data with a model whose structure supports a stochastic resolution of mitotic mode “decisions”, fails in the face of an alternative which locates stochasticity elsewhere. Simply 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. 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 Harris' inferential logic, variability in RPC outcomes can be explained by variability in virtually any parameter which affects fate outcomes. The explanation collapses into itself, which is why Boije et al. [BRD<sup>+</sup>15] are forced to defend their idiosyncratic definition of "stochasticity" at length: variability is being used to explain variability. If "randomness" or "stochasticity" is accepted as a property of existents, rather than representations of our uncertainty about them, all biological variability is trivially explained by referring to itself. Because of the seriousness of this problem for biological inferences<sup>1</sup>, an explanation of the Bayesian epistemological view of probability has been provided in ???. Moreover, thorough explanation of the sense in which there is no good argument for "randomness" being a property of real existents is provided in Section 15.1. It suffices here to conclude that variability in RPC outcomes is not well explained by the SMME models, in part because their interpretation depends entirely on an incorrect understanding of the concept of stochasticity, and in part because they were never tested against alternatives.

The second theoretical maneuver was to nominate a particular macromolecular system as the physical locus of the stochastic process, in this case represented by *Atoh7* and *Ptfla* as labels for abstract Bernoulli distributed processes. It remains obscure in what sense the model variables relate to their namesake transcription factors (transcription of the TF itself? activation of other genes?). 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 would be superfluous.

On the basis of the above, we can 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 proliferative and speciative phenomena 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 developing 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>2</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 predominant importance of RPC-intrinsic processes in determining both the proliferative and speciative outcomes for late RPC lineages, since 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 these are the two most obvious and critical parameters that must

---

<sup>1</sup>Having experienced well-respected stem cell biologists confidently asserting that Harris' work proves that RPC mitotic outcomes are 'spontaneous', ie. causeless and without explanation, I believe these studies have been both influential and confusing for many.

<sup>2</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.

be achieved for RPCs to produce a functional retina of the appropriate size, the suggestion that both are largely determined by intrinsic processes seemed a striking 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 even 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 Harris is able to continue the rhetorical focus on the stochastic elements of the model, and so is lead down the garden path of spurious model fits and logically unsound interpretation of model structure.

## 3.2 Implications of the SMME's failure for modelling CMZ RPCs

Having taken seriously the possibility that an adequate model of zebrafish retinogenesis already exists, and concluded in the negative, we must proceed to a plan to rectify this state of affairs.

Firstly, the obvious lessons in statistical acumen must be learned. The statistical practices used in the SMME reports are not acceptable by any contemporary standard, and would not have been accepted in another field where practitioners are more familiar with modern statistical techniques. We must be dissuaded of the notion that a single, hand fitted model has any explanatory power at all; it may be an exercise in modelling prowess, but it is not a legitimate part of the quantitative scientific discourse unless some statistical property of the model is actually computed<sup>3</sup>. Moreover, to have any confidence about our interpretation of the model, we must test alternatives that make fundamentally 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 broad elements to consider here: the appropriateness of the overall 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, by far its most attractive features are its computational efficiency and the ease of producing a custom model to fit some particular case- for all their failings, 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.

---

<sup>3</sup>The commonly maligned Fischerian t-test procedure [HKB08, pp.181] is, in this sense, better than simply plotting some model output over observations.

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 determined 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.

We can therefore predict 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, a certain uneasiness is warranted. The AIC calculation is, after all, based on a single estimate for the locally optimal parameterisation of the model. Relative AIC rankings, then, are strongly influenced by the “well depth” of the AIC surface at the local minimum in parameter space. We can easily imagine a case where a model with a very narrow range of parameter space that fits data very well appears to be better, by the AIC metric, than one that fits the data slightly more loosely, but over a much broader range of the parameter space. The first model is a “fragile” fit, probably depending heavily on the particular structure of the training dataset, while the second would likely be much more robust across a range of observations; still, in this case, AIC would lead us to select the fragile model over its robust counterpart. 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 an eminently practical algorithm, useful in a wide variety of contexts, its statistical guarantee is only that it will almost-surely find the local minimum of the loss function. While

this will be good enough for many applications, for highly parameterised models with complex loss function surfaces, there will be many local minima for SPSA to get "stuck" in that are nowhere near the global optimum<sup>4</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 rapidly 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 14.2.6. 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 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 Desiridata for spatial CMZ models in a putative model comparison framework

The simulations presented in ?? 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>5</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 some weeks. We can therefore see how a simple approach to ranking basic models against a smallish dataset approaches the reasonable limits of what most labs are likely to be able to achieve with any given machine, in a month or so. Because the introduction of explicit three-dimensional spatial simulation implies, perhaps, an order of magnitude more computational time, we may plausibly be limited to one inference based on model comparison per year with local resources. The problem of computational limits to model selection is discussed in the relevant section of the Technical Appendix, so it will suffice here to state that this constraint dominates all other considerations in the selection of the overall boundaries within which we intend to build models of the CMZ. In any environment where 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 from the cheaper-but-inadequate in search of the adequate-but-still-affordable. In this case, we move from the aspatial SSM into the realm of explicit

---

<sup>4</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.

<sup>5</sup>I.e. the specified identity of post-proliferative cells in these models has no function within the model.

spatial modelling, 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 to compare many interesting hypotheses. Because numerous observations suggest that individual RPC lineages contribute to the retina in linear cohorts of neurons, it follows that any particular centrally-oriented slice of the CMZ annulus will be responsible for the generation of the neurons central to it. Conceptually, then, the retina can be thought of as 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 mainly 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. 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 and adjacent central neurons, the conceptual equivalent of the histological section through the eye<sup>6</sup>.

Slice models are especially attractive because the observed proliferative status, position, specified fate, etc. of simulated cells can be easily related to the summary statistics used to describe fixed sections of retinal tissue in this thesis and elsewhere. If our histological sections are of an appropriate width, the slice model can be selected to produce output that is directly comparable to observed histological results in central sections, where the assumption that adjacent cells derive from the same collection of lineages is justified. 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>7</sup>.

If the retina can be usefully thought of as a series of slice models, and the activity of the CMZ is basically homogenous around its circumference, it follows that the activity of the CMZ can 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 is 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. However, if the CMZ homogeneity premise is valid, this can easily be incorporated into the model by providing a layer of "ghost" cells on either side of slice whose parameters are determined by the slice itself<sup>8</sup>. That is, given the premises, a slice model which interacts with copies of itself is a complete model of CMZ-driven retinogenesis.

All this said, the zebrafish retina is a manifestly asymmetrical structure, with an optic nerve posi-

---

<sup>6</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>7</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.

<sup>8</sup>This can be implemented in CHASTE with that framework's "ghost node" objects.

tioned 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. 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 of, say, ranking causal influences on the proliferative and specificative behaviours of RPCs in the dorsal extremity of the CMZ, this could provide significant insight into the regulation of peripheral stem cells in other species. In this sense, a slice model could still be valuable even if it fails to explain aspects of tissue-level zebrafish retinogenesis.

### 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 reasonably cover the period of interest. Let us 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 consume too many computational resources to allow for effective application of Monte Carlo techniques, requiring thousands or millions of simulations per call of the model likelihood function.

### 3.4 Bayesian decisionmaking for structuring models under uncertainty

From the foregoing discussion, we can see that 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 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 formulate this as a problem of decision theory, where we 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 productively 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 has been to estimate model evidence over broad, uninformative prior distributions on model parameters. In general, we assume as little as possible about the processes underlying our results. This leads us to begin with the exercise of testing the relative superiority of Normal and Log-Normal models of the data. 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. It is important to note that raw evidence values do not have a direct interpretation: these values depend on the size of the dataset tested<sup>9</sup>. On the other hand,  $\log ZR$  ratios are dimensionless; they arise from dividing  $\log Z$  values from models explaining equivalent amounts of data. The  $\log ZR$  value can be interpreted directly as the number of orders of magnitude in favour of (positive values) or against (negative values) the numerator model in the ratio. We use two methods of estimating  $\log Z$  values, nested sampling, as described above, as well as the Empirical Bayes method of linear regression implemented in `BayesianLinearRegression.jl`. The former provides for an estimate of the error on  $\log Z$  calculations, and therefore allows the calculation of the significance of these results; these are generally presented as standard deviations of significance, following cosmological practice. This value indicates how far from the estimated  $\log ZR$  the hypothesis that the models are equally evidenced lies (i.e. that the evidence ratio is 1., and the  $\log ZR$  is therefore 0.), and can be understood as our estimate of the certainty we have about the selection of one model over the other. This general model-comparison logic is iterated to address most of the important hypotheses we test in the following chapters.

---

<sup>9</sup>That is,  $\log Z$  values take data units. See [Ski12] for an extensive discussion of this issue.

# Chapter 4

## Bayesian periodization of postembryonic CMZ activity

### 4.1 Preliminaries: Calculation of Bayesian evidence supports independent Log-Normal modelling of CMZ parameters

To take up the question of how CMZ RPC activity evolves over time, we have collected observations of a variety of CMZ and retinal parameters derived from histological studies of cryosections of zebrafish eyes harvested over the first year of the animal’s life. We wish to extract as much information as possible about the structure and time-evolution of the CMZ population as a whole, in order to know how to best model its constituent RPCs. The initial approach here will be to estimate parameters of the whole-eye CMZ from sample cryosections, and to use these estimates as the dataset which will inform our selection of appropriate models.

While it remains common to assume that population census data are Normally distributed, and so simply to calculate means and standard deviations as the statistical representation of the underlying population, it has been known for decades [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). We therefore start with the selection of appropriate models for the CMZ- and retina-level population measurements critical to the inferences that follow.

As a first pass at this question, we calculate the likelihood ratio for the hypothesis that the parameter measurements, and the calculated quantities derived from them, are Log-Normally distributed against the one that they are simply Normally distributed. These results are displayed in Table 4.1.

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, calculated as described in Section 11.1.4, 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,

---

<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 INSERT METHODS AUTOREF, lenses are difficult to retain in this histological context.

Table 4.1: 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.

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. As the two calculated quantities will be the primary ones used in our inferences about population-level CMZ dynamics, we are most concerned with these measures; at this point, the Log-Normal distribution looks to be more informative for both.

We have emphasized the danger of relying overmuch on the parameterisation of single most-likely model fits in Chapter 3, which is what the simple likelihood ratios above represent: the joint likelihood of the maximum a posteriori distribution fitted to the measurements taken from each age cohort. 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. Although it is very unlikely that GMC-NS will result in conclusions from the likelihood ratio, this simple test serves to prove the function of the `GMC_NS.jl` package, and demonstrate the inferential logic. Evidence estimates and ratios are presented in Table 4.2.

Table 4.2: Evidence favours Log-Normal models of retinal population parameters

Parameter	$\mathcal{N}$ logZ	Log- $\mathcal{N}$ logZ	logZR	$\sigma$ significance
CMZ Population	$-4953.7 \pm 7.7$	<b><math>-1148.3 \pm 4.5</math></b>	$3805.4 \pm 8.9$	428.4
Estimated Retinal Volume	$-9539.0 \pm 40.0$	<b><math>-2337.5 \pm 9.0</math></b>	$7202.0 \pm 41.0$	176.0

$\mathcal{N}$ : Normal distribution. logZ: 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.

Unsurprisingly, full estimation of the Bayesian evidence for the Normal vs. Log-Normal hypotheses for our calculated parameters produces the same basic story as the rough calculation of likelihood ratio from the single-fit MAP models. There are approximately 3800 orders of magnitude more evidence for the Log-Normal model of interindividual variation in estimated CMZ annulus RPC population over time. This result has greater than 420 standard deviations of significance<sup>2</sup>. However, there are approximately

<sup>2</sup>The typical standard for a "discovery" in particle physics is  $>5\sigma$  [Lyo13]. Assuming Normally distributed error, the

7200 orders of magnitude more evidence for the Log-Normal model for the variability in the cellular retinal volume estimate, with 176 standard deviations of significance. Given the likelihood ratios alone, we might have suspected that the Log-Normal model was more justified for modelling the population data than for the volume data. In fact, with the full estimation of the evidence, it becomes clear that the converse is true, demonstrating how evidence measurements can supply a fuller picture than maximum likelihood methods. In any case, based on these results, we need not have any compunction about modelling population outcomes for these parameters with Log-Normal distributions, as the evidence supplied by our observations plainly supports it.

If between-individual variation in retinal CMZ population and retinal volume are both well-modelled Log-Normally, the question of their independence immediately arises. It seems plausible that the size of the CMZ population would be roughly proportional to the overall volume of the retina, upon 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>3</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 so-called 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 better described by uncorrelated models at all ages, with the exception of 23.0 dpf, where the evidence weakly favours correlation. It is interesting to note 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 4.3. 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 4.1. 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.

From these analyses, we conclude that organismal variability in the CMZ population 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 ("the underlying"). That is, our prior and posterior belief about the relative likelihood of values of the mean of the underlying may be modelled with a Normal distribution, and our beliefs about its variance with a Gamma, such that our joint uncertainty is the product of the two distributions. This is explained in more detail in ???. A useful analytic feature of the Normal-Gamma prior is that the marginal posterior distribution of the mean,

---

probability that the models were, in fact, equally good at 3 standard deviations of significance would be about a tenth of a percent (i.e. .001). At 10 standard deviations the figure is 7.62e-24. Assuming the sample is representative, we have high certainty about these results and no reason to pursue the matter further.

<sup>3</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 4.3: 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.

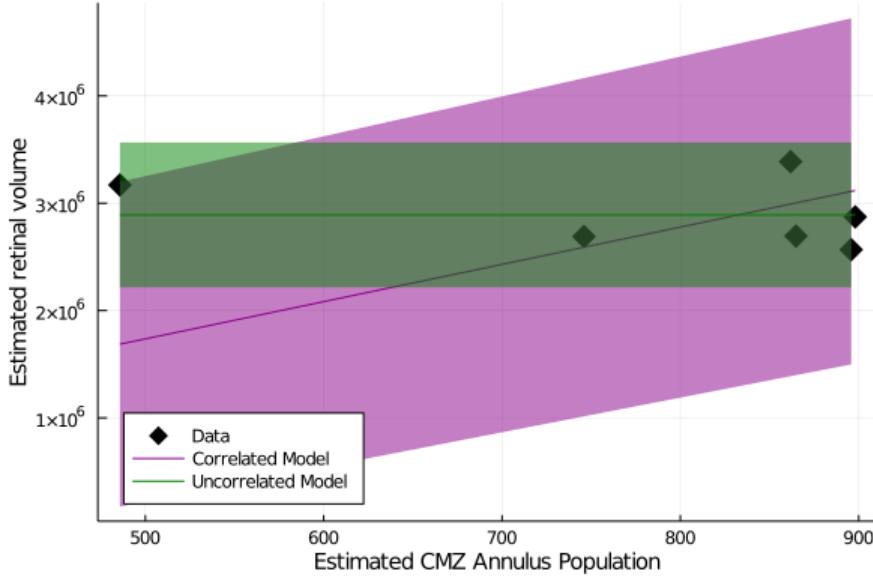


Figure 4.1: CMZ population and retinal volume estimates are uncorrelated at 3dpf  
Individual CMZ population estimate vs retinal volume estimate for 3dpf animals. Uncorrelated and correlated linear models of these variables are plotted as the mean $\pm$ 95% CI of the predictive distribution of the fitted model.

assuming an uninformative (ignorance) prior, is a location-scaled T distribution; this is so because the weighted sum of an infinite series of Normal distributions (i.e. the likelihood-weighted sum of all the Normal distributions that could underly the Log-Normal models), is a T distribution. T distributions are notably more resistent to outlier distortion than Normal distributions themselves are, and may be estimated with as few as two observations, making them highly flexible. Most of the descriptive statistics in the next section, therefore, calculate the 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. With our Log-Normal models selected, and the descriptive tool of the posterior mean T distribution in hand, we turn now to a survey of the zebrafish CMZ in the first year of life.

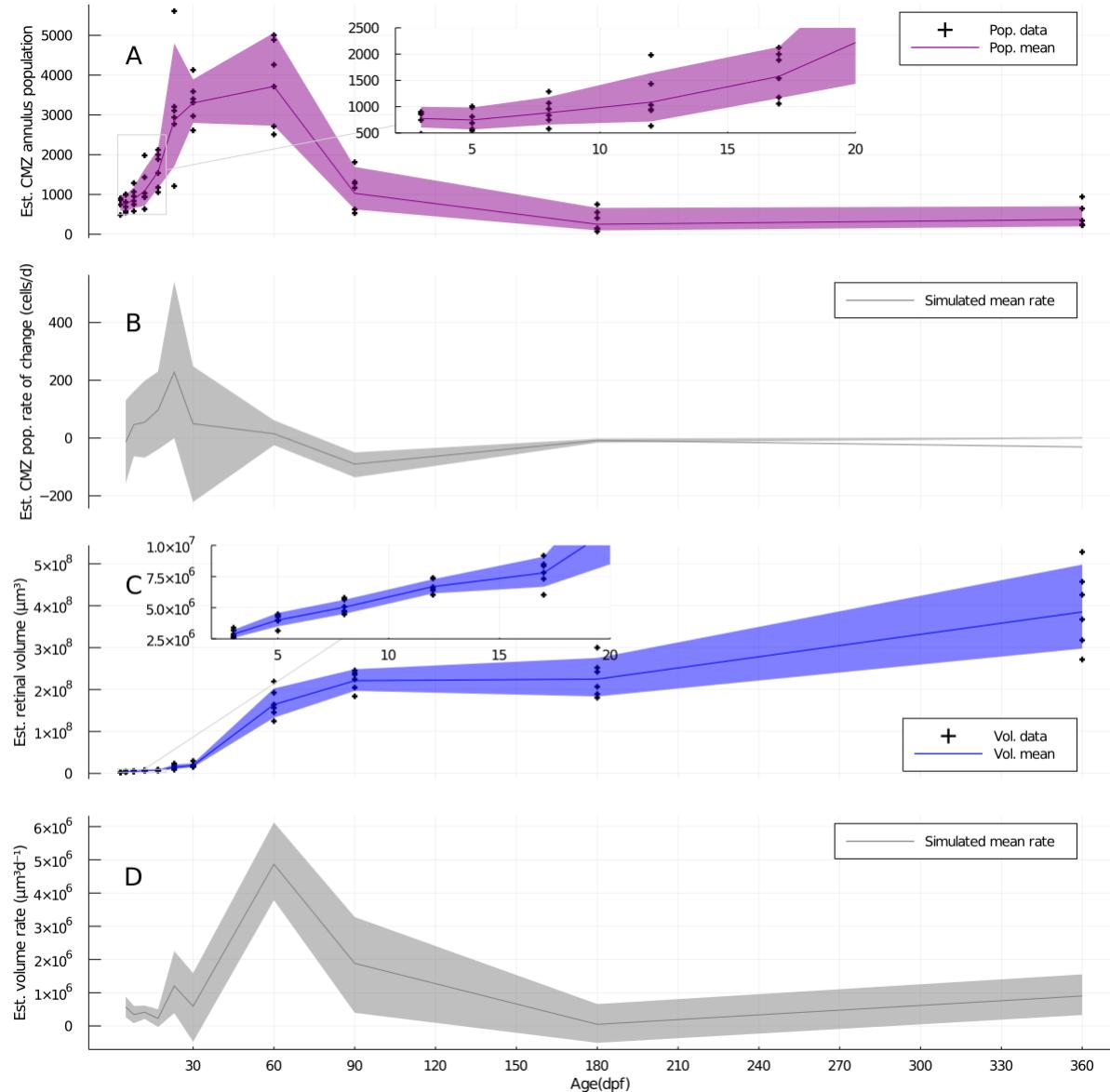
## 4.2 Survey of CMZ population and gross retinal contribution

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? We begin answering this question by presenting our estimates of individual CMZ annulus population and retinal volume over the first year of life, in [Figure 4.2](#), panels A and C.

An initial, relatively quiescent period in the population history of the CMZ can be inferred from the lack of growth observed in the first week of life, as well as the observations in [Figure 5.2](#), which demonstrate that CMZ RPCs are proliferating too slowly to be labelled by a day's pulse of a thymidine analogue at 5dpf in the wild-type and heterozygote siblings of npat mutants. Interestingly, 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, suggesting that this proliferative pause is too short to appear in the retinal volume data.

In any case, the first two to three weeks of life (magnified in lens insets in panels A and C) appear to mark a relatively slow build in both estimated CMZ annulus population and retinal volume compared to the explosion which follows immediately thereafter. While zebrafish are better staged by size than age [[PEM<sup>+</sup>09](#)], the onset of this explosive growth seems to come somewhat earlier than the typical metamorphic transition from larval to juvenile stages at 45dpf [[SH14](#)]. This raises the question of the manner in which 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 output of stem cells, with RPCs specifying at some steady rate, or that the CMZ builds itself up for a wave of niche exit and specification somewhat later, similarly to the sequence of events in the embryonic central retina.

In order to get a better sense of this timing, we simulated the mean daily rate of CMZ annulus population and retinal volume change, by performing Monte Carlo difference operations between samples from the marginal posterior means of subsequent timepoints. These simulated mean rates and their associated confidence intervals are plotted in [Figure 4.2](#) panels B and D. These show the basic time-structure of the phenomenon; the large increase in simulated daily CMZ population growth rate occurs before the large 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 is characterised by much slower population growth and more rapid volumetric growth, implying steady and elevated specification of RPCs. The greater uncertainty associated with our population estimates, relative to their magnitude, results in correspondingly less certainty about the size of the population growth rate spike. For instance, we calculate a >99% probability that the mean retinal



**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  $\pm 95\%$  credible interval for the marginal posterior distribution of the mean.

volumetric rate growth peak at 60 dpf is at least 6-fold greater than the mean at 30dpf. By contrast, only about 97% of the posterior mean density of the population growth rate at the 23d peak ( $230.2 \frac{\text{cells}}{\text{d}}$ ) is greater than the mean at 3dpf, which is a small negative value ( $-5.5 \frac{\text{cells}}{\text{d}}$ ).

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 process thins out the CMZ population to below its size immediately after embryogenesis, 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.

Given this rough description, it seems that 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 bespoke model that captures, for instance, the initial quiescent period, or the post-180dpf growth of the retina. Because this is straightforwardly a question of how much model structure is justified by our data, we may address it as a model selection problem, using the system of Bayesian inference provided by nested sampling, and it is to this we now turn.

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

To perform our model selection task, we wish to simulate the time-evolution of CMZ population and retinal volume. This requires us to supply 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 4.1, we are justified in initializing CMZ population and retinal volume by independent samples from the Log-Normal models of their interindividual variability at 3dpf, at the end of embryogenesis and the beginning of CMZ-driven retinal growth. In order to produce new, simulated values of CMZ population and retinal volume, we apply a system of difference equations as follows, where  $\text{pop}_n$  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  $\mu\text{m}^3$ :

$$\text{pop}_n = \text{pop}_{n-1} \cdot 2^{\frac{24}{CT}} - \text{pop}_{n-1} \cdot \epsilon \quad (4.1)$$

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

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,  $p - 1$  phase transition times.  $\mu_{cv}$ , which is taken to apply equally to all phases, is estimated from 3 dpf nuclear measurements, as described in ???. Given an initial population and volume sampled from the Log-Normal models of their interindividual distributions, Equation 4.1 and Equation 4.2 may be applied to these values difference equations can be applied to produce simulated

sample values at the times actually observed. Many such samples obtained by Monte Carlo can be used to estimate Log-Normal distributions for the model parameters, 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 ???. Using the typical procedures applied in nested sampling [Ski06], we estimated the Bayesian evidence, maximum a posteriori, and marginal posterior distributions on parameters for 2 and 3-phase models.

It is useful to begin with the maximum a posteriori model output, as this plainly shows the primary problem with this simple model; the model relationship between changes in CMZ population and changes in volume breaks down after 30dpf. That is, the later volume estimates are too large for the CMZ to produce, given the calculated cellular volume at 3dpf. The result is that the models fit the early population and volume data quite well, but the population peak is dragged upward and forward to produce more-likely volume output. While it is possible that the later CMZ contributes more volume per neuron to the cellular retina, it is much more likely that the volume approximation applies better to more-nearly spherical eyes at younger ages, than to the flattened eyes of later ages. Although we had hoped that the estimated retinal volume data would constrain the  $\epsilon$  exit rate parameters, this was not the case, as discussed below. 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 a reasonable 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 4.4. There are greater than 5600 orders of magnitude more evidence for the 2-phase model; this result has over 500 standard deviations of significance. This reflects the greatly expanded parameter space in the 3-phase model, which adds an additional 3 parameters to the 2-phase models' count of 5 parameters. 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 therefore conclude that, on the basis of these data, the 2-phase hypothesis must be accepted.

Table 4.4: Evidence favours a 2-phase periodization of CMZ activity

2-phase logZ	3-phase logZ	logZR	$\sigma$ Significance
<b>-5583.1 ± 4.9</b>	-11247.0 ± 9.6	5664.0 ± 11.0	525.425

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.

While the parameter estimates associated with these models are clearly unreliable, they are useful to inspect in order to demonstrate some properties of nested sampling, and for comparison to the simulations to follow. To begin with, we present the parameterization of the MAP models displayed above in Table 4.5. Unsurprisingly, the selected 2-phase model begins with a first phase of rapid proliferation, with a  $CT$  of 13.8 hr, followed by a second, slower phase of 27.2 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 much less, with 86% of

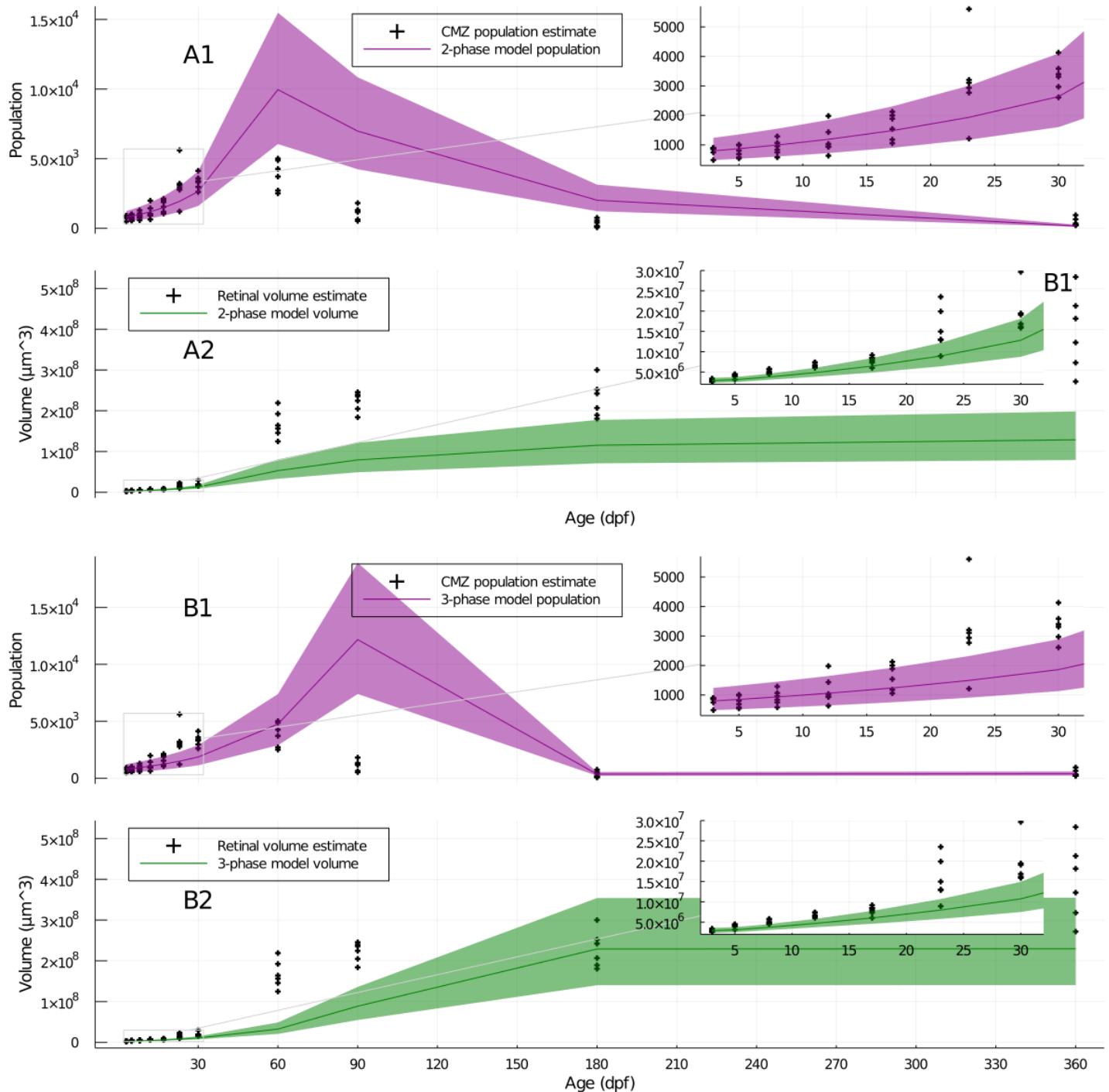


Figure 4.3: Maximum a posteriori output of periodization models

Population and volume estimates from observations (crosses) plotted with mean  $\pm$  95% posterior mass 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.

the day's starting population exiting the CMZ, again suggesting a residency time of about one cycle. The imputed phase transition age is about 62dpf. 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, and the transition age too late; all favoured in order to produce higher volume estimates at later ages.

Table 4.5: Maximum a posteriori parameter estimates for periodization models

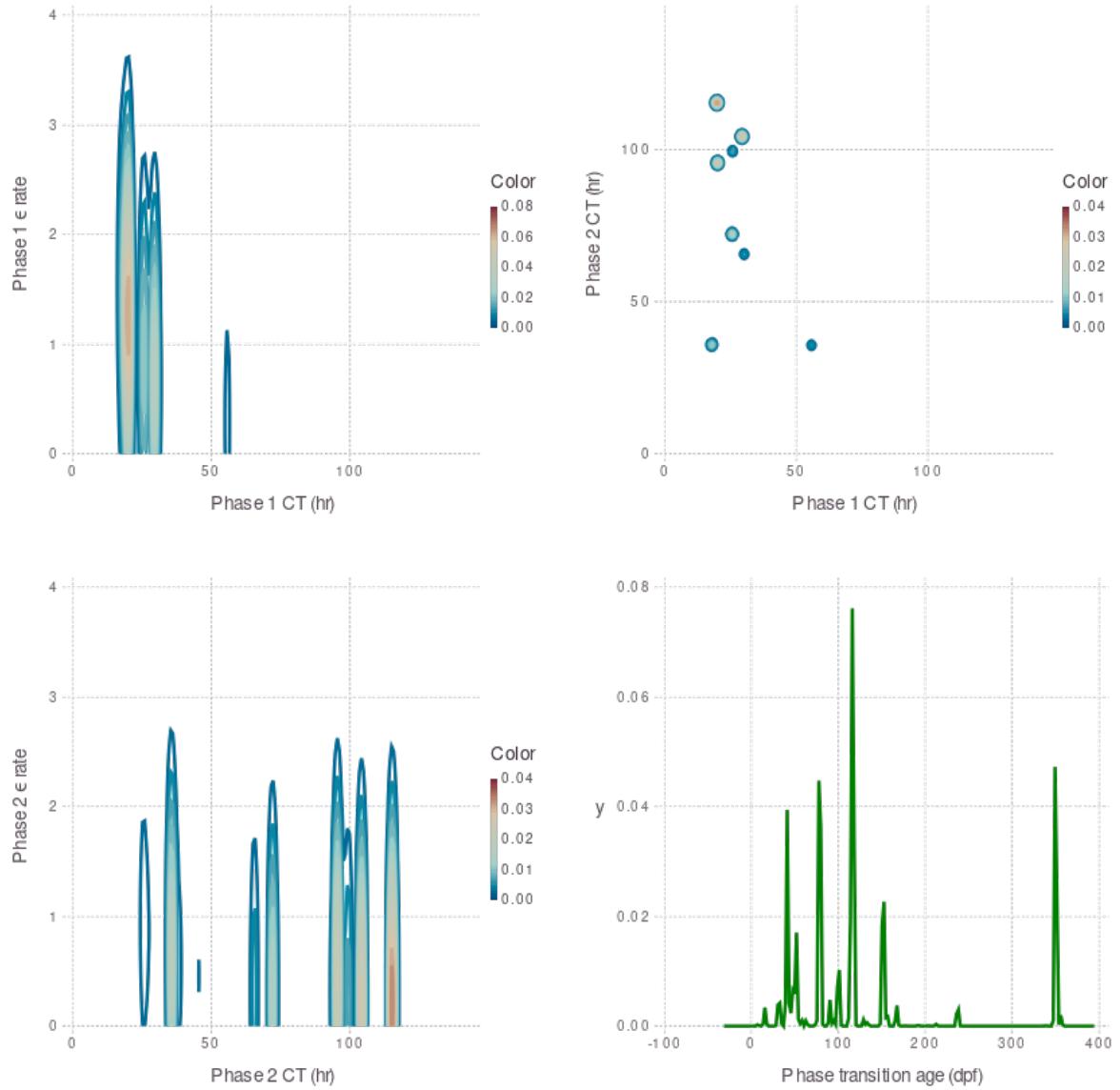
Parameter	2-phase MAP	3-phase MAP
Phase 1 $CT$ (h)	13.8	14.3
Phase 1 $\epsilon$	2.28	2.18
Phase 2 $CT$ (h)	27.2	35.4
Phase 2 $\epsilon$	0.86	0.66
Phase 3 $CT$ (h)	NA	109.1
Phase 3 $\epsilon$	NA	0.12
Transition 1 age	61.9	115.5
Transition 2 age	NA	252.1

Because nested sampling naturally produces samples from the posterior, we can estimate posterior distributions using the evidence values for these samples. We performed this by kernel density estimation, specifically to investigate the extent to which the marginal posterior distributions for the selected model are polymodal; that is, the extent to which 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 plainly reveal the polymodality of the posterior distributions, as well as the lack of constraint that the data impose on phase exit rates  $\epsilon$ . For instance, the first phase parameters (top left) display 3 major modes; the best evidenced mode occurs between 15-20 hr  $CT$ , with the bulk of this posterior mass distributed above an exit rate of 1.0, but with a much larger range of  $\epsilon$  supported than  $CT$ . The second phase parameters (bottom left) display even greater polymodality, with the bulk distributed around 140-150 hr  $CT$ , with an exit rate of less than .75. Plotting the two-dimensional marginal  $CT$  from both phases (top right) reveals that these parameters are the best constrained by the data; still, the posterior is highly polymodal, with numerous combinations of  $CT$  values receiving at least some support, although, obviously, shorter phase 1  $CT$  values combined with longer phase 2  $CT$  values are favoured. The marginal posterior distribution on the age at which the transition between phases occurs is plotted in the bottom right of Figure 4.4; while the largest peaks of the KDE are at times greater than 50dpf, a substantial portion of the prior mass falls earlier than this, significant for the simulations to follow.

It should be noted here that the maximum a priori model parameters are not perfectly reflected in the KDE. While the  $CT$  and  $\epsilon$  parameters for both phases do fall broadly within the best-evidenced posterior modes, they are not centered therein; moreover, the MAP phase transition time does not occur in the best-evidenced transition time mode. As discussed in ??, an important property of nested sampling is that the accuracy of evidence calculations is traded off against the accuracy of estimating the posterior distribution. Since we have here prioritized evidence estimation, this is the cause of this discrepancy; the MAP models themselves have relatively little weight in the KDE estimates.

We conclude that, while our global model of CMZ population and volumetric retinal contribution is too badly flawed to make credible parameter estimates, a 2-phase model of this activity is far better



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

Line height (bottom right panels) or color (other panels) indicates the estimated marginal posterior mass present at the indicated parameter value. Top left panel: Phase 1  $\epsilon$  exit rate vs Phase 1  $CT$  cycle time (hr). Bottom left panel: Phase 2  $\epsilon$  exit rate vs Phase 2  $CT$  cycle time (hr). Top right panel: Phase 2  $CT$  cycle time (hr) vs Phase 1  $CT$  cycle time (hr). Bottom right panel: Estimated marginal posterior mass vs. phase transition age (dpf)

substantiated by the evidence than a 3-phase model. We proceed on this basis, 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.

## 4.4 Slice-model characterisation of asymmetrical CMZ population dynamics demonstrates anatomical homogeneity of proliferative schedule

In the course of the preceding investigations, it became apparent that the 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 Chapter 1. Rather, both the extent and orientation of asymmetry seem to evolve 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 12.2).

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 strongly suggests that the dorsal and ventral CMZ populations 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.

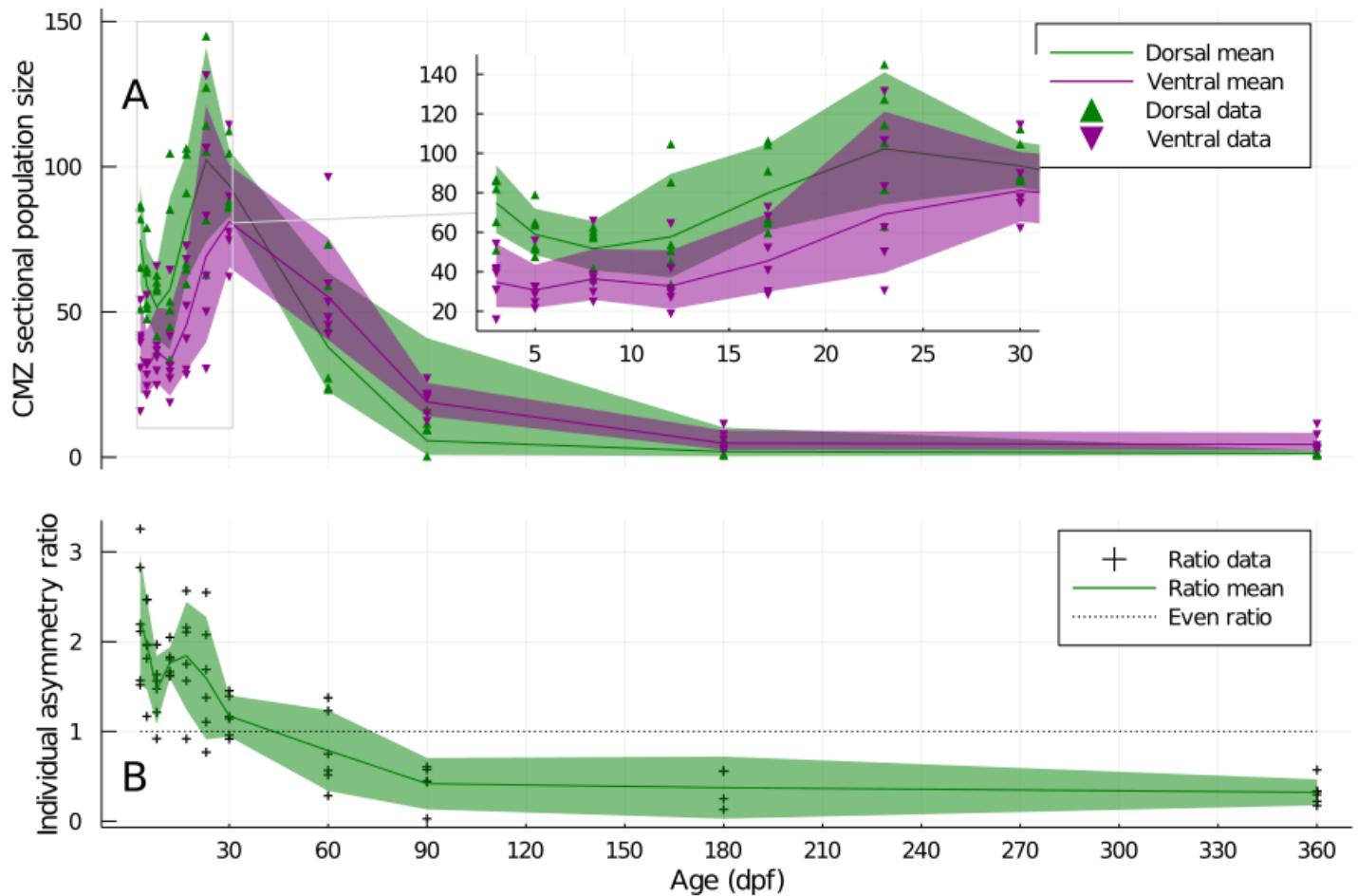
To test this hypothesis, we used a “slice model” of the CMZ, where the thickness of the slice is taken to be the same as the observed cryosection thickness ( $14\text{ }\mu\text{m}$ ). The population of the CMZ is modelled with a difference equation, as above, but with an additional exit term representing 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 implied by a power-law model of lens growth fitted to observations, as 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.

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

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. a combined total sum for both lobes of the slice annulus<sup>4</sup>. In this case, the combination of two phase-shifted populations in the total model should produce a “fuzzy” peak relative to the separate modes. We therefore estimated the evidence, MAP, and posterior marginals for 2-phase models given the sectional sum, dorsal, and ventral populations. Given our belief that the whole-eye model presented above gives an over-long transition time, and in order 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.

---

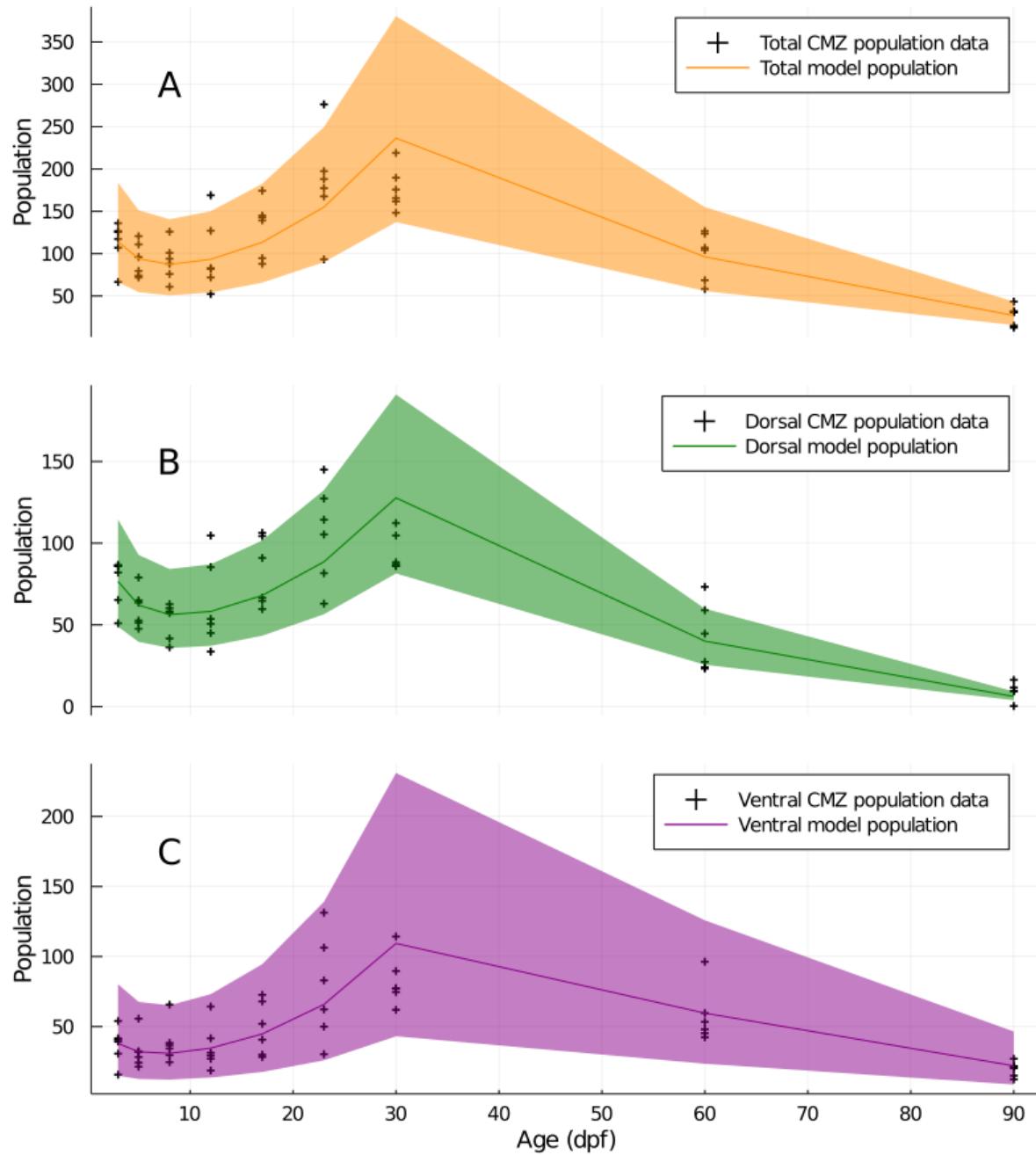
<sup>4</sup>The “total” model is required to supply double the circumferential exit rate of the dorsal or ventral models, to reflect the requirement for this “full” slice to grow both lobes of the eye



**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.

The slice model proves to have much 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; if there is not sufficient evidence to justify a separate, slow, late phase of proliferation, there is clearly none to justify a separate, slow, early phase.

The hypothesis that there is a time-shift in the phase transition across the dorso-ventral axis is thoroughly refuted by these models, in two ways. First, the evidence estimates demonstrate that we are not epistemically 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 190 standard deviations of significance, as displayed in Table 4.6. It is interesting to note that the evidence for the dorsal model is substantially lower than



**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% posterior mass model output, for the total CMZ population model (panel A), dorsal CMZ population model (panel B), and ventral CMZ population model (panel C).

either the total or ventral models. 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.

Table 4.6: Evidence favours a combined slice model over separate dorsal and ventral models

Total logZ	Dorsal logZ	Ventral logZ	logZR	$\sigma$ Significance
<b>-498.1 ± 1.6</b>	-638.3 ± 1.8	-371.7 ± 1.0	512.0 ± 2.6	194.861

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.

A second line of evidence indicating that this hypothesis is unsupported are the MAP model parameter values, summarized in Table 4.7. The MAP phase transition age for the total slice model is functionally identical<sup>5</sup> to that for the ventral model, and within two days of the MAP transition for the dorsal model. Additionally, 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 evidence. Reassuringly, the MAP parameters for the total slice model are similar to those found for the 2-phase MAP in Figure 4.4. Interestingly, the MAP phase parameters differ markedly between the split dorsalventral models and the total slice model, although all suggest a longer  $CT$  and lower  $\epsilon$  for the second phase. This observation suggests that the additional noise incurred from splitting the sectional CMZ population into dorsal and ventral lobes has a strong effect on parameter estimates. This may provide a practical reason to prefer combining these counts in slice models, beyond the fact that it is epistemically unjustified on the support of these data.

Table 4.7: Maximum a posteriori parameter estimates for slice models

Parameter	Total MAP	Dorsal MAP	Ventral MAP
Phase 1 $CT$ (h)	16.1	29.1	20.5
Phase 1 $\epsilon$	1.72	0.69	1.15
Phase 2 $CT$ (h)	23.5	37.0	55.2
Phase 2 $\epsilon$	1.06	0.62	0.38
Transition age	35.5	37.2	35.8

Examining the marginal posterior distributions of the total slice model, presented in Figure 4.7, shows that exit rate posteriors are no less constrained than the whole-eye model, suggesting the retinal volume estimate supplies little additional information, beyond the population estimate, regarding the rate at which RPCs leave the niche. However, the marginal posteriors on cycle length are less constrained than the whole-eye model, with noticeably less separation between modes, which suggests that the volume estimate may nonetheless narrow the range of credible  $CT$  values. This suggests a rationale for pursuing better volumetric estimates for retinae from fish older than 30 dpf. The posterior distribution on the age of phase transition indicates that many values for this parameter retain some credibility, and, like the whole-eye model marginals in Figure 4.4, the MAP estimate does not occupy the most probable KDE kernel for this value, for the same reason noted above.

On the basis of this analysis, the best available hypothesis about the observed ontogeny of RPC population asymmetry across the dorsoventral axis is the pre-establishment of differential population size by asymmetrical progress of the embryonic specification wave, rather than differential proliferative schedules.

<sup>5</sup>That is, both values round to 35 days in the likelihood function, which uses the discrete difference equation, with a step size of 1dpf, as indicated above.

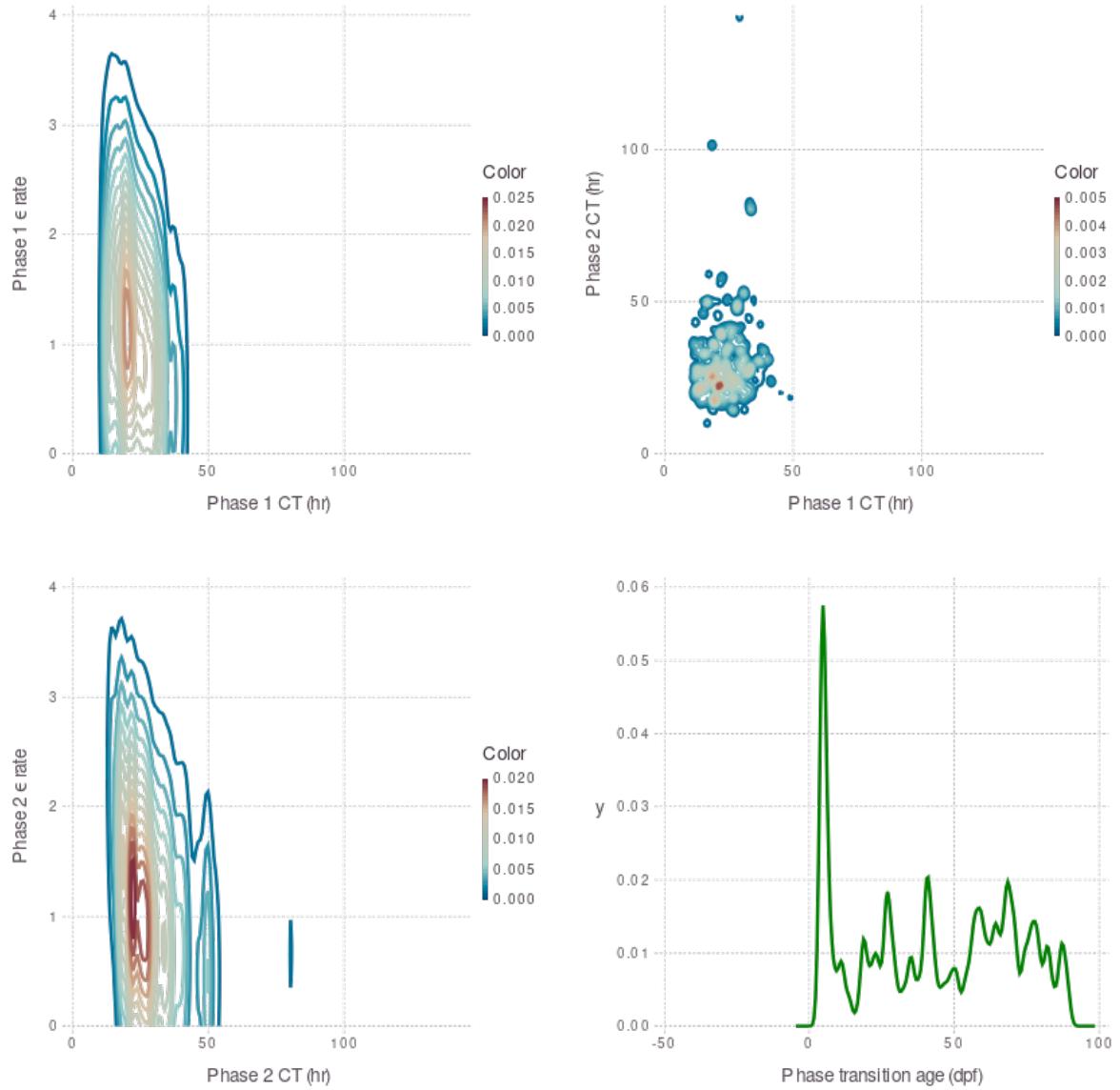


Figure 4.7: Kernel density estimates of marginal posterior parameter distributions, total slice model

Line height (bottom right panels) or color (other panels) indicates the estimated marginal posterior mass present at the indicated parameter value. Top left panel: Phase 1  $\epsilon$  exit rate vs Phase 1  $CT$  cycle time (hr). Bottom left panel: Phase 2  $\epsilon$  exit rate vs Phase 2  $CT$  cycle time (hr). Top right panel: Phase 2  $CT$  cycle time (hr) vs Phase 1  $CT$  cycle time (hr). Bottom right panel: Estimated marginal posterior mass vs. phase transition age (dpf)

#### 4.4.1 Cumulative thymidine labelling supports Galilean Monte Carlo Nested Sampling parameter estimate

The similarity of  $CT$  and  $\epsilon$  exit rate parameters in the whole-eye and total slice models, despite the defective cellular retinal volume estimate, suggests that these values may be relatively accurate. In

order to partially validate this idea, we examined 3dpf CMZ RPCs labelled with a 10.5 hour pulse of the thymidine analogue EdU. 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 described in ??, it can serve to substantiate the first phase cycle time parameter. These results are summarized in Table 4.8, with the relevant linear regressions displayed in Supplementary Figure 12.4.

Table 4.8: Evidence favours whole-CMZ linear cycle models over separate D/V models

Model	Implied $T_c$ (hr)	Implied $T_s$ (hr)	logZ
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. logZ: logarithm of p(D), the marginal likelihood of the data, or model evidence. Largest evidence value bolded.

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). There is no evidence, on this basis, for differing cell cycle characteristics across the D/V retinal axis of asymmetry at 3dpf. Moreover, 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  $TC$ , 13.8 hr, within one standard deviation, while the total slice-model first phase  $TC$ , 16.1, is within two. This confirms that the differing D/V MAP parameter estimates observed in the slice model are unsubstantiated; the evidence supports broadly similar cell cycle characteristics across anatomical axes in 3dpf CMZ RPCs. Moreover, at least the MAP cell cycle times implied by our model selection time seem realistic, given the assumptions of the of Nowakowski model. That said, the data clearly diverge from a linear trend toward the end of the pulse, and the calculated S phase lengths are unrealistically short, showing the limitations of this model.

## 4.5 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 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 are 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.

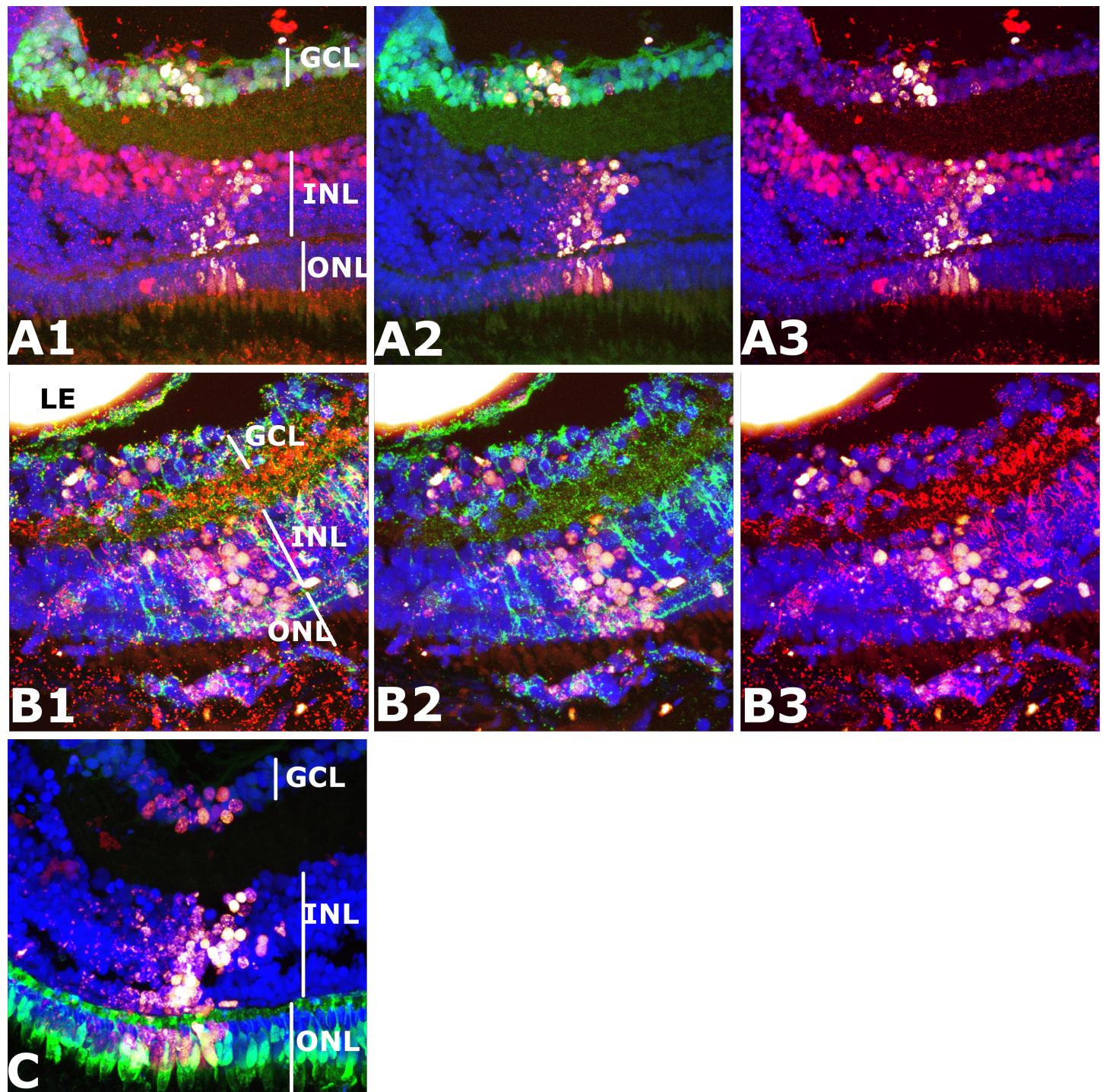


Figure 4.8: **Representative 23dpf lineage marker confocal micrographs**  
Panel A1: RGC/Amacrine staining group. A2: Isl2b channel. A3: Pax6 channel.  
Panel B1: MG/BPC staining group. B2: PKC $\beta$  channel. B3: GS channel.  
Panel C: Double cone staining group. Zpr1 channel.  
GCL: Ganglion cell layer. INL: Inner nuclear layer. ONL: Outer nuclear layer.

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 4.8, while data from all ages are plotted in Figure 4.9. It is worth noting that the relative position of the CMZ-contributed cohort is very different in older animals, with 7 days of chase time being just enough for the majority of the 90dpf cohort to be reliably located within the specified neural retina, as depicted in supplementary ???. Because we suspect that CMZ-contributed retinal cohorts may be subject to a process of attrition, and that this turnover might be higher near the CMZ, as discussed in Section 4.6, if this hypothetical turnover process has differential effects over time on specified retinal neurons of different lineages, results may not be directly comparable between ages. With that caveat, we proceed to the lineage data.

These data take the form of fractions of the presumptive CMZ-contributed, thymidine-labelled cohort entering each of the three cellular layers (panel A of Figure 4.9), 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 immediately 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 12.1. Our evidence supports Normal modelling of all measurements aside from GCL Pax6<sup>6</sup>, PKC $\beta$ -, GS-, and Zpr1-positive cells, along with those displaying horizontal nuclear morphology. We noted that the evidence estimates contradicted simple likelihood ratio tests in most cases, as summarized in Supplementary Section 12.4, with only GCL Pax6+, GS+, and HM+ results in agreement with the evidence estimates. This demonstrates a case where full evidence estimation produces the opposite result from maximum likelihood methods for multiple measurements. 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 4.9.

These calculations speak to overall layer-stability of retinal contributions across this period, with strong evidence for a time-varying model for particular lineages. In particular, we determined that time-varying models of Isl/Pax6+ cells in the GCL, PKC $\beta$ + bipolar cells in the INL, and Zpr1+ double cones of the ONL are all well-supported, with more than 8 standard deviations of significance in each case. Interestingly, the trend of the posterior mean for all three of these lineages declines from the early postembryonic period (3dpf) to the late juvenile period (90dpf). It is tempting to speculate that these lineages may be functionally related; however, we have no specific evidence that would implicate Isl2b/Pax6 double-positive RGCs in circuits with bipolar cells or double cones. Still, this provides a plausible explanation for the observation of a more-ordered retinal mosaic in later retinal contributions relative to the embryonic central remnant: if one or more sublineages in each layer is depleted in older fish, this would result in a different overall mosaic pattern through the layer-structure as the neurons associate and pack together. Based on the overall timing of these changes, we tentatively ascribe this

---

<sup>6</sup>Measured Pax6-positive fractional lineage contributions to the INL support Normal models better than Log-Normal.

Table 4.9: Evidence supports stable layer contributions with time-varying lineage contributions

Layer	Marker	Cell type	Stable logZ	Time-vary logZ	logZR	$\sigma$ sign.
GCL	Cohort	All GCL cells	<b>-41.64 ± 0.63</b>	-57.84 ± 0.34	16.2 ± 0.72	22.5
GCL	Isl2b	RGC	<b>-47.58 ± 0.84</b>	-65.6 ± 0.93	18.0 ± 1.3	14.4
GCL	Pax6	Displaced am.	<b>-32.65 ± 0.68</b>	-42.5 ± 1.3	9.9 ± 1.5	6.6
GCL	Isl2b/Pax6	RGC subtype	-40.94 ± 0.77	<b>-10.53 ± 0.26</b>	-30.41 ± 0.81	37.5
INL	Cohort	All INL cells	-83.5 ± 1.0	<b>-72.97 ± 0.83</b>	-10.5 ± 1.3	8.0
INL	Pax6	Amacrine cell	<b>-32.65 ± 0.68</b>	-42.5 ± 1.3	9.9 ± 1.5	6.6
INL	PKC $\beta$	Bipolar cell	-10.52 ± 0.46	<b>4.61 ± 0.49</b>	-15.13 ± 0.68	22.4
INL	GS	Müller glia	<b>13.69 ± 0.36</b>	3.98 ± 0.7	9.7 ± 0.79	12.3
INL	HM	Horizontal cell	<b>9.81 ± 0.37</b>	-1.18 ± 0.67	10.99 ± 0.77	14.3
ONL	Cohort	All ONL cells	<b>-73.06 ± 0.84</b>	-116.56 ± 0.88	43.5 ± 1.2	35.7
ONL	Zpr1	Double cones	-41.17 ± 0.75	<b>-29.28 ± 0.54</b>	-11.89 ± 0.92	12.9

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.

differential lineage production to the second phase of CMZ contribution in our periodisation; the data do not provide enough information to test evidence for testing timing hypotheses with more precision.

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 12.2.

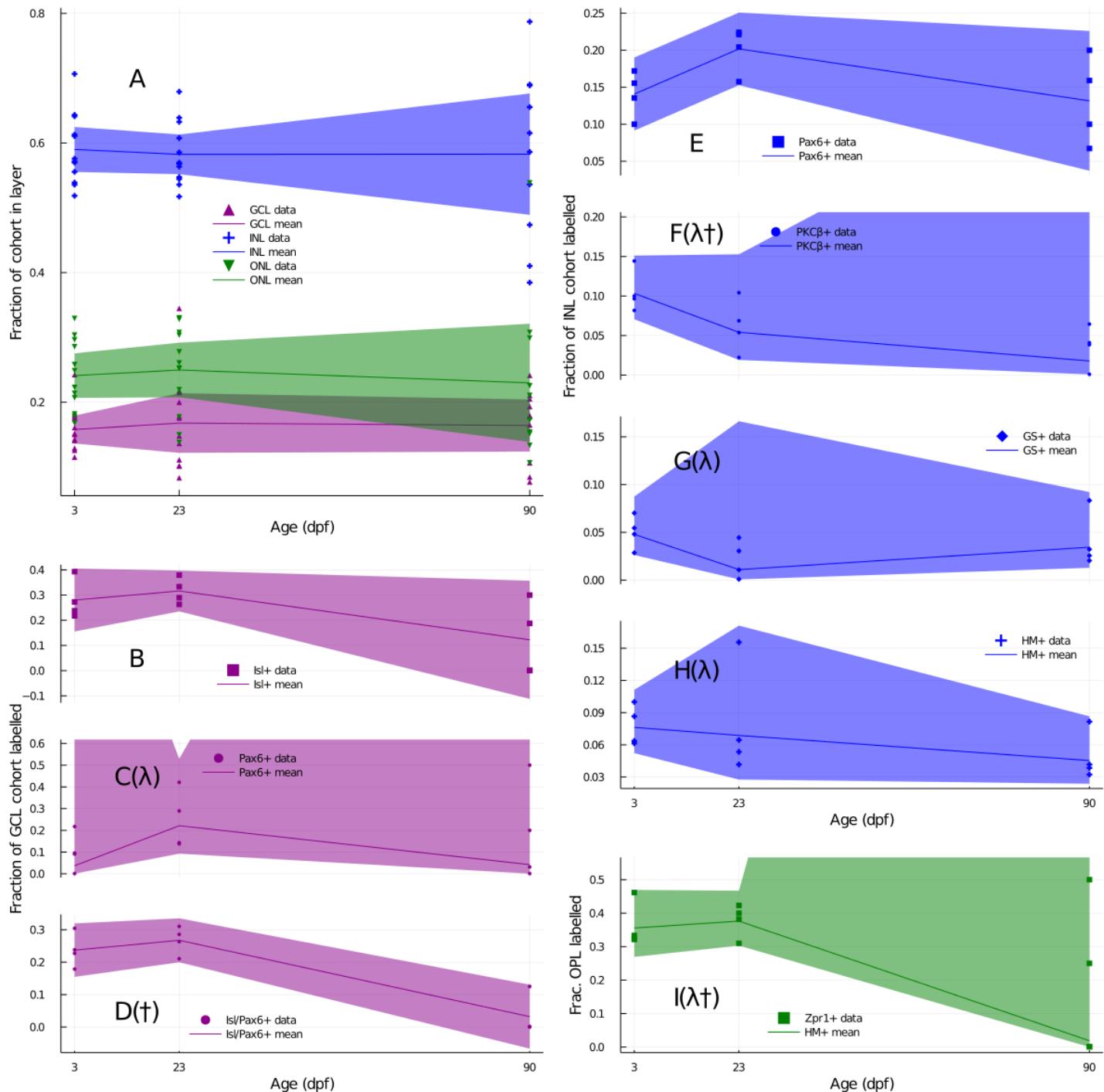


Figure 4.9: Second-phase declines in CMZ-contributed IslPax6+ RGCs, PKC $\beta$ + bipolar neurons, and Zpr1+ double cones

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

## 4.6 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, described as a "neurodegenerative pathology" and suggested as a model of age-related neurodegeneration [VGV<sup>+</sup>18]. In the course of our thymidine analogue pulse-chase studies, we noted that it often appeared that CMZ-contributed cohorts were less numerous noticeably 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 ??, which raised the possibility that these cells may be involved in pruning CMZ contributions. If neural retinal turnover is a general phenomenon throughout the life of the organism, this has fundamental implications for the view that this phenomenon should be treated as pathological, rather than constitutive. However, the thinning phenomenon could be explained by processes involved in the changing morphology and geometry of the neural retina during this period. In particular, the neural retina thickens noticeably over this time period, as displayed in Supplementary Figure 12.1. 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.

In order to investigate this phenomenon, 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 animals at 30, 60, and 90 dpf, we sampled these cohorts from both morphological axes of the retina and counted labelled sectional totals. Data from the axes were combined 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. In effect, we suppose that if the addition of a time-dependent term in the correlated regression model is justified by the evidence at these early ages, this supplies some support to the notion that *D. rerio* retinal turnover is a lifelong phenomenon; if not, the apparent contraction of the cohorts is likely a function of one or more of the alternative morphological explanations noted above, or is occurring at too low a rate to be detected by these means. The results are summarized in Table 4.10, with the regressions plotted in Supplementary Figure 12.3.

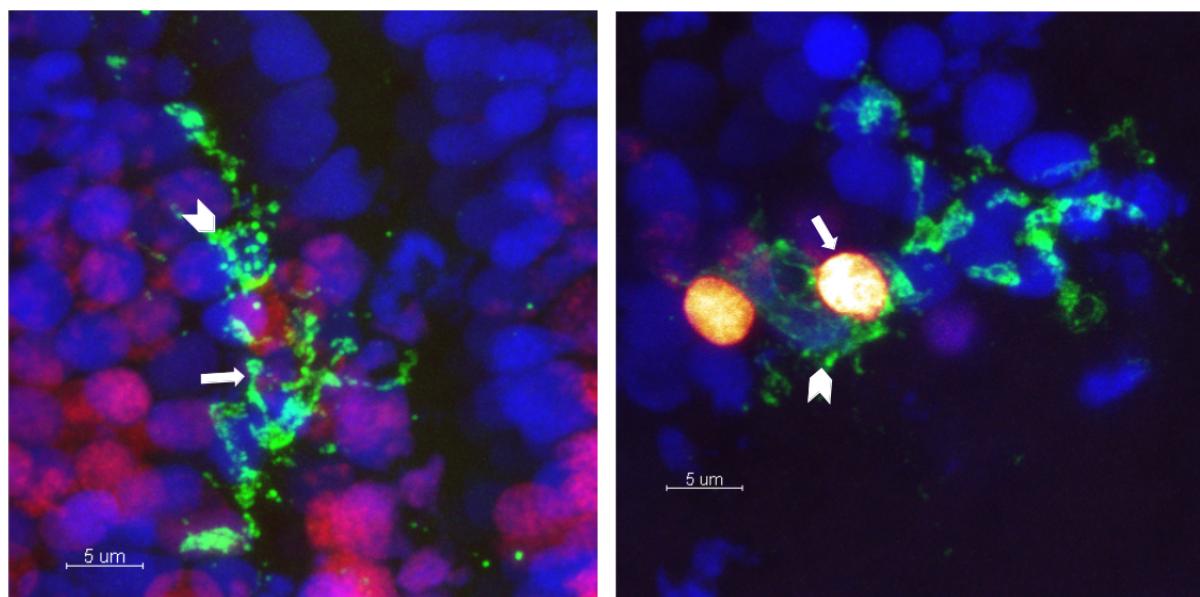
Table 4.10: Evidence for linear regression models supports early cohort population stability

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.

In both cases, there is better 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 12.5.

Moreover, we also observed TUNEL-positive nuclei in the central retina of *rys* siblings in the early postembryonic period (Figure 5.6). This indicates that some level of turnover is indeed occurring during this earlier period. The evidence ratio in favour of the uncorrelated models is not overwhelming, which suggests that the best interpretation of the data is 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 followed with estimating 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 proves to resolve any ambiguity: no time-varying model is justified by the available evidence, as summarized in Table 4.11



**Figure 4.10: 4C4+ microglia associate with and engulf EdU-labelled CMZ contributions in the specified neural retina**

Representative maximum intensity projections from confocal micrographs of 14 $\mu\text{m}$  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.

**Table 4.11: 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>-314.6 ± 3.6</b>	-413.4 ± 5.1	98.8 ± 6.2	15.8
1 month CMZ	<b>-312.3 ± 5.9</b>	-416.6 ± 4.5	104.3 ± 7.4	14.1

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.

## 4.7 Summary: Two-phase periodisation of postembryonic CMZ activity & functional significance of CMZ activity

We have established that our data is best explained by a 2-phase periodization of postembryonic CMZ activity, with an initial month-long phase of relatively rapid proliferation and lower rate of exit into the specified neural retina, followed by a second phase of slower proliferation and higher exit rate. We have established posterior distributions on likely parameter ranges that convey the uncertainty we have about them; these demonstrate that our estimates of cell cycle lengths are much more certain than of exit rates. The population of the CMZ displays an asymmetric structure which reverses over time; we have demonstrated that this is not due to time-shifting of the two proliferative phases across anatomical axes. The proportional layer composition of CMZ-driven contributions is highly 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 found in all three cellular layers in the second phase, providing 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.

These studies demonstrate that the niche history of the CMZ is not adequately explained by a “frozen” population of RPC progenitors, homeostatically recapitulating the program of development found in embryonic progenitors, as has been repeatedly advanced previously [HP98, WAR<sup>+</sup>16]. Instead, a month-long coordinated buildup in population is followed by a concerted slowing of proliferation and increase in the rate of retinal growth in the second month of life. This suggests that the postembryonic CMZ is under a separate regulatory regime from the embryonic RPCs which contribute the central, larval remnant. This establishes that this period of RPC activity justifies separate modelling and investigation. If it is true that *D. rerio* an initially productive peripheral proliferative phase is followed by a second phase decline into quiescence, this regulatory pattern may be common across vertebrate species, and its mechanistic explanations must form the basis of any attempt to entrain endogenous mammalian peripheral retinal stem cells for regenerative medicine.

We are interested in identifying macromolecular factors involved in the control of proliferation and specification of CMZ RPCs. These analyses suggest that these may be most profitably investigated at different ages. If we take the phase transition times estimated in Section 4.4 to be the most reliable in this chapter, this would suggest that factors implicated in proliferative control would be best identified by assays bracketing the transition time at 35dpf; we would expect the rate of change of macromolecular correlates of cycle activity to be greatest around this time. There seems to be little spatial structure apparent in proliferative activity, and it is probably pointless to break out these data by anatomical location, if experiments are conducted at this time, as the population size asymmetry is smallest at this time.

Overall, the data collected thus far provide the weakest evidentiary support for hypotheses about RPC fate commitment and specification. This emphasizes the sense in which proliferative and specificative activities of RPCs must be treated separately, since separate datasets, more carefully calibrated than the crude anatomical estimates used here, will be required to estimate parameters related to RPC contribution to the retina more precisely. While we were able to demonstrate an overall change in lineage subtype contribution, these pulse-chase data were not able to inform the analysis of niche exit

rate. The use of saturating labelling concentrations and times strongly suggest that the population of the EdU-labelled cohorts in these lineage tracing experiments should be directly proportional to the population of proliferative cells at the time of contribution; given the long chase times, assuring that enough generations have passed to dilute the label in the niche to nil, nothing can be gleaned from the EdU population size count alone other than a proxy of the size of the CMZ population at the time of labelling. The use of multiple, separately labelled thymidine analogues administered in pulses separated by a day could be a cost-effective way to inform models of niche exit and retinal contribution of the later postembryonic CMZ. Ultimately, useful explanations of retinal lineage contribution are likely to require more explicit spatial representation in order to incorporate physical effects, signalling hypotheses, and the like.

In addition to discovering some important senses in which postembryonic CMZ activity differs from embryonic central retinogenesis, we have proved out a general logic for Bayesian evaluation and selection of arbitrary biological models, which can inform both future experiments as well as modelling and theoretical choices. For instance, the slice model proves to be a more successful way to combine population and morphological information to answer questions about CMZ population structure, than are abstract whole-population models of the CMZ. Given the relative ease of producing datasets for such models, we would likely do well to establish a multiple-label thymidine analogue experimental paradigm that uses this model form for analysis. 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.

# Chapter 5

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

### 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 mechanisms regulating 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 [RBPP06], 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.

Mapping revealed the causative *rys* mutation lay 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 have 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 clusters [ZKL<sup>+</sup>00] in association with histone nuclear factor P (HiNF-P) [MXM<sup>+</sup>03]. The protein's effects

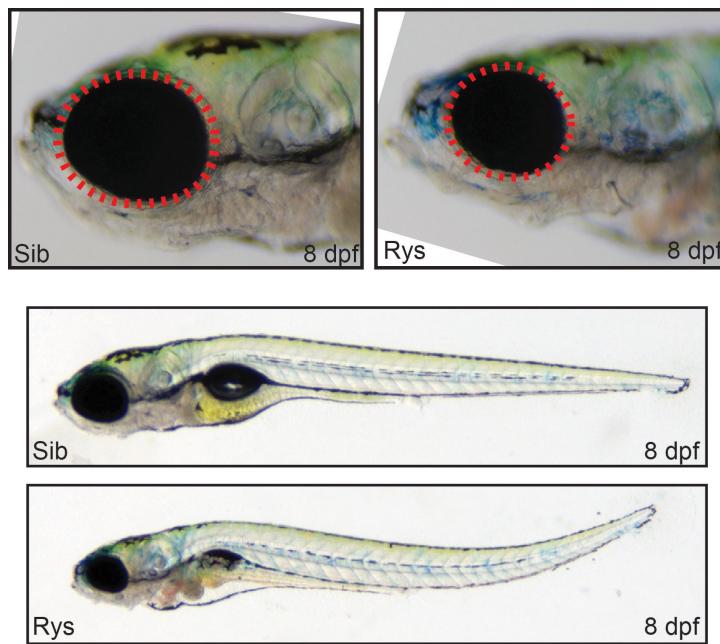


Figure 5.1: *rys* mutants exhibit a small-eye phenotype

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

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 known to be a component of the E2F transcriptional program [GBB<sup>+</sup>03] and a substrate of cyclin E/CDK2 [ZDI<sup>+</sup>98], although E2F-independent activation by cyclin D2/CDK4 in human ES cells has also been described [BGL<sup>+</sup>10]. The expression of NPAT protein peaks at the G1/S boundary [ZDI<sup>+</sup>98], as does its phosphorylation, which promotes its transcriptional activation of replication-dependent H2B [Ma00] and H4 [MGv<sup>+</sup>09] 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>+</sup>09]. NPAT and HiNF-P have also been found associated with the U7 snRNP complexes that perform this function [GDL<sup>+</sup>09]. 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 have been conducted in tissue culture contexts, 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>+</sup>97]. The availability of *rys*, 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 the normal proliferation and differentiation 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 *rys* 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 *rys* CMZ phenotype is characterised failure of RPCs to specify, altered nuclear morphology, aberrant proliferation and expanded early progenitor identity

*rys* has previously been described as having an enlarged CMZ [WSM<sup>+05</sup>], but whether this is a consequence of an enlarged proliferative population, altered cellular morphology, or other causes, remained unclear. We sought to learn more about the ontogeny of the mutant niche by examining two histochemical markers of cycle activity during the early life of *rys*; 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 24 hour pulse, which indelibly marks cells that have passed through S-phase whilst exposed to it. These data are displayed in Figure 5.2. During this period, the PCNA positive population of the sibling CMZ declines precipitously (Panel A), with a corresponding decrease in proliferative activity as measured by the incorporation of EdU (Panel B). Whether or not the *rys* CMZ population is enlarged by comparison depends on the age at which it is sampled, with there is a 99.6% probability of the posterior mean sectional *rys* CMZ PCNA population being below the sib mean at 4dpf, while 99.8% of the marginal posterior of the 10dpf *rys* mean lies above that of sibs. Therefore, the *rys* 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>.

---

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

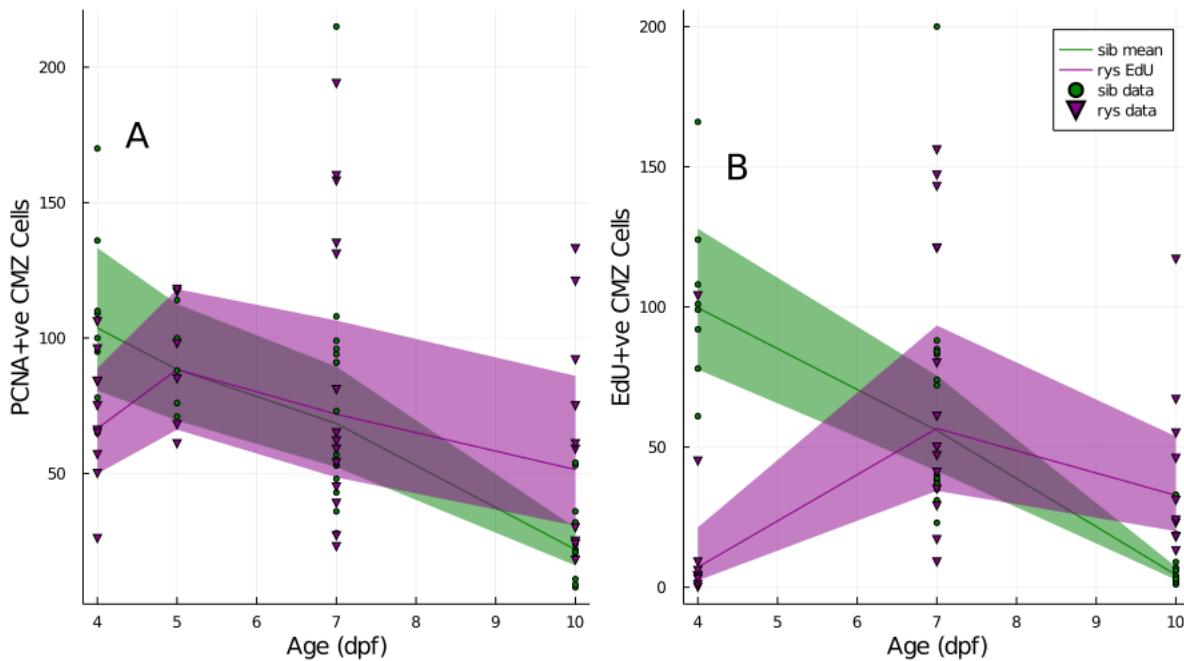


Figure 5.2: *rjs* CMZ populations start relatively small and quiescent, end abberantly large and proliferative

Panel A: Counts and estimated mean  $\pm 95\%$  credible intervals of PCNA-positive cells in the peripheral CMZ of *rjs* (magenta) and their siblings (green). Panel B: As above, but for counts of double PCNA-, EdU-positive cells in the CMZ after a 24 hour pulse of EdU.

Surprisingly, very few *rjs* animals have many actively cycling RPCs at 4dpf, by comparison to the robustly cycling sib RPCs at this age; a mean estimate of  $96.2 \pm 3.4\%$  of sib RPCs are labelled during the 24 hr pulse at 4dpf, while only  $24.1 \pm 37.2\%$  of *rjs* 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 *rjs* RPCs. While we questioned whether this late-stage increase in *rjs* thymidine analogue labelling might not represent bona fide mitotic activity, we were able to readily find mitotic figures within these populations, shown in Supplementary Figure 13.3. As the data convey, these outcomes are highly variable in both sib and *rjs* animals, with, for instance, an isolated mutant at 4dpf sporting an EdU-positive population near the sibling mean. However, even in those *rjs* animals which do have actively proliferating RPCs at these earlier ages, there is a marked failure of the CMZ to contribute to the postmitotic, specified neural retina. Confocal micrographs displaying *rjs* CMZ cohorts labelled with BrdU at 3dpf that have failed to enter the neural retina after 7 days of chase time are displayed alongside their normal siblings in Figure 5.3.

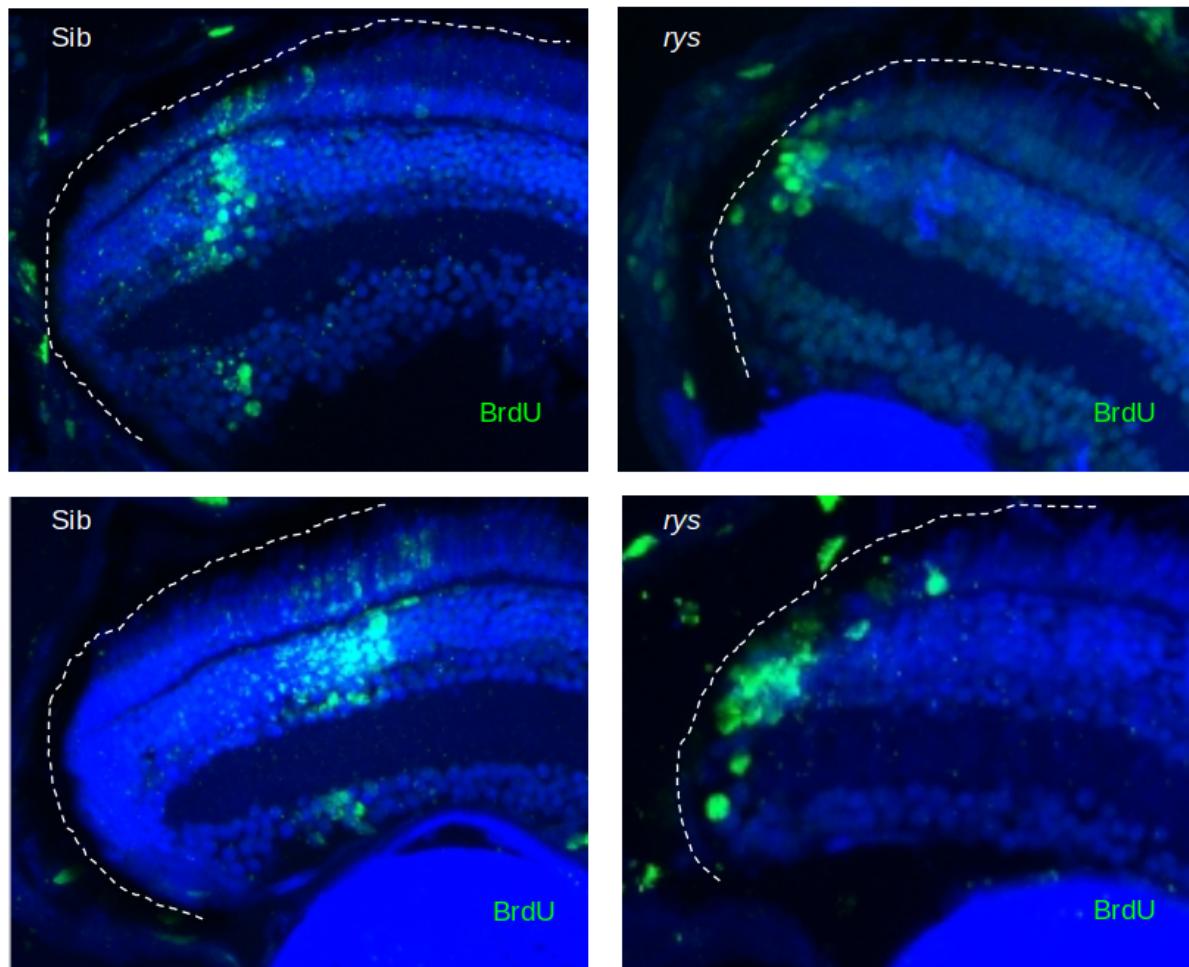


Figure 5.3: *rys* CMZ RPCs fail to contribute to the neural retina

M 14 $\mu$ m coronal cryosections through representative sib (left panels) and *rys* eyes at 10dpf, 7 days after an 8hr BrdU pulse at 3dpf. Note that few labelled *rys* cells have entered the specified retinal layers.

Indeed, a close study of the 5dpf retinae, held back from EdU processing to preserve their nuclear features (presented in Figure 5.4), suggests that the primary reason for the enlarged appearance of *rys* CMZs, even at this age, when the niche's population is numerically similar to siblings (Panel A), is this failure to contribute to the neural retina, leading to *rys* central retinae that are about half as populous, per cell of the CMZ, as their siblings (Panel B). This appearance may be enhanced in the dorsal CMZ, there is an 87.8% probability that the posterior mean for this region in *rys* is above the sibling mean, although this is compensated for by 97.5% of the marginal posterior distribution on the ventral mean laying below the same sibling mean (Panels C and D). More significantly, *rys* nuclei tend to be much larger than their siblings (Panel E), as well as consistently less spherical (Panel F).

In order to quantitatively rank these contributions to the *rys* nuclear phenotype, we calculated the evidence for combined LogNormal (population measurements) and Normal (nuclear measurements) models of sib and *rys* data, against the joint evidence for separate models. These estimates are presented in Table 5.1. Our measurements indicate that the best evidenced contributors to the *rys* CMZ

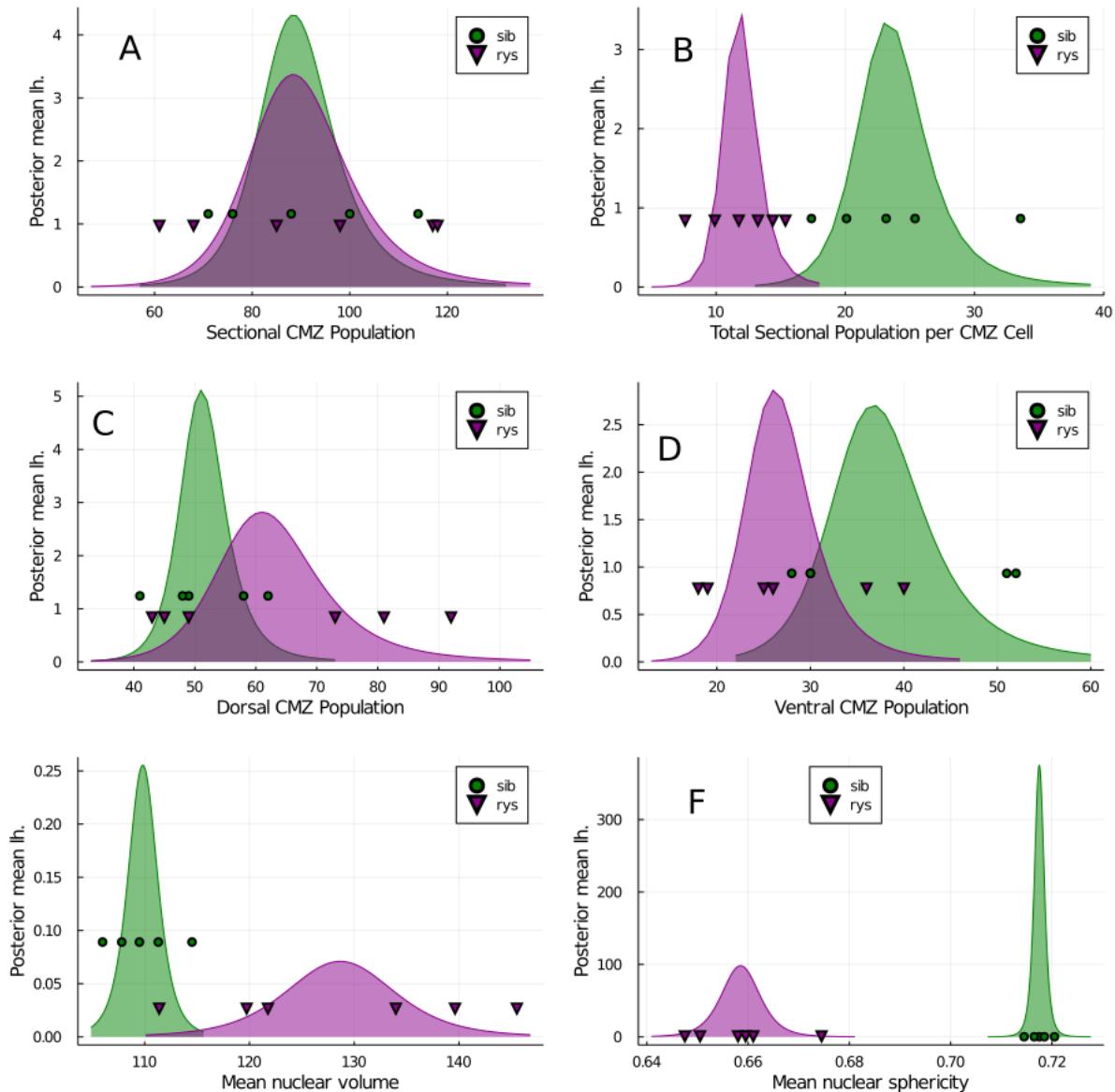


Figure 5.4: 7dpf *rys* CMZ RPCs display enhanced asymmetry, increased volume, decreased sphericity, and relative but not absolute enlargement

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 dorsal + ventral CMZ population per central coronal section. Panel B: Number of PCNA negative, specified central retinal neurons per PCNA positive CMZ cell. Panels C and D: Dorsal and Ventral CMZ 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.

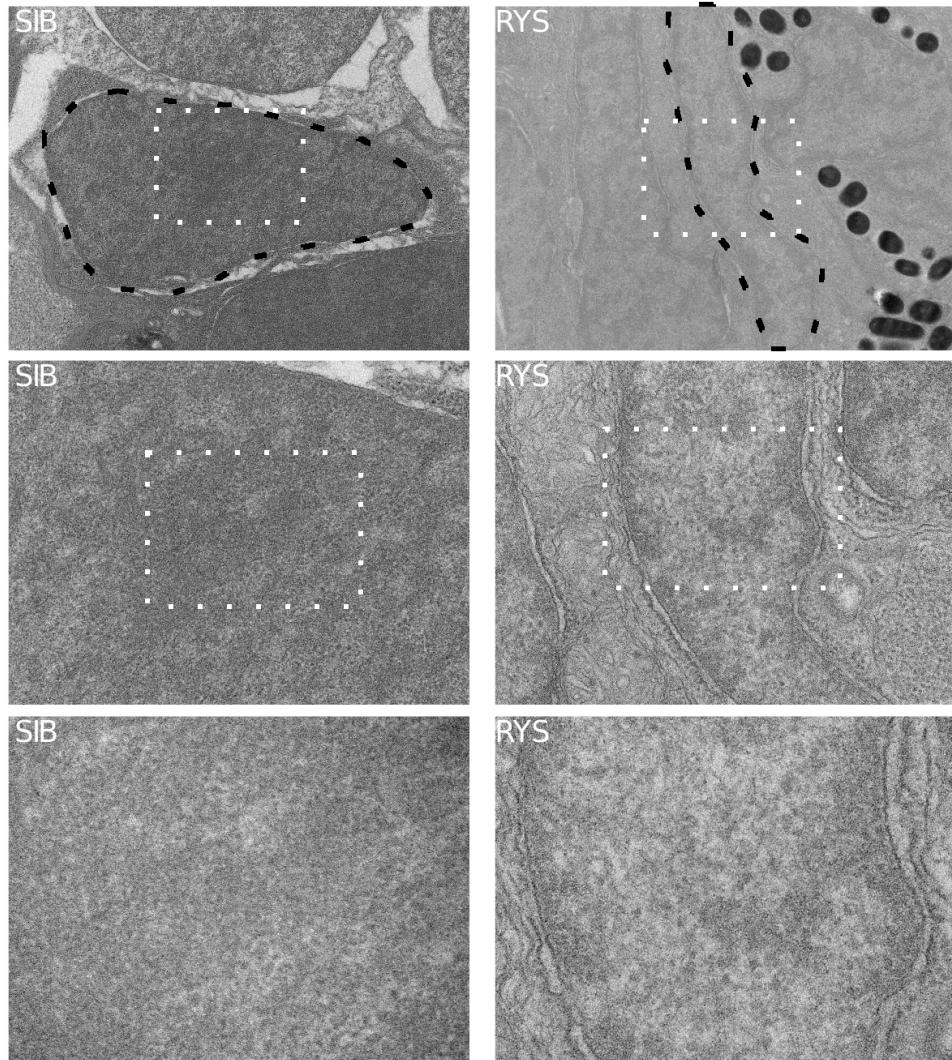
phenotype are, first, the decrease of nuclear sphericity, followed by the decreased number of central

retinal neurons relative to the CMZ, then, the increase in nuclear volume, and lastly the expansion of the dorsal CMZ. These calculations also demonstrate that there is no evidence for overall differences in sectional CMZ population, and that the ventral CMZ is less populous in *rys*, which may contribute to the appearance of an enlarged dorsal CMZ. The large size of the nuclear morphological changes and the decreased number of central cells indicates that these are the most important contributors to the *rys* phenotype.

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

Parameter	Separate logZ	Combined logZ	logZR	$\sigma$ sign.
Sectional CMZ pop.	-280.8 $\pm$ 1.7	<b>-183.44 <math>\pm</math> 0.9</b>	-97.4 $\pm$ 1.9	50.7
Central pop./CMZ cell	<b>-62.51 <math>\pm</math> 0.64</b>	-89.88 $\pm$ 0.12	27.38 $\pm$ 0.66	41.7
Dorsal CMZ pop.	<b>-180.1 <math>\pm</math> 1.0</b>	-192.8 $\pm$ 1.1	12.8 $\pm$ 1.5	8.3
Ventral CMZ pop.	<b>-136.86 <math>\pm</math> 0.32</b>	-147.36 $\pm$ 0.46	10.5 $\pm$ 0.56	18.8
Nuclear Volume	<b>-164.18 <math>\pm</math> 0.29</b>	-229.3 $\pm$ 1.9	65.1 $\pm$ 1.9	34.4
Nuclear Sphericity	<b>-89.1 <math>\pm</math> 1.8</b>	-311.2 $\pm$ 3.9	222.0 $\pm$ 4.2	52.3

The characteristically enlarged nuclei in *rys* have a billowy appearance suggestive of chromosomal disorganization. In order to investigate this possibility further, we used electron microscopy to examine the nuclear ultrastructure of RPC nuclei in *rys* and sibling CMZs. Representative electron micrographs are presented in Figure 5.5. At 36000x magnification, the unusual and disorganized structure of *rys* nuclei become apparent, particularly in contrast with the consistently teardrop-shaped nuclei of sibling RPCs; the *rys* nucleus pictured is so pancaked it extends out of the frame that readily captures a sibling nucleus. When the chromatin itself is imaged at 210000x magnification, it appears much less electron-dense in the *rys*, with much larger tracts of presumptive euchromatin, and less regular spacing of chromosomal material.



**Figure 5.5: 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.

If RPCs in *rys* CMZs are failing to enter the specified neural retina, but by 7dpf are becoming mitotically active, this leaves the question of why this apparently proliferative niche's population is declining by 10dpf. We suspected that *rys* RPCs may be undergoing apoptosis *in situ*. Although we did not detect pyknotic nuclear fragments in our EM investigations, it is possible that the individual apoptotic events are too rare in *rys* to reliably detect in this manner. In order to investigate this possibility, we assayed the presence of caspase-3 in *rys* and sib CMZs. As displayed in Figure 5.6, 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 *rys* RPC nuclear fragments in the CMZ, displayed in Supplementary Figure 13.4.

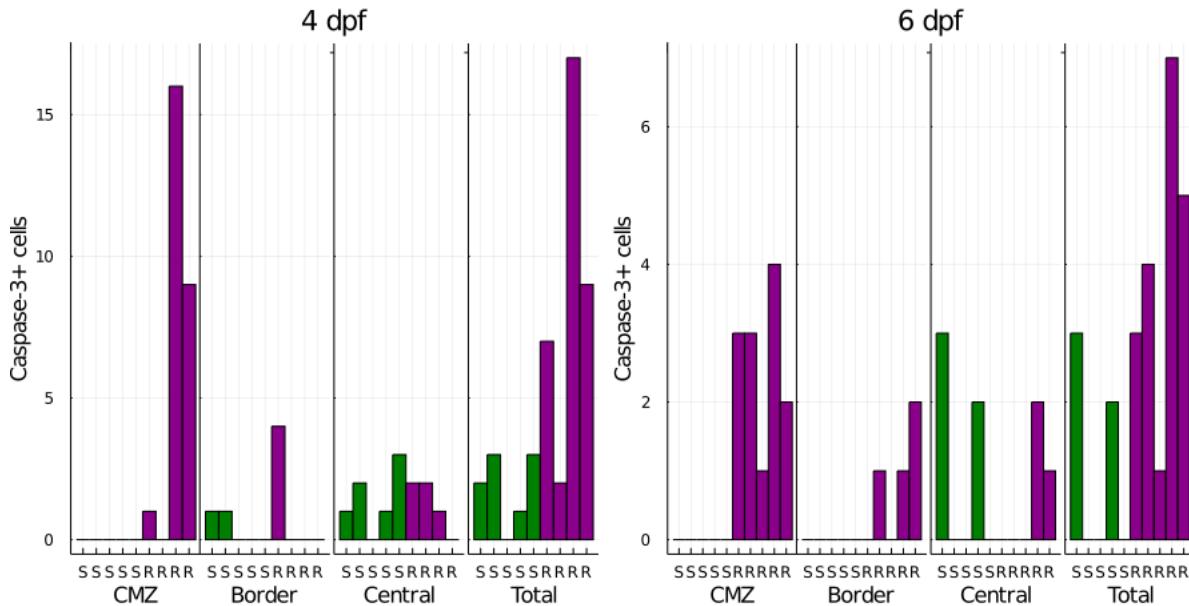
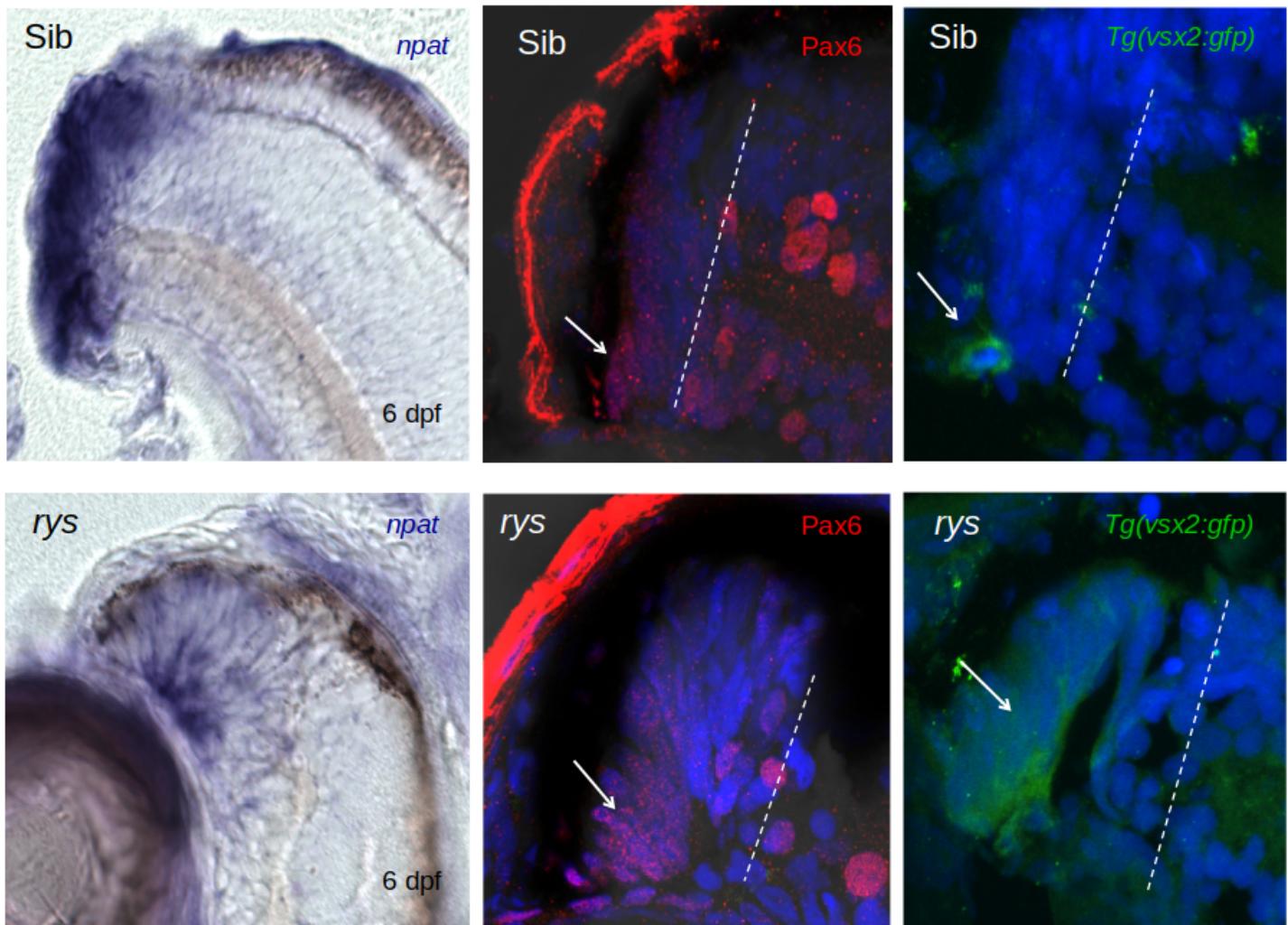


Figure 5.6: *rys* mutant CMZs have increased caspase-3 positive nuclei

Number of caspase-3 positive cells 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  $\mu\text{m}$  section per sibling (S) and *rys* (R) larva.

Suspecting that *rys* CMZ RPCs may maintain an early progenitor identity, causing their failure to contribute to the specified neural retina, 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 [RBBDP06]. The Pax6-stained region was enlarged in *rys* CMZs relative to their siblings (Figure 5.7, center panels). As vsx2 is also known to be a marker of retinal progenitor cells in the CMZ [RBBDP06], we also generated a transgenic vsx2::eGFP *rys* line using a fragment of the zebrafish vsx2 promoter which drives eGFP expression in the utmost retinal periphery in siblings. Mutant fish from this line displayed substantially expanded eGFP expression in the CMZ (Figure 5.7, right panels).



**Figure 5.7: 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 on 20 $\mu$ m coronal cryosection. Middle panels: anti-Pax6 immunohistochemistry. Right panels: GFP expression in a Tg(vsx2:GFP) line introgressed into *rys*.

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

In order to determine the causative mutation responsible for the *rys* phenotype, we performed linkage mapping to identify candidate genes, followed by PCR analysis of the transcript products of these candidates. 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 11, in 6dpf *rys* mutants, shown in Figure 5.8. The predicted protein sequence expressed from the mutant transcript 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 5.9.

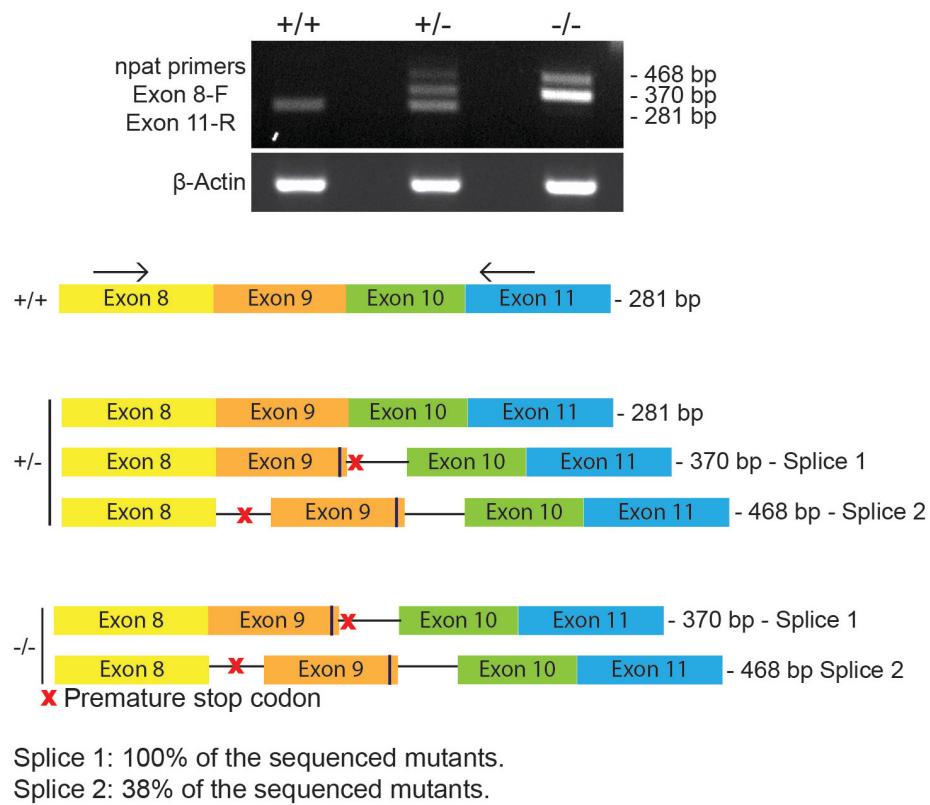


Figure 5.8: RT-PCR analysis reveals two aberrant intron retention variants in *r ys* mutant npat transcripts

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.

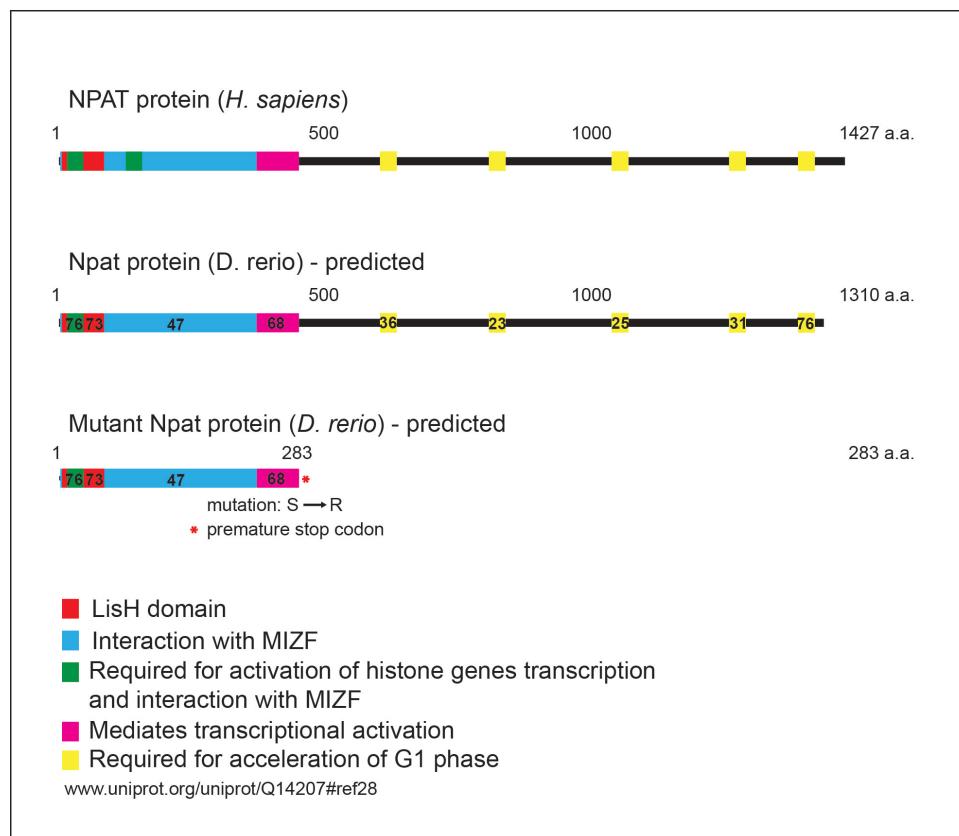
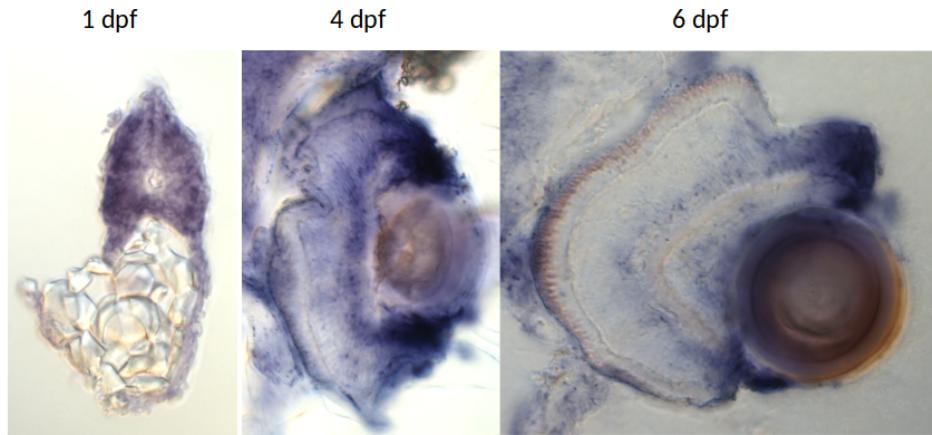


Figure 5.9: Functional domains of Human NPAT compared to predicted wild-type and *r ys* *Danio* npat

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 duplicates; plausibly, a duplication in zebrafish npat could explain the lessened severity of the mutant phenotype when compared to mammalian proviral inactivators. The results of this analysis are presented in Supplementary Figure 13.1. While npat is in the midst of a region which appears to be duplicated on chromosomes 5 and 15, relative to the unduplicated homologous synteny run on *H. sapiens* chromosome 11, it is not, itself, duplicated.

We performed in situ hybridisation using a probe directed to the wild type 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 5.10, focusing on the retina at the times when it has formed. While 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 we might expect on the basis of its cell cycle functions.

Having taken note of the unusual chromatin ultrastructure present in *r ys* CMZ RPCs, and with a mutant npat allele reliably linked to the appearance of the *r ys* phenotype, we investigated the transcriptional status of npat and its histone regulatory targets in these animals. We first assayed npat itself by RT-PCR, finding that homozygous *r ys* mutants overtranscribe npat by about 3-fold compared to their wild-type counterparts at both 6dpf and 8dpf, while sibling overabundance declines from about 2.5-fold



**Figure 5.10: In situ hybridization reveals progressive restriction of npat expression to the CMZ**

20µm sections of wild-type embryo (1dpf) and retinae (4 and 6 dpf), displaying progressive restriction of npat expression as assayed by in-situ hybridisation.

to 1.5-fold over this time period, as shown in Figure 13.2, with calculated marginal posterior mean mass over the WT standard given in ??.

Since mammalian NPAT is known to regulate histone transcription and is critical for coordinating the correct expression of replication-dependent histone transcripts required to package genomic DNA during S-phase [ZKL<sup>+</sup>00], we hypothesized that the altered nuclear morphology and truncated S-phase we observed in proliferating *rys* CMZ cells may be a consequence of perturbed regulation of 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 all members of the 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 the above-described qPCR assays on oligo-dT-primed cDNAs, we were able to examine the population of polyadenylated histone transcripts in isolation.

To plausibly be implicated in the *rys* phenotype, a candidate histone transcript ought to be overexpressed in mutants relative to both WT and siblings at 6dpf and 8dpf. In order to determine the best-supported candidates for causative involvement in *rys*, we calculated the standard deviations of significance which obtain on the test of mean *rys* transcript being greater than either mean sibling transcript or the WT standard (set to 1.0). These values are presented in Table 5.2.

Following these calculations, we find the greatest significance at 6dpf for the *rys* overexpression of total H2B transcript, followed by polyadenylated H2A, total H2A, and polyadenylated H2B. The ordering for 8dpf is similar: the most significant overexpression results are for polyadenylated H2B, then total H2A, polyadenylated H2A, and total H2B. 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.

<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.

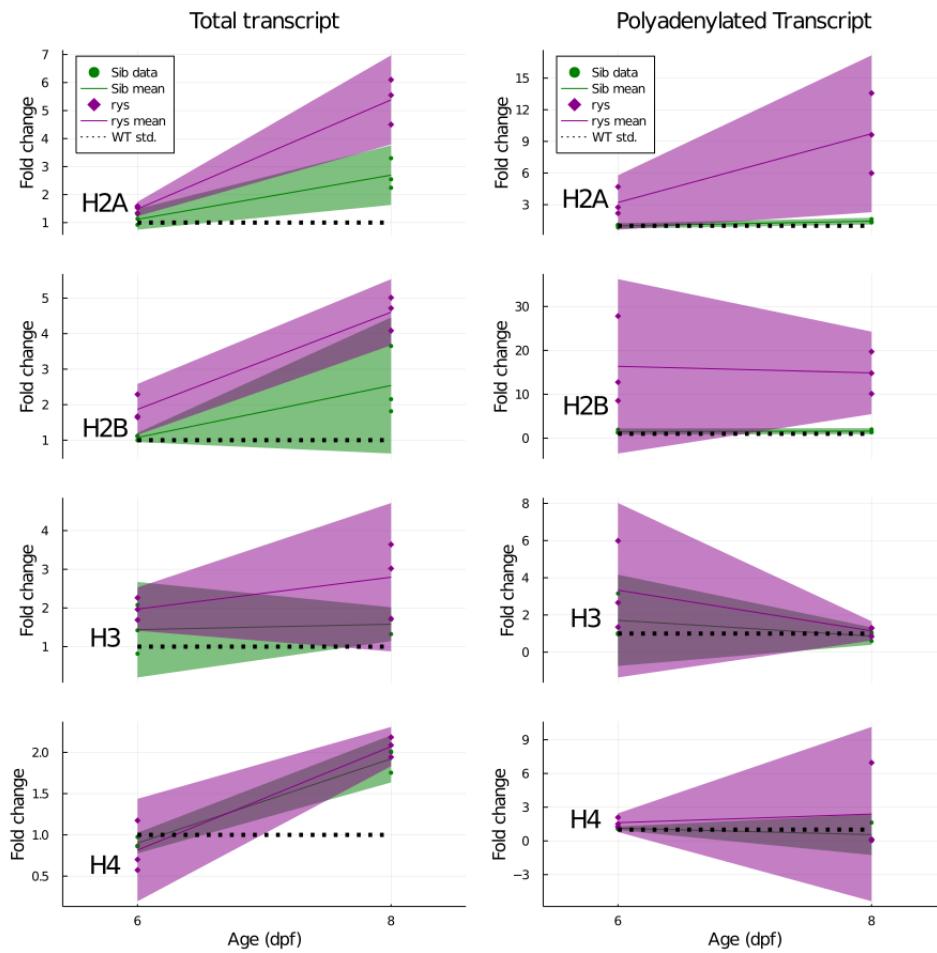


Figure 5.11: *rys* overexpress total and polyadenylated core histone transcripts  
qPCR results for degenerately-primed families of core histone transcripts, from random hexamer primers for panels A and B, measuring total core histone transcripts, and oligo-dT for panels C and D, measuring polyadenylated transcripts. Panels A and C display results from cDNAs of 6dpf *rys* mutants and siblings compared to wild-type conspecifics, while B and D are from 8dpf animals.

We assess the joint probability that mean *rys* mutant total H2A is greater than the sib mean at both 6dpf and 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, we suggest the most plausible causal contributor to the *rys* phenotype, of the transcript families examined, are H2A and H2B transcripts. While the relative magnitude of the overabundance of total H2A and H2B transcript is less than that of the polyadenylated pool, we have significantly less uncertainty about these figures. Still, the polyadenylated transcript results seem to speak to a fundamental loss of control over the abundance of these gene products. Unlike the total transcript pool, siblings retain tight, WT-like control over polyadenylated

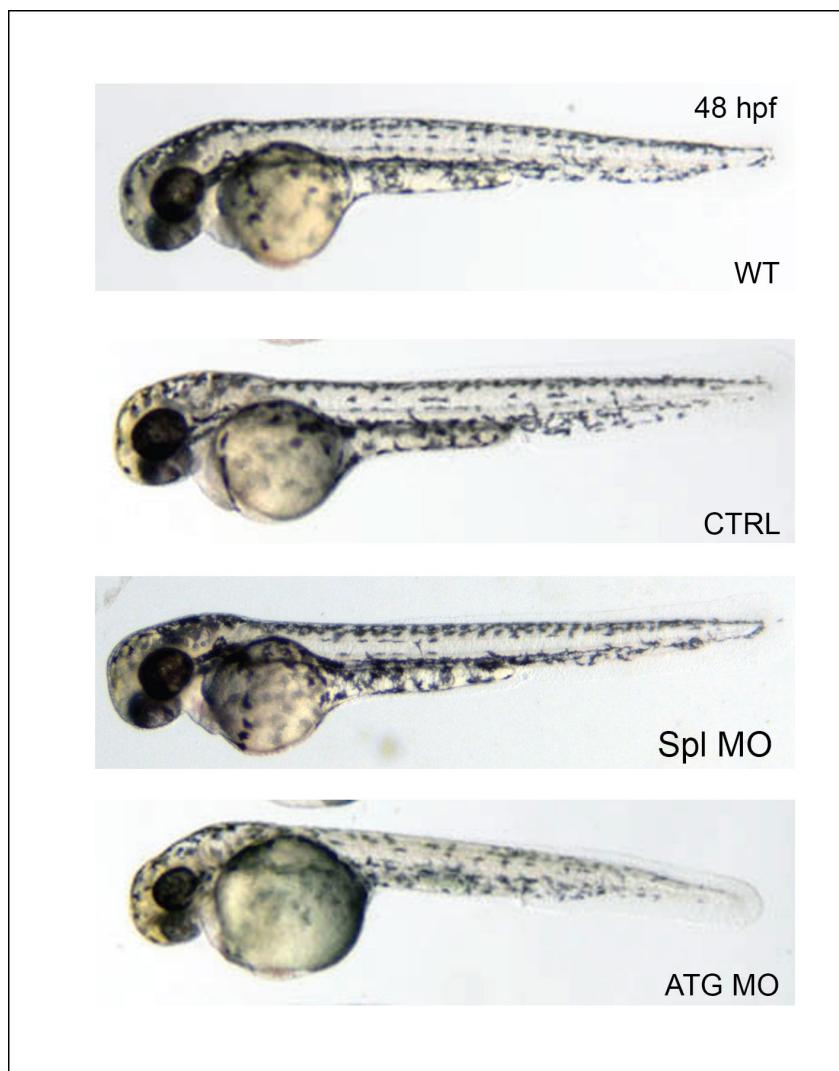
Table 5.2: Standard deviations of significance for *rys* mutant transcript > sib or WT

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

transcripts. *rys* mutants display much more extensive variability in polyA transcript expression, even where the mean is not elevated above siblings or WT controls.

While these observations substantiate the involvement of npat in the *rys* phenotype, we pursued the matter further by perturbing npat using morpholino injections of wild-type fish. As we do not suppose morpholino transcriptional blockade will precisely replicate the cellular conditions of a genetic null some days after birth, 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 consistently replicated the *rys* phenotype on both total and polyadenylated core histone transcript abundance, while the ATG morpholino more narrowly replicated the effect on polyadenylated transcript (Figure 13.5). The ATG morpholino produced animals with small eyes by 72dpf, as shown in Figure 5.12. We also conducted an analysis of the nuclear morphological parameters measured in Figure 5.4 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 CMZ cell, confirming our observation of overall decreased eye size in ATG morpholino-injected animals. 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 concomitant decrease in CMZ population.



**Figure 5.12: 48hpf embryos injected with an npat ATG-targeted morpholino display a small-eye phenotype**

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

### 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

Our observation of perturbed histone expression in *rys* embryos led us to suspect that the aberrant nuclear morphology observed in *rys* CMZ progenitors arises from disrupted chromatin organisation, which is a possible consequence of altering the dynamic composition of the histone pool available for nucleosome formation during cell cycle. We therefore sought to characterise nucleosome positioning in *rys* and wild-type siblings by micrococcal nuclease (MNase) digestion of pooled genomic DNA (gDNA). Nucleosome-protected fragments from the MNase digests were sequenced and positions called as described previously.

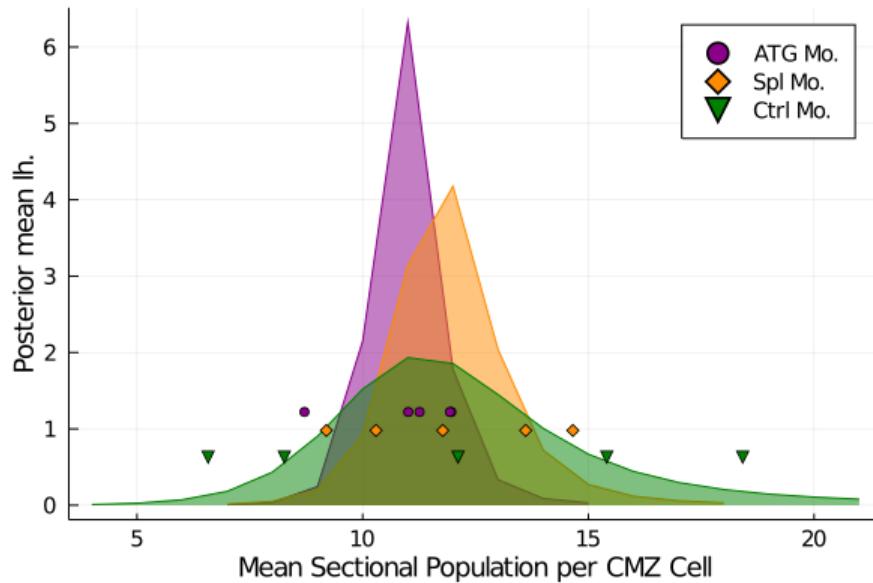


Figure 5.13: ATG morpholino perturbation of npat recapitulates decline in mean central retinal population relative to CMZ

We first examined the genomic disposition of nucleosome positions within wild-type siblings and *ryst*. We found nucleosome positions distributed relatively evenly over sib genomic material, with only tiny deviations from a neutral assumption of a uniform distribution of the total position number over the full length of the genome, as displayed in Fig. 5.14, panel A. Bulk *ryst* chromatin displays minor differences from the proportions of positions found in each sib scaffold (panel B). 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, *ryst* chromatin is somewhat more evenly occupied than sibs; scaffolds with positions that are more heavily occupied in sibs tend to be depleted in *ryst* and vice versa (panel D).

By mapping the called nucleosome positions in *ryst* to those in sibs, and calculating the number of bases the *ryst* position is displaced from its mapped position, we characterised the population of *ryst* positions by translational displacement from the arrangement found in siblings. The probability distribution of this displacement parameter for the population of *ryst* positions is displayed in Figure 5.15. 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, 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 *ryst* 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 *ryst* positions display more extreme displacements, up to the entire length of the position. The periodic multimodality observed in the displacement distance of *ryst* positions relative to sibs is commonly observed in patterns of nucleosome positioning and arises from the 10-base periodicity of nucleosome contacts with DNA [?]. This distribution of population displacement strongly suggests that the npat mutation in *ryst* results both in a loss of translational control of nucleosomes that are in approximately the expected positions, most commonly

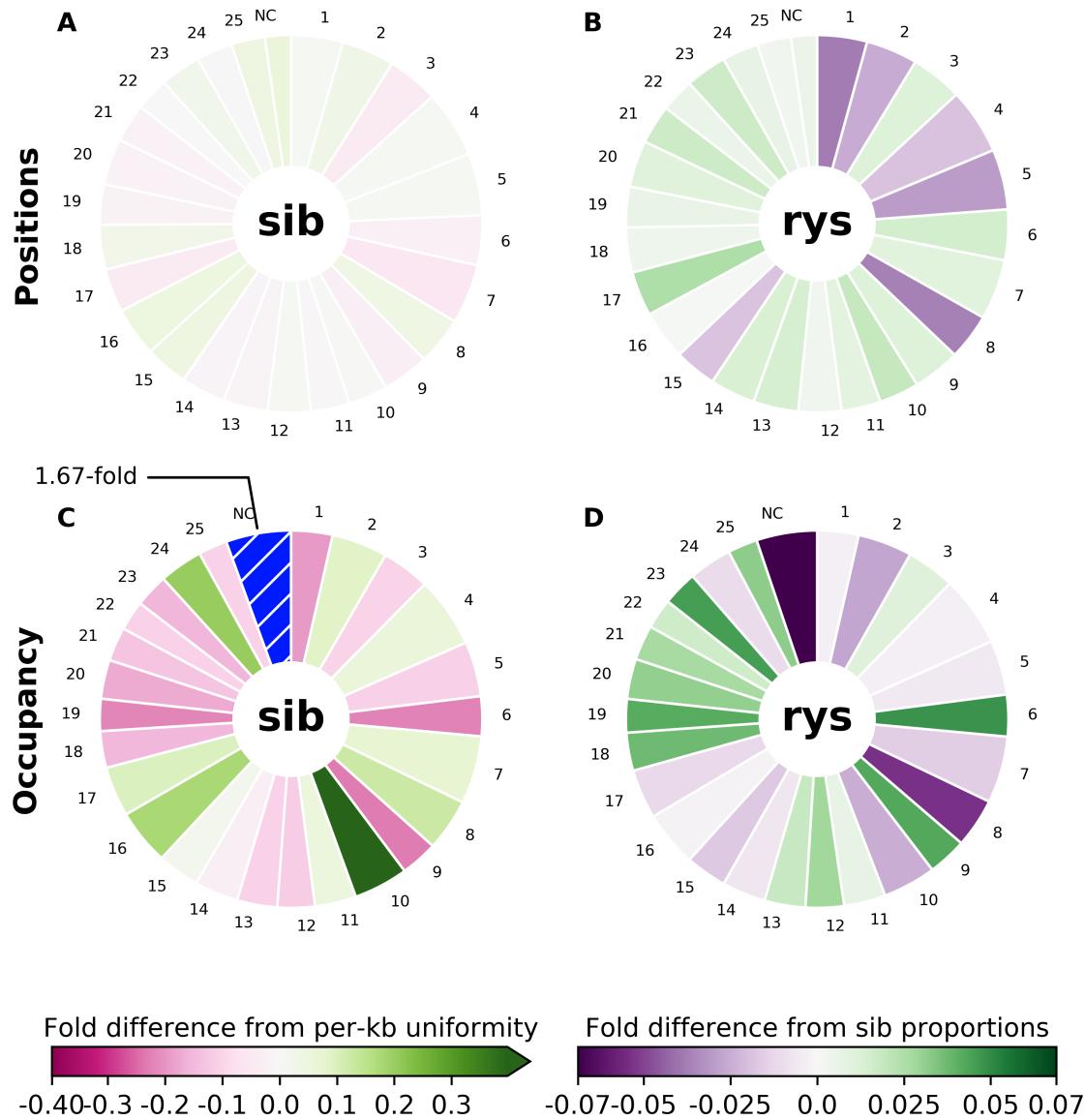
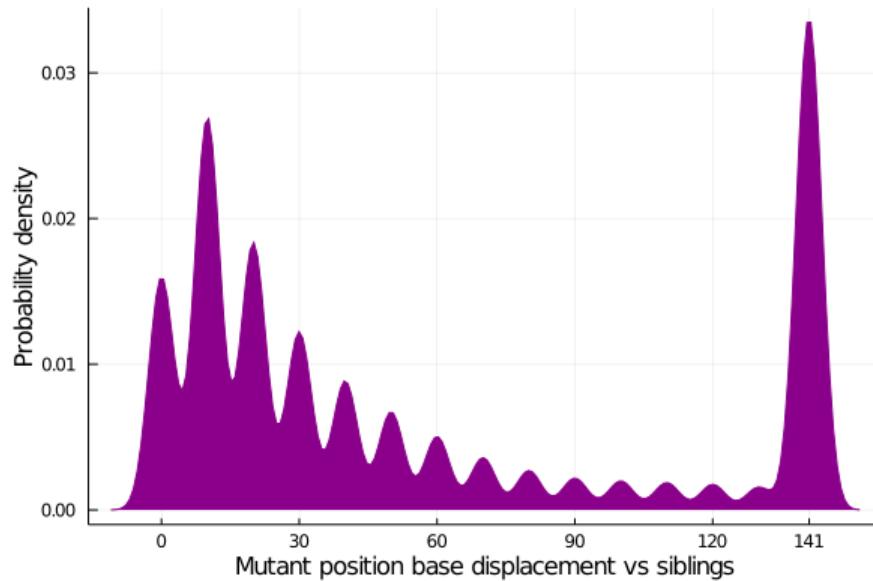


Figure 5.14: *rys* chromosomes are differentially enriched and depleted of nucleosome position density and occupancy.

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 C 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.



**Figure 5.15: PWM sources detected in sibling differential nucleosome positions.**  
Probability distribution of *rys* position displacement distance from mapped sibling positions, in base pairs.

by a nucleosome “roll” of one contact, as well as a loss of trans-acting nucleosome positioning control which gets nucleosomes to the correct positions to begin with, represented by the novel *rys* positions, with no mapped sibling counterpart.

Mindful of the whole-organism source of the nucleosomal genomic sample used to generate the called positions, and of the heterogenous nature of the *rys* phenotype, with RPCs displaying the nuclear phenotype but not specified retinal neurons, the second, novel, group of positions mentioned above was of interest, as we sought to identify position subpopulations that might represent those involved in the nuclear phenotype. 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 *rys*. Intriguingly, sib positions which are 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 13.6.

This result suggested that a particular subpopulation of wild type nucleosomes are mislocalised in *rys*. If the pool of available histones in proliferating *rys* progenitors is substantially different from that of siblings, *rys* nucleosomes forming from this pool may have altered physico-chemical interactions with DNA, which could result in 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 formally address this hypothesis, we calculated the Bayesian evidence ratio for separate emission processes for sib and *rys* position sequences against a combined process for both sets of sequences. If the molecular process resulting in the differential set arises from altered nucleosome composition in proliferating *rys* cells, we expect these sequences to provide greater support for separate emission processes than for a single, combined process. On the other hand, if aberrant DNA-nucleosome interaction is not

causally relevant, and the reason for the apparently displaced nucleosomes is another macromolecular process, the evidence ratio should favour a combined process for the emission of the differential set—that is, there should be no difference between the unique sib and *rys* sequences that would justify the additional complexity of separate models.

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 some 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 ??) 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. 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 13.7.

We next used the Julia independent PWM component analysis nested sampling library BioMotifInference (presented in ??) to sample from the posterior distribution, 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 5.16 and Figure 5.17 respectively, while the inferred combined sources are available in Supplementary Figure 13.8.

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 ± 0.43	-1.161415e6 ± 0.35	<b>-2.278988e6 ± 0.56</b>	-2.279356e6 ± 0.42	368.5 ± 0.7

BioMotifInference's source detection was highly conservative, likely due to the good quality of the background models, which 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

<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.

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

r<sub>ys</sub> 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 *r<sub>ys</sub>* positions, we found twenty-eight separate PWM sources expressed in more than 10% of observations, in a variety of posterior modes. This suggests sequence preferences are much more explanatory for the *r<sub>ys</sub>* differential position set.

The detected sources in the siblings are not altogether unexpected; CWG motifs are among the most commonly reported in nucleosome sequences, and have been described as promoting nucleosome formation. 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. *r<sub>ys</sub>* 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 *r<sub>ys</sub>* positions, and the rare ATGG repeat is not found at all.

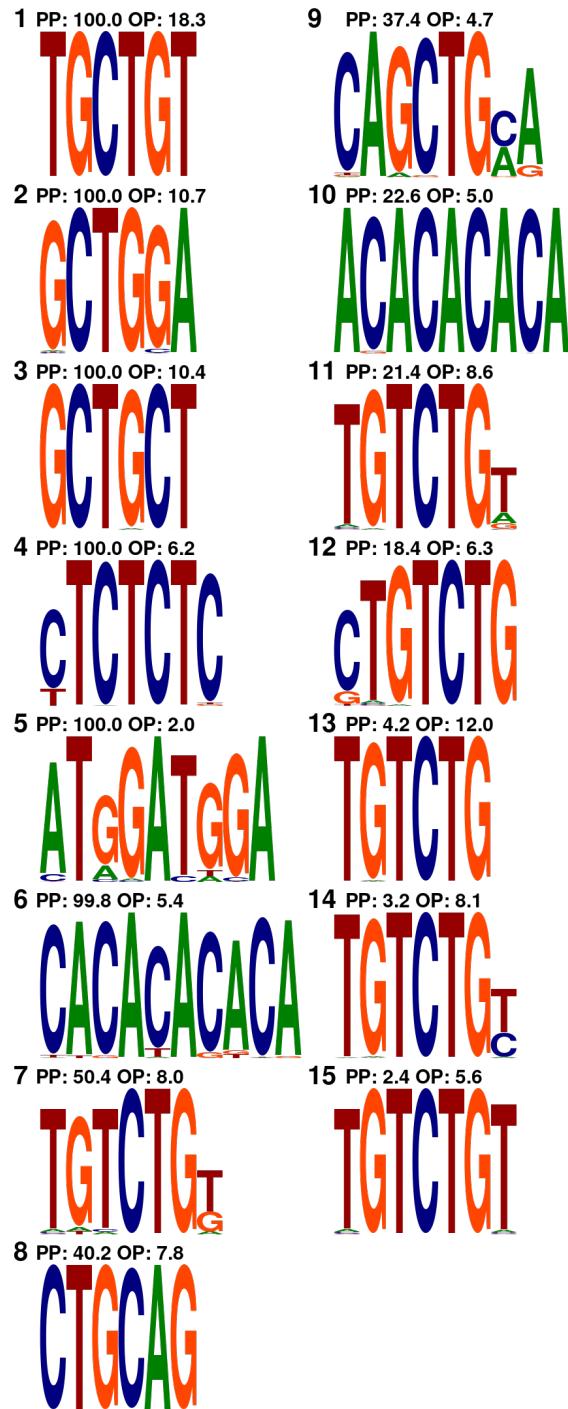
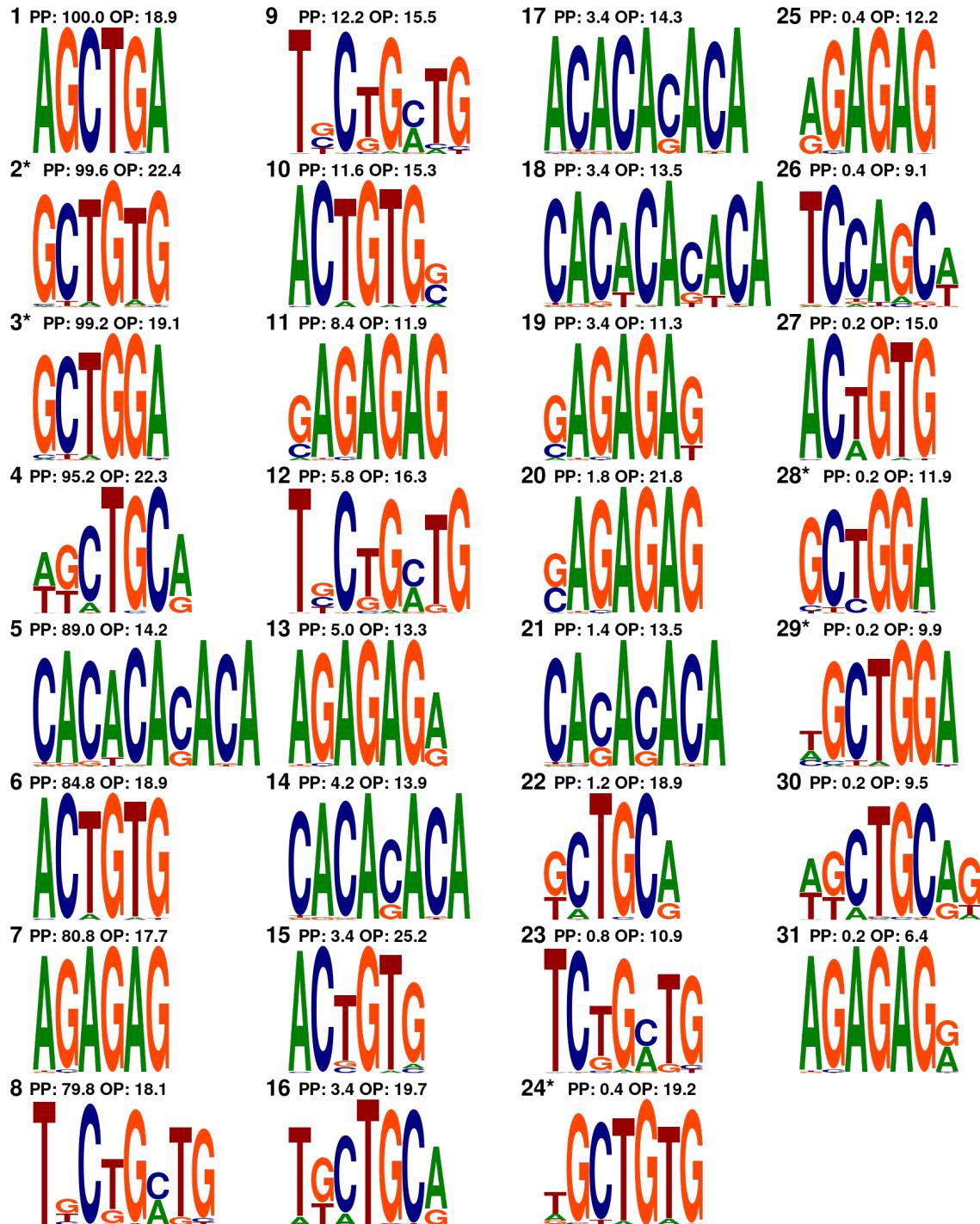


Figure 5.16: PWM sources detected in sibling differential nucleosome positions.  
PWMs presented as sequence logos with letter height proportional to position informational contentemission probability.

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

Figure 5.17: PWM sources detected in *rys* mutant differential sources.

PWMs presented as sequence logos with letter height proportional to position informational content/selection 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

### 5.3 Discussion

We began our investigations by adding depth to the original description of the *rys* mutant CMZ phenotype; on the basis of these investigations, we believe that a revision is in order. At any given time, the apparently increased size of the CMZ in *rys*, 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 a failure of RPCs to specify as particular retinal neural lineages. This is substantiated by the inappropriate retention of early progenitor markers in these cells. We demonstrate that *rys* RPCs are not, however, arrested in cell cycle. In contradistinction to results finding elevated polyadenylated histone transcript levels associated with 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 general, well-evidentiated macromolecular mechanism underlying phenomenon 3 which could plausibly produce phenomena 1 and 2. That is, the 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 combined emissions process, we found substantial evidence in favour of separate emissions processes. 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 highly suggestive against this

---

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

background. It seems likely that the shifted and novel nucleosome positions observed in mutants reflects all of the following:

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 perturbed histone expression arising from altered npat activity. As the spatiotemporal dynamics of chromatin architecture are known to be critically involved in both replication timing [GTR<sup>+</sup>10] and cell identity and fate [SVT13], the significant alterations we observe in *rys* mutant chromatin architecture also provide a plausible molecular mechanism by which mutated npat protein can produce 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 seems to 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 to compress the ICA ensembles in our nested sampling procedure. For reasons explained in ??, 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 by several of magnitude as a consequence. If we concede that there are perhaps 5 orders of magnitude more error than estimated, this has the effect of reducing the significance of the result from to standard deviations, which is still well above the typical  $5\sigma$  significance typically accepted as a discovery in the physical sciences. We conclude that even given a more conservative estimate of the sampling processes’ error, it is far 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 lacunae in this explanation; it remains unclear what form of the npat protein is expressed, if any, as well as what the overall effect on the expressed pool of histones in RPCs is, for instance. It would not be very surprising if there are a number of macromolecular species intimately 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 specificative 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, we suggest 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 specific 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 heretofore 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 (Di Frusco 1997). 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 due to our failure to identify any significantly similar CDS sequences in the Zv9 zebrafish genome using BLAST, and the synteny analysis in Figure 13.1. 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 hiflal, 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 *rys* phenotype. In both 6 and 8 dpf *rys* larvae as well as npat-morpholino treated 1dpf embryos, we observe increased abundances of histone and, specifically, polyadenylated histone transcripts, although the composition of affected histone families differs between these contexts. The specific mechanism by which these effects are produced in *rys* remains unclear, and the increases in histone transcript abundance observed in both mutant and morpholino-treated animals is 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 *rys* npat protein, or treatment with a CDK inhibitor, result in similar declines in histone transcript abundance [Ma00, MGv<sup>+</sup>09]. This may imply the role of zebrafish npat in regulating histone transcription is different from what has been described for human NPAT. We are unable to determine from these data whether the overall increases histone transcript abundance are a consequence of increased histone transcription, or of the greater stability of improperly polyadenylated histone transcript, however. The increased abundance of polyadenylated transcript in *rys* 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. As noted above, 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 *rys* 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 *rys*, this seems to be related to the effect of altered histone expression on chromatin structure.

# 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 detail in Chapter 5 provides a useful denouement to the basic theme running through this thesis, the notion 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]. In *rys*, we receive a graphic illustration of the sense in which this is so: 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, many 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.

That this dissociability has fundamental implications for the quality of our inferences about RPCs, and, therefore, our ability to predict and control their behaviour, can be gleaned from the basic problems encountered in each of the modelling studies. To begin with, the Stochastic Mitotic Mode Explanation models failed, in large part, because the underlying Simple Stochastic Models conflate proliferation and specification. 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. It thus becomes plain that lineage specification and niche exit processes need to be modelled separately from mitotic processes. Next, proliferative data and morphological measurements largely 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. This demonstrates that, in order 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 and nucleosome sequence preferences. This last model directs us to the relevant

---

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

level of biological organization for any rigorous explanation of RPC lineage outcomes: the progression of chromatin states that we take to underlay long-term changes in gene expression which are required for the generation of the specified protein complement. Our inferential approach, while providing good evidence for the chromatin dynamics we believe arise from the effects of aberrant histone transcript regulation on the nucleosome pool, does not give us insight into how the appropriate progression of chromatin states might be usefully parameterised, beyond confirming that it is likely under the active control of *trans*-acting factors.

While a large number of techniques for studying aspects of chromatin structure exist, few suggest model parameterisation for capturing 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, it remains unclear what sort of parameter space would allow us to infer where the ball is. Plainly, older ideas of cellular state space that rely exclusively on measures of protein expression are unable to address hypotheses about chromatin structure. One might, in the “-omics” enumerationist mode, 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, even with the sophisticated Bayesian inference tools developed here, which can estimate the evidence for models with thousands of parameters across millions of observation. Practically speaking, it remains unclear what aspects of chromatin organisation are causally important in fate commitment, as little formal model comparison has taken place. Given this, prior distributions on the model parameters are unlikely to be well enough constrained to allow the convergence of ensembles on posterior distributions with reasonable compute budgets.

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

*rys* 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 *rys* mutants, many aspects of the suggested mechanism remain obscure. This arises, in large 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 [AlQ19]. 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 5.17 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 *rys* 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 *rys* “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 bioorthogonally labelled transcripts, as has recently been reported 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 functional biological “meaning” arises from the integration of salient cellular information by a combination of nonarbitrary physicochemical relations and arbitrary biological coding relations, is currently best described by the subdiscipline of biosemiotics [Hof08, Fav15, Hof15]. Of particular interest here is the clear-eyed and unobscuring work of Marcello Barbieri, which may well provide the metaphysical framework necessary to plumb the depths of this topic [Bar15].

## **Part II**

# **Software Technical Reports**

# Chapter 7

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

`GMC_NS.jl` implements a basic Galilean Monte Carlo nested sampler [Ski12, Ski19] in pure Julia. It uses the updated Galilean reflection scheme of Skilling's later GMC publication [Ski19], which improves the algorithm's performance and preserves detailed balance. 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 achieves the same end as algorithms which dynamically adjust the ensemble size based on parameters of the ensemble [FHB09, HHHL19], without any computational overhead or additional objective function. 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 even if the number of trajectories started with is insufficient to guarantee this.

### 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 by its current likelihood (which is the ensemble's likelihood contour for that iterate). GMC tends to move model-particles across the parameter space very 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. This is accomplished by random "diffusive" proposals, in contrast to the ordinary, directed Galilean movements. This means 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 unit-standardized hypersphere. The user is responsible for ensuring that the density of the prior is uniform, that is  $\pi(x) = 1$ , over the -1:+1 range of the hypersphere dimension. Utility functions `to_unit_ball` and `to_prior` which accept the prior and transformed positions and the relevant prior probability distribution, returning the unit hypersphere or prior coordinate, respectively. In future versions, this scheme is likely to be modified to a 0:1 unit hypercube, which does not require additional 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 (eg. is the plane at  $n = 0$ .), the reflection is performed by stopping the particle at the box side and giving it a new velocity vector identical to the old but with reversed sign in dimension  $n$ .

`GMC_NS.jl` attempts to maintain efficient GMC sampling by per-particle PID tuning of the GMC timestep. GMC treats models as particles defined by their parameter vectors, which give their positions in parameter space, as well as a velocity vector, normally chosen isotropically and only changed when the particle "reflects" off the ensemble's likelihood contour. In GMC, when a model-particle is to be 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 fairly evenly in order for GMC to be efficient, 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 for the trajectory, tuned to target a user-supplied unobstructed move rate (given as a floating point fraction in the range 0:1). In short, this will reduce the timestep if the particle is repeatedly 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 repeatedly moves without encountering the boundary, so that particles that are closely sampling an open region without encountering the boundary begin to "speed up".

## 7.2 Ensemble, Model, and Model Record interfaces

In operation, `GMC_NS.jl` maintains an `GMC_NS_Engine` 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_Engine` 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_Engine`'s `obs`, `priors`, `constants`, and `box` fields, respectively. The `GMC_NS_Engine` also specifies settings for the GMC sampler and the PID tuner. Sensible defaults for these values may be passed 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 with the fields specified by the docstrings in the `/src/ensemble/GMC_NS_Engine.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_Engine)` function for displaying the model with observations. 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_Engine` 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 defines the 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 `[-11]`. 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 PID tuner settings for GMC

As noted, GMC particle timesteps are governed by individual PID controllers. These controllers are defined by a number of values which must be supplied as `GMC_NS_Engine` fields. These are summarized below:

`GMC_Nmin` (Integer): 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.

`GMC_τ_death` (Float): Value of  $\tau$  timestep associated with a trajectory below which the particle is considered to be dead. This value is in hypersphere coordinates. It is reasonable to make it very small

for likelihood functions with low or no Monte Carlo noise; it should be somewhat larger for functions with significant MC noise.

**GMC\_init\_** $\tau$  (Float): 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 current location.

**GMC\_tune\_** $\mu$  (Integer): Length of tuner memory; the proposal acceptance rate  $\alpha$  is calculated over the previous  $\mu$  sampling steps for this trajectory.

**GMC\_tune\_** $\alpha$  (Float): Target acceptance rate for proposals. Should generally be .8 or higher, depending on desired sampling density.

**GMC\_tune\_PID** (NTuple3,Float): kP, kI, and kD constants for the tuner. Keep in mind that the range of values for  $\alpha$  is 0.:1.; given a **GMC\_tune\_** $\alpha$  constant of .8, the maximum error that will be encountered is likewise .8, so the values of the constants will generally be much smaller than 1. As with most PID applications, kP should be the largest value, and in this case generally determines the number of searches conducted before **GMC\_** $\tau$ \_death is reached.

**GMC\_timestep\_** $\eta$  (Float): 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\_** $\eta$  (Float): If this value is >0., it is used to perturb reflection vectors, as suggested by [Ski12]. It should be a small value.

**GMC\_exhaust\_** $\sigma$  (Float): Scale parameter applied to ensemble size to determine the maximum number of unsuccessful searches to perform before terminating sampling. Can usefully be fairly large.

### 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

If desired, 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 `convergeensemble!()`. **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 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 #Index 2: The minimum allowable particle timestep is set to a very small number; the
23   ↳ likelihood of the model parameterisation can be determined precisely and is well
24   ↳ behaved over the parameter space.
25 gmc=GMC_DEFAULTS
26 gmc[1]=5
27 gmc[2]=1.e-15
28
29 #Assemble the ensemble. Begin with 10 particles
30 e=Normal_Ensemble("NE_test", 10, samples, prior, box, gmc...)
31
32 #Define the upper displays to be rendered above the status line. Here we rotate a
   ↳ single display between three readouts.
33 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]])

```

```

33 #Define the lower displays to be rendered below. We want to look at the most likely
  ↵ model in the ensemble plotted against the observations the whole time.
34 lds=Vector{Vector{Function}}([[model_obs_display]])
35
36 #Converge the ensemble, serializing to pwd/NE\test every 100 iterates and rotating
  ↵ the chosen displays every 1000 iterates.
37 converge_ensemble!(e,backup=(true,100),upper_displays=uds, lower_displays=lds,
  ↵ disp_rot_its=1000)

```

---

Typical output after executing this program is presented in Figure 7.1. 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.

```
julia> converge_ensemble!(e, backup=(true,100), upper_displays=uds, lower_displays=lds, disp_rot_its=1000)
    Information History 4, i 2655, τ 0.00016045922271270535 ± 1.0e-155
    Ensemble H

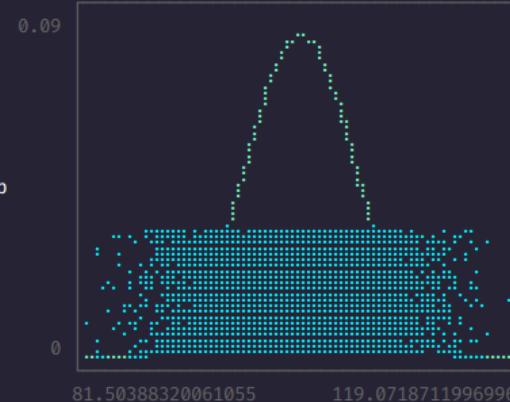
    Iterate
GMC-NS:: Converged. Time: 0:00:22 (5748 iterations). logZ: -31343.506034949176
Current MAP model
    Normal Model 1.19, log_Li -30390.369402920995
    Model
    Obs

    p
    X
    θ: [100.2878772001551, 0.04534653431580314]
    v: [0.17847619452553798, 0.4335501569005585]
    [ Info: Job done, sampled to convergence. Final logZ -31343.50451521636 ± 12.098247734502335
```

Figure 7.1: Example BMI.jl Output

# 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](#). This package extends `GMC_NS.jl`, described in [Chapter 7](#) and requires it. Prior distributions are specified by `Distributions.jl`, `ConjugatePriors.jl`, or other `Distributions`. The simulators are organized around the nested sampling procedures implemented in that package. This means that simulations are executed as sample chains 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 sample chains 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_1lh`, found in Section 16.5.3) 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:

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

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 in this, and other models of the package, 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 16.9.5.

### 8.3 Slice\_Ensemble 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 16.5.5 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.

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)

```

---

## Chapter 9

# BioBackgroundModels.jl: Parallel training 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 about as fast as 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 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 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 `: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 for worker in remote_pool
15     load_dict[worker]=remote_config
16 end
17
18 worker_pool=vcat(worker_pool, remote_pool)
19
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 13.7. 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>2</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 HMMs are correctly converged on a global optimum, the autotransition probabilities for all chains should

---

<sup>2</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.

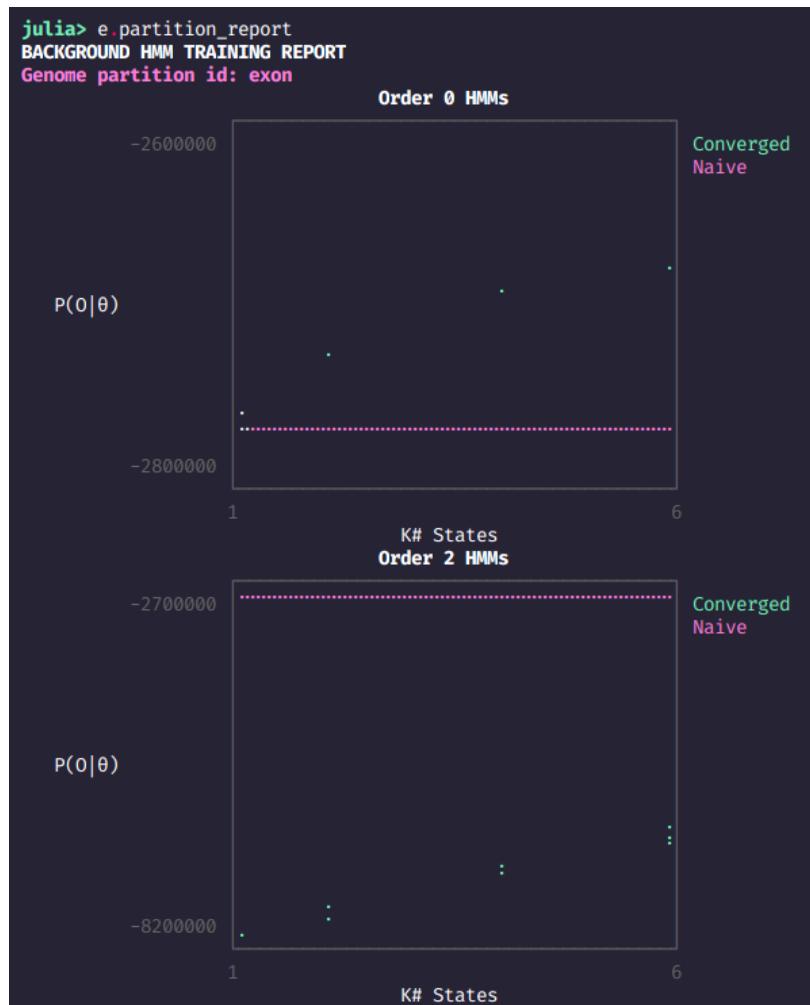


Figure 9.1: Example BBM.jl Partition Report

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 as the genomic partition on which it was trained. This is followed by the likelihood of the model given

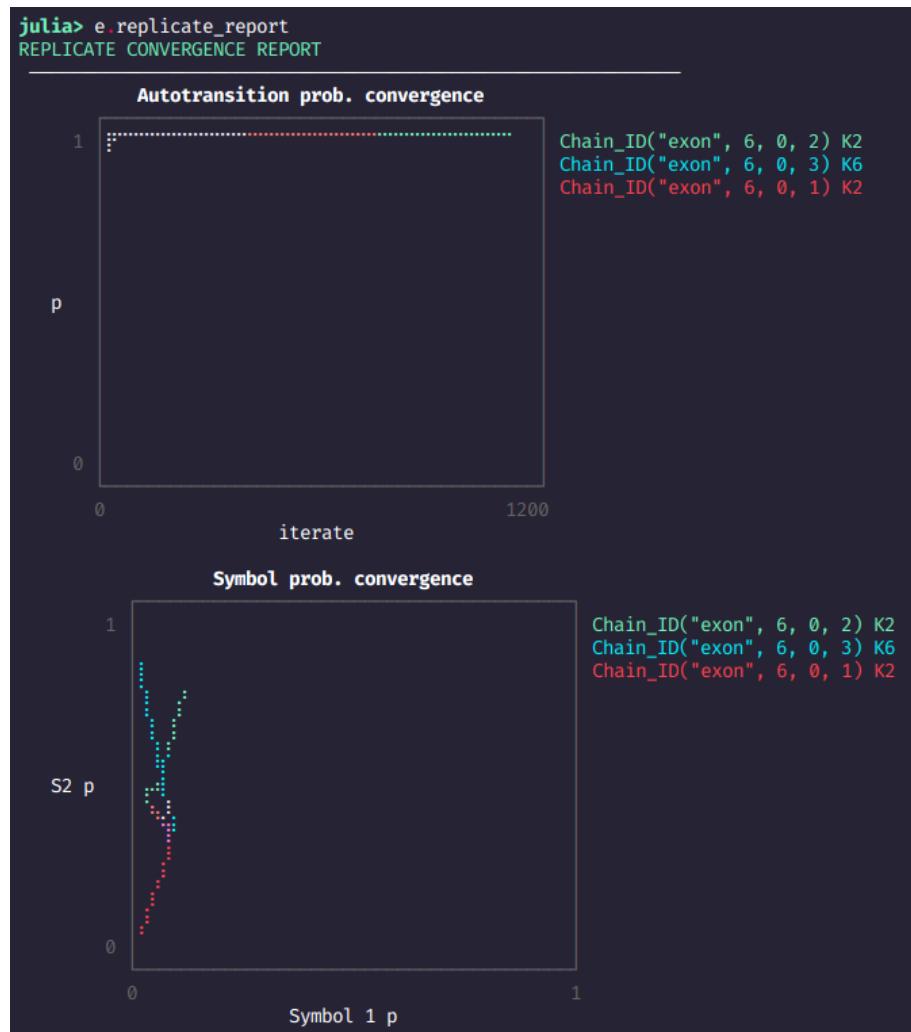


Figure 9.2: Example BBM.jl Replicate Report

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 states encode.

```
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: [0.3178632049327706, 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

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 can be determined by finding the parameters with a test value of 0. Taken together, these reports can

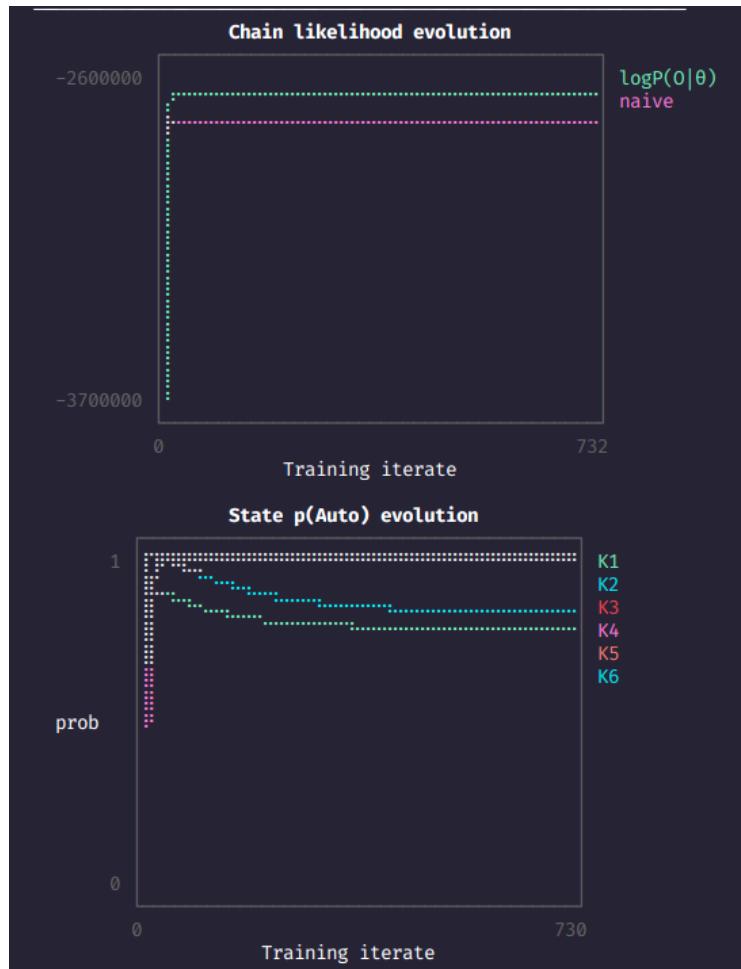


Figure 9.4: Example BBM.jl Chain Report - Part 2

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
$\log P(\mathbf{o} \theta)$	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	0.9938	0.0002	1.0000
K3	0.0000	1.0000	0.2333	0.9936	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

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.

# 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 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.

### 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]. Skilling acknowledges an important assumption underpinning this guarantee, which 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 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 massively increasing computational cost to questionable benefit<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<sup>2</sup>. 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. Despite this, we show that its evidentiary accumulation behaviour strongly resembles that of a nested sampler that *is* in detailed balance, suggesting that this problem is not fatal to the overall approach. Moreover, because of the extremely 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

---

<sup>1</sup>Probably some combination of dimensional analysis and fast alignment algorithms could make some headway here, but it's not clear that it's necessary to do so.

<sup>2</sup>It is also likely to remain beyond my abilities. Please notify the author or submit a relevant pull request at <https://github.com/mmattocks/BioMotifInference.jl> if this is something you know how to do.

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 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 be 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. This feasibility constraint 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 for model structure given the data, given as its logarithm, 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:

---

```

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_clmps` and `max_clmps`), 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

If all of the above-described tasks are met, 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)

```

---

## **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 2.5. 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 2.7, 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 2.7, 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  $S = \pi \times d_{lens} \times h_{section}$ , and the total surface area of the lens by  $A = \pi \times d_{lens}^2$ , where  $d$  is the diameter of the lens and  $h$  the section thickness, we may calculate an estimate of the total population by  $pop_{total} = \frac{pop_{section}}{S} * A$ . We therefore obtained our annular CMZ population estimates using our measurements with the simplified formula  $pop_{total} = \frac{pop_{section} \times d_{lens}}{h_{section}}$ , 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 2.7.

### 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>, as well as in the supplementary archive [??](#). 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 similarly produced. 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 2.4) 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 2.4) 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 2.4 (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

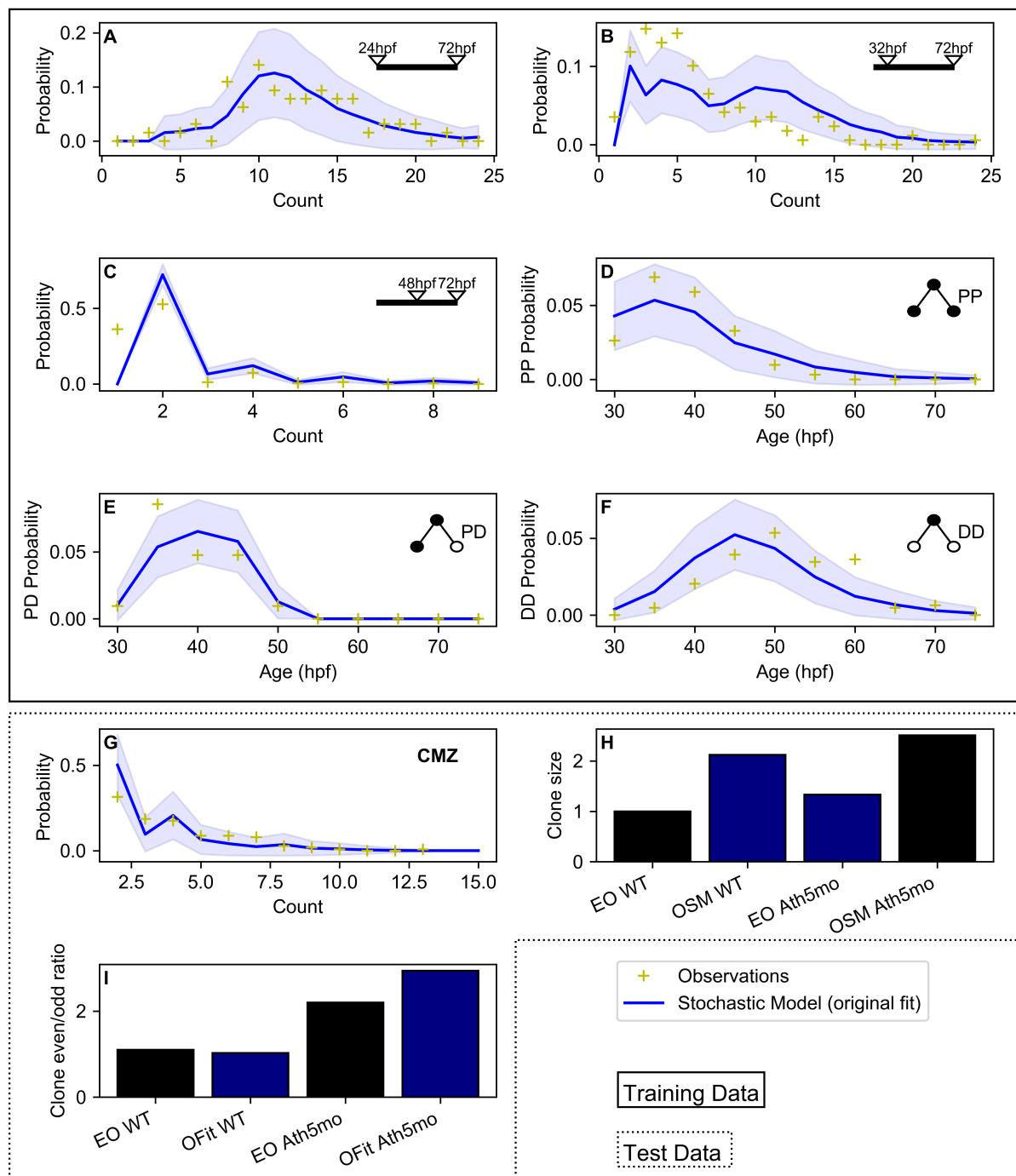
CMZ, “forced to differentiate”, in the terms of Wan et al.). A further population of stem cells, sized at  $\frac{1}{10}$  of the RPC population, is added using the `WanStemCellCycleModel`. This stem population divides asymmetrically except when it falls under its target population value determined by 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 2.4 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.

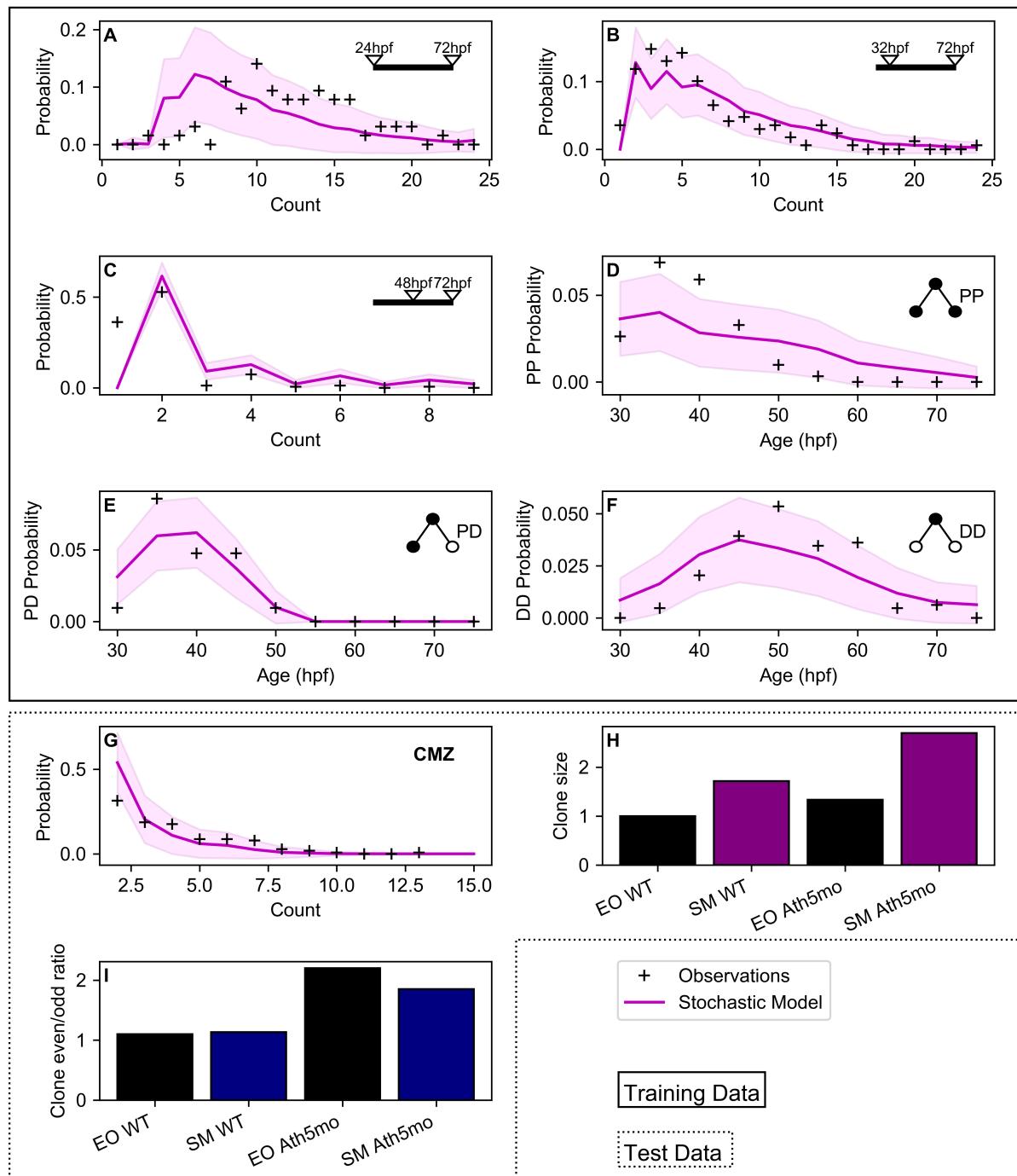
In order to assess the performance of the models used in our simulations, we used Akaike’s Information Criterion (AIC). This allows us to compare models with dissimilar numbers of parameters, as AIC trades off goodness-of-fit against parameterisation. The values reported in Table 2.1 were produced by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

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.

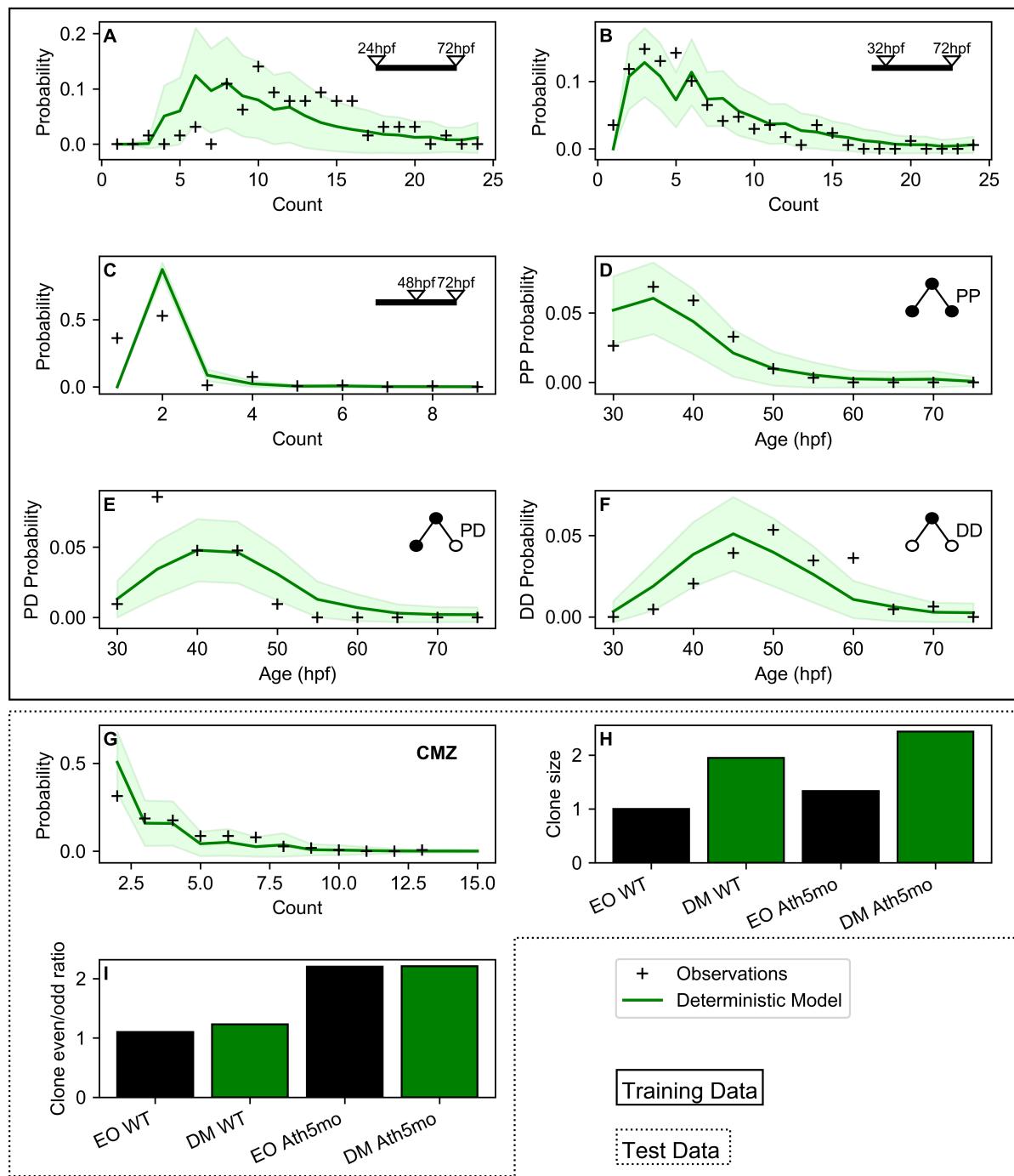
## 11.2 Supporting figures



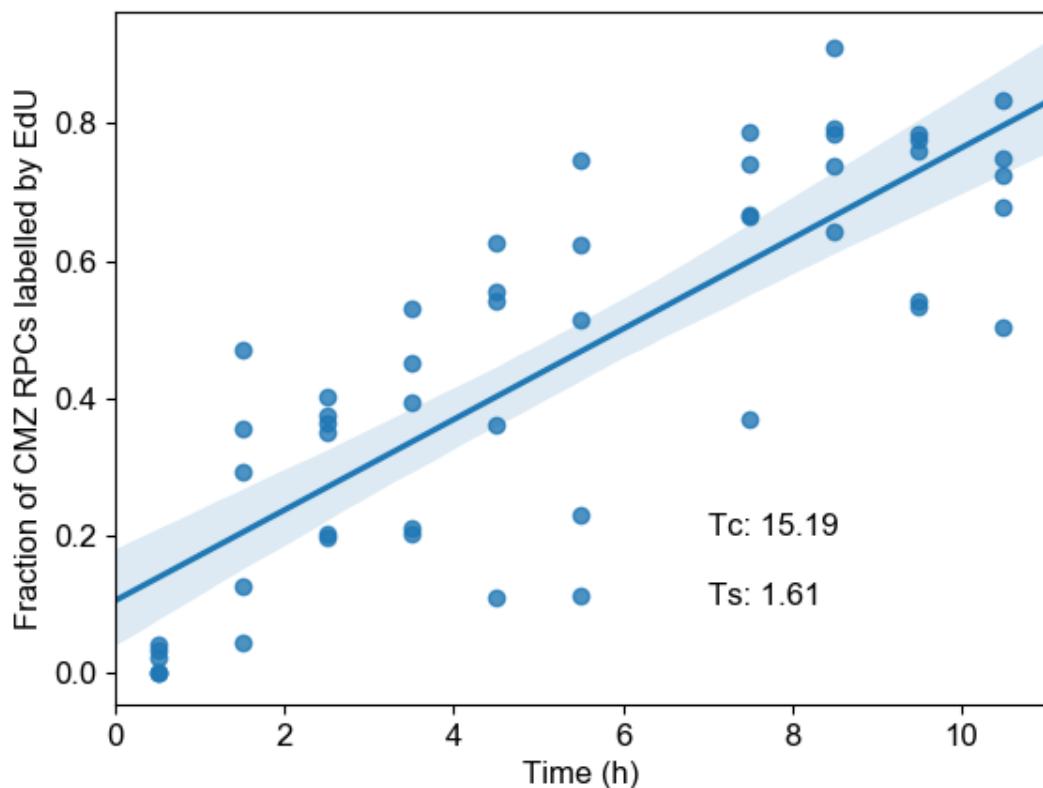
**S1 Fig. He SSM original parameterisation model output.** As Fig 2.4; model output uses the parameterisation given in He et al.[HZA<sup>+</sup>12]. Note significant deviation from observations in panel B, reflecting excess proliferative activity of modelled RPCs.



**S2 Fig. SPSA-optimised He SSM model output.** As Fig 2.4; stochastic model output displayed alone.



**S3 Fig. SPSA-optimised deterministic mitotic mode model output.** As Fig 2.4; 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

### 12.1 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 4.4.1](#), as well as the pulse-chase histochemistry used for the study of neural fate in [Section 4.6](#) 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 all cases, animals were pulsed by allowing them to swim freely in a 10mM EdU solution for 2 hours. 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 in fridge overnight.
2. Chemicon mouse anti-GS primary antibody diluted 1:500 in blocking solution at 4 in fridge overnight.
3. ZIRC mouse anti-Zpr1 primary antibody diluted 1:400 in blocking solution at 4 in fridge 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).

### **12.1.5 4C4 immunohistochemistry**

### **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 (v.). 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 overlap of signal from these channels with the segmented nuclear volumes. The resultant segmented Imaris scenes and the settings Tableautoref summarizes these datasets and

### **12.1.7 Estimation of overall CMZ population and retinal volume**

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

### **12.1.9 Simulation of phased CMZ activity by systems of difference equations**

### **12.1.10 Slice model simulations of phased CMZ activity**

### **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 ayesian evidence], or marginal probability of the model over the posterior sample is calculated by the ested sampling algorithm] [Ski06]. 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. Tableautoref summarizes the Chapter 4 figures which use this technique, along with links to model documentation and the code used to produce the figure.

### 12.1.12 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 in a high performance manner. Tableautoref summarizes the Chapter 4 figures which use this technique, along with links to model documentation and the code used to produce the figure.

### 12.1.13 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

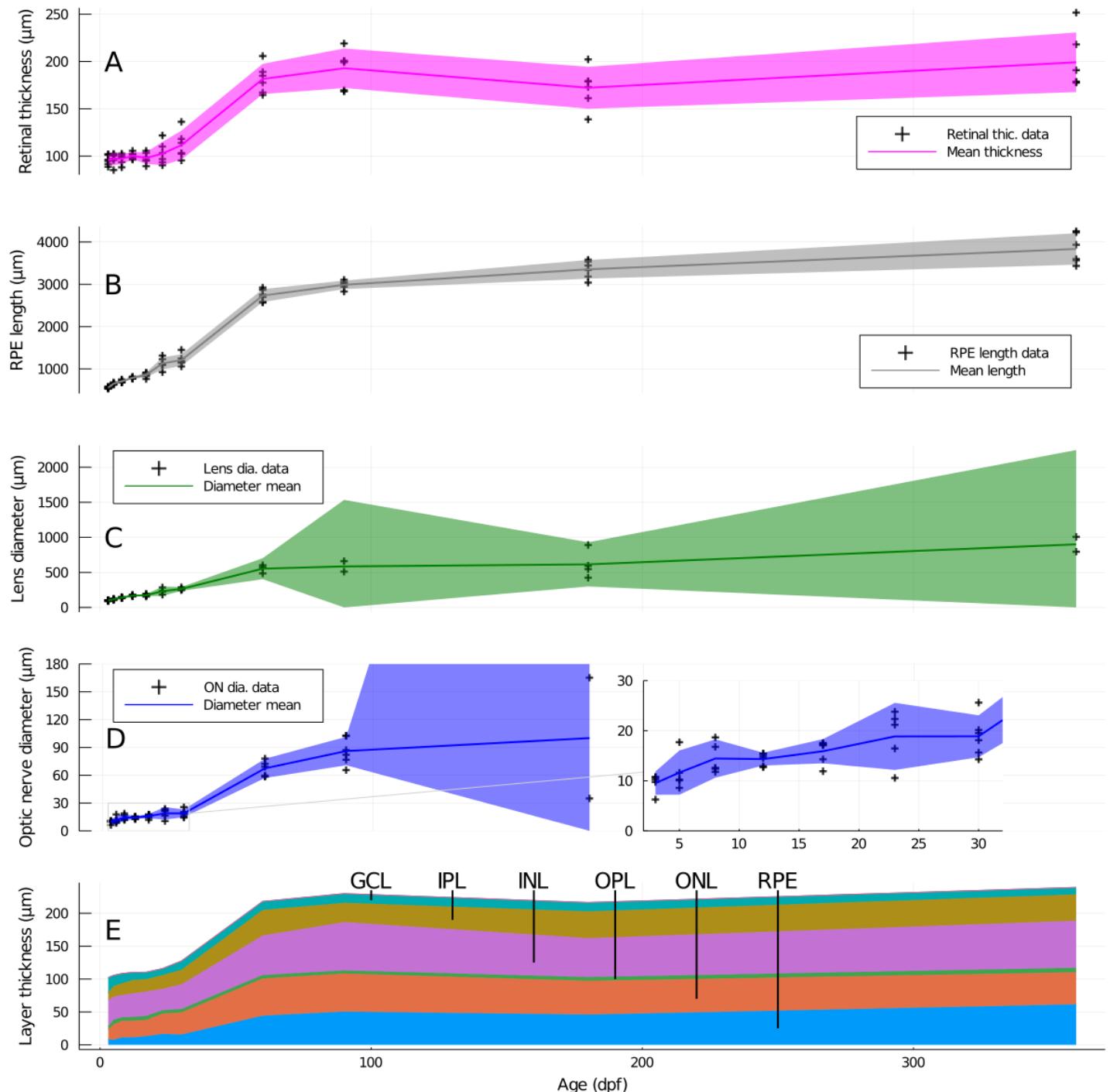


Figure 12.1: Developmental progression of naso-temporal population asymmetry in the CMZ.

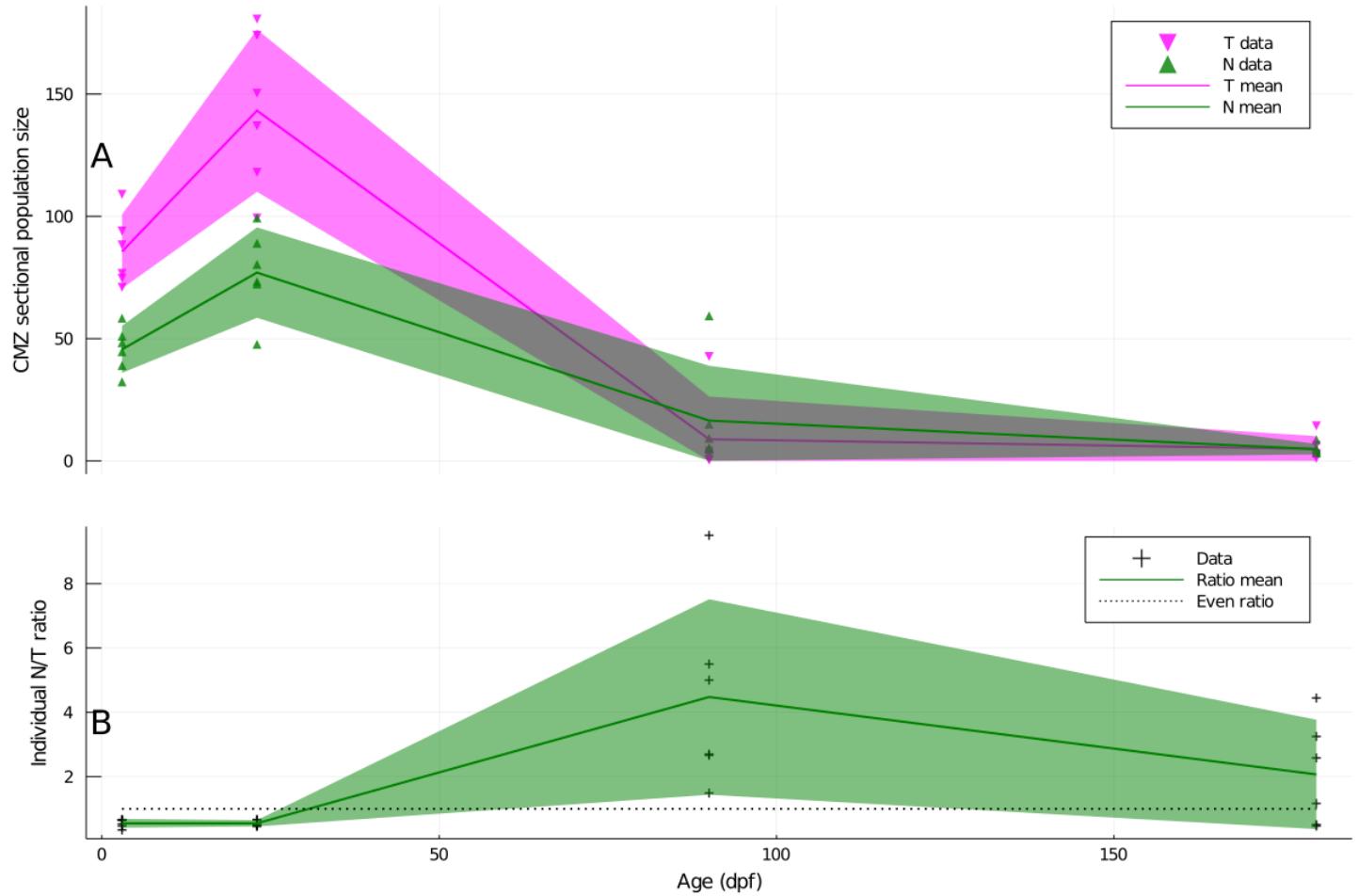


Figure 12.2: **Developmental progression of naso-temporal population asymmetry in the CMZ.**

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.

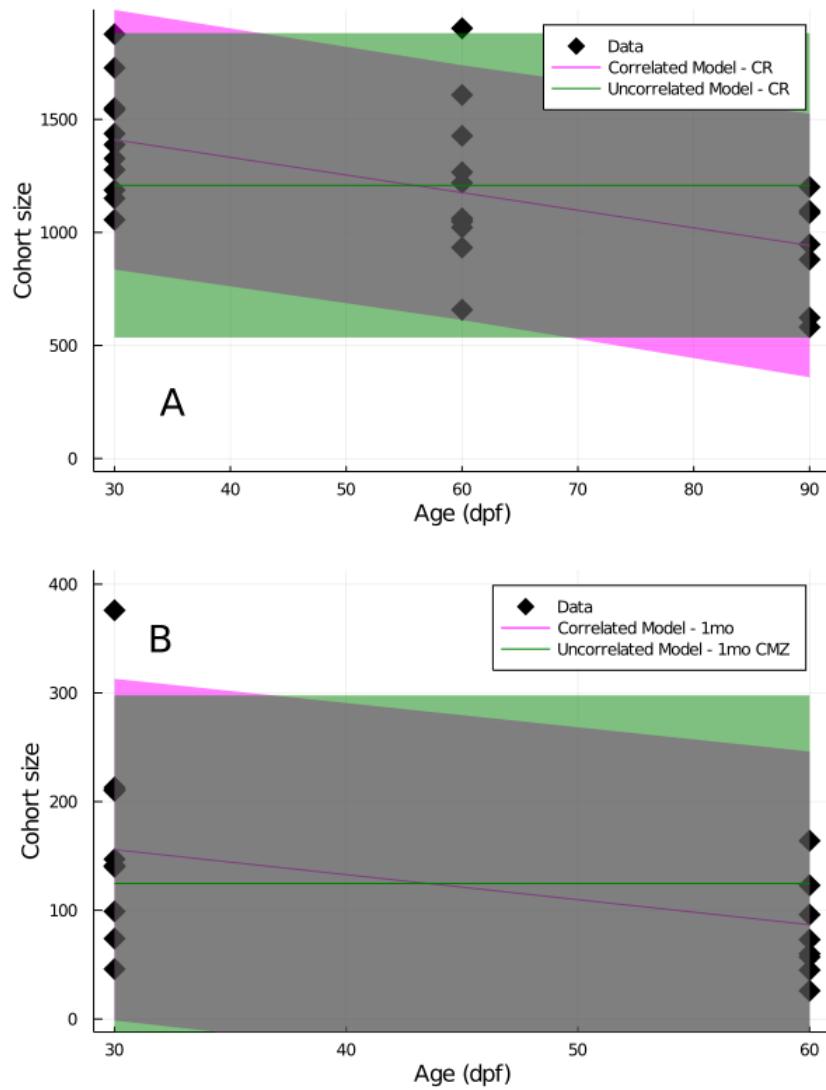


Figure 12.3: Linear regressions of temporally correlated and uncorrelated models of central retinal and 30dpf CMZ-contributed cohorts

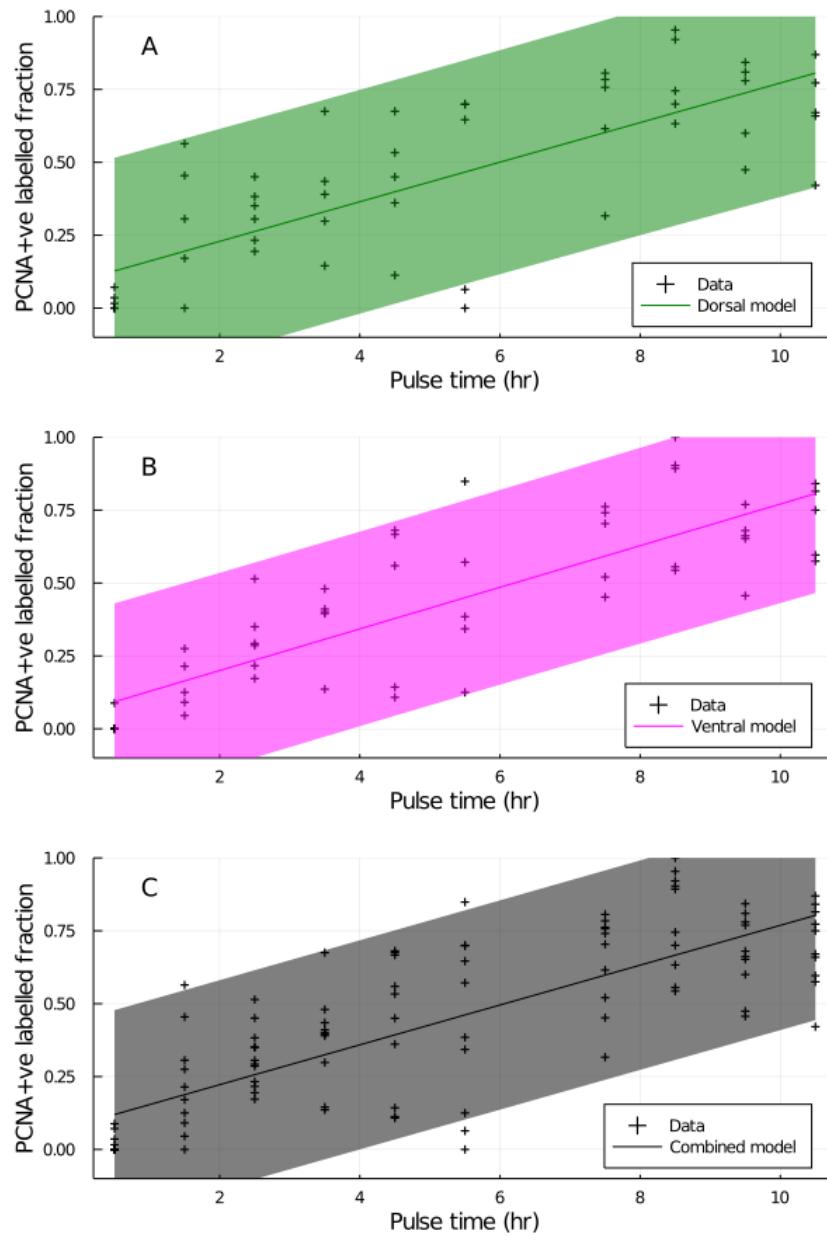


Figure 12.4: Linear regressions performed on cumulative labelling data from dorsal, ventral, and combined CMZ sectional populations

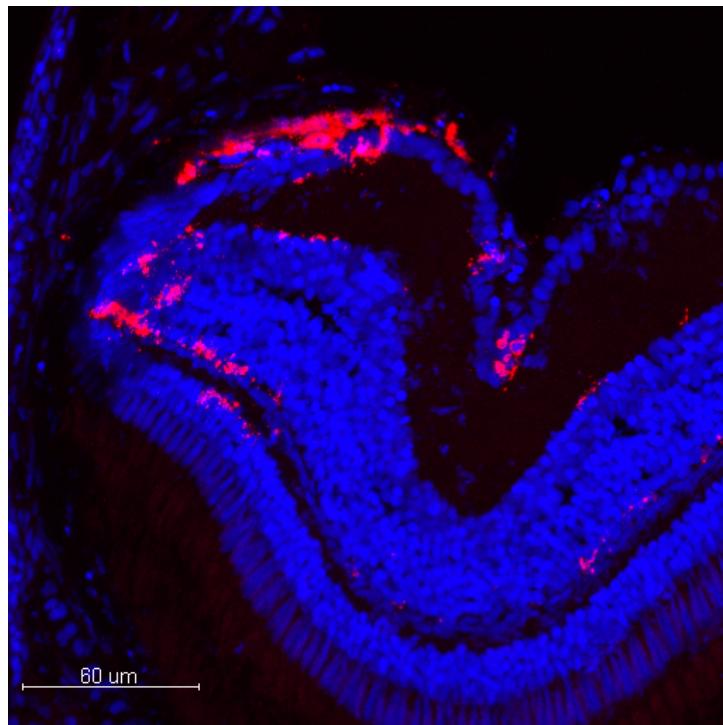


Figure 12.5: **4C4+ microglia are associated with the CMZ**

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 Evidence calculations for models of layer and lineage contribution

Table 12.1: Evidence for Normal vs. Log-Normal models of layer and lineage contribution

Layer	Marker	Cell type	$\mathcal{N} \log Z$	$\text{Log-}\mathcal{N} \log Z$	$\log ZR$	$\sigma$ sign.
GCL	Cohort	All GCL cells	<b>-189.0 ± 1.4</b>	-193.5 ± 1.9	-4.5 ± 2.4	1.9
GCL	Isl2b	RGC	<b>-74.5 ± 1.0</b>	-98.68 ± 0.24	-24.2 ± 1.1	-22.74
GCL	Pax6	Displaced am.	-101.93 ± 0.96	<b>-77.87 ± 0.27</b>	24.06 ± 1.0	24.2
GCL	Isl2b/Pax6	RGC subtype	<b>-97.9 ± 1.0</b>	-113.4 ± 0.74	-15.5 ± 1.3	12.4
INL	Cohort	All INL cells	<b>-184.0 ± 1.4</b>	-474.5 ± 2.7	-290.4 ± 3.1	94.0
INL	Pax6	Amacrine cell	<b>-36.83 ± 0.88</b>	-65.43 ± 0.24	-28.6 ± 0.92	31.3
INL	PKC $\beta$	Bipolar cell	-37.83 ± 0.2	<b>-0.77 ± 0.67</b>	37.06 ± 0.7	53.1
INL	GS	Müller glia	-25.11 ± 0.2	<b>6.7 ± 1.1</b>	31.8 ± 1.1	29.2
INL	HM	Horizontal cell	-31.97 ± 0.27	<b>6.73 ± 0.61</b>	38.7 ± 0.66	58.4
ONL	Cohort	All ONL cells	<b>-201.0 ± 1.5</b>	-335.3 ± 1.5	-134.2 ± 2.1	63.7
ONL	Zpr1	Double cones	<b>-93.7 ± 1.4</b>	-146.91 ± 0.8	-53.2 ± 1.6	33.2

$\log Z$ : logarithm of  $p(D)$ , the marginal likelihood of the data, or model evidence. Largest evidence values bolded.  $\log ZR$ : evidence ratio; positive values in favour of stable model.

Table 12.2: Evidence for combined vs. split models of layer and lineage contribution across the dorso-ventral axis

Layer	Marker	Cell type	Combined D-V $\log Z$	Split D-V $\log Z$	$\log ZR$	$\sigma$ sign.
GCL	Cohort	All GCL cells	<b>-41.64 ± 0.63</b>	-186.04 ± 0.39	144.4 ± 0.74	194.1
GCL	Isl2b	RGC	<b>-47.58 ± 0.84</b>	-69.6 ± 1.0	22.0 ± 1.3	16.7
GCL	Pax6	Displaced am.	<b>-32.65 ± 0.68</b>	-54.3 ± 1.5	21.7 ± 1.6	13.4
GCL	Isl2b/Pax6	RGC subtype	<b>-40.94 ± 0.77</b>	-90.3 ± 1.2	49.3 ± 1.4	35.5
INL	Cohort	All INL cells	<b>-83.5 ± 1.0</b>	-124.2 ± 1.1	40.7 ± 1.5	26.5
INL	Pax6	Amacrine cell	<b>-15.66 ± 0.041</b>	-56.15 ± 0.91	40.49 ± 0.91	44.4
INL	PKC $\beta$	Bipolar cell	<b>-10.52 ± 0.46</b>	-23.3 ± 1.1	12.8 ± 1.2	11.1
INL	GS	Müller glia	<b>13.69 ± 0.36</b>	3.5 ± 2.4	10.2 ± 2.4	4.2
INL	HM	Horizontal cell	<b>9.81 ± 0.37</b>	-10.1 ± 1.3	19.9 ± 1.4	14.4
ONL	Cohort	All ONL cells	<b>-73.06 ± 0.84</b>	-141.3 ± 1.1	68.2 ± 1.4	48.0
ONL	Zpr1	Double cones	<b>-41.17 ± 0.75</b>	-63.0 ± 0.89	21.8 ± 1.2	18.8

$\log Z$ : logarithm of  $p(D)$ , the marginal likelihood of the data, or model evidence. Largest evidence values bolded.  $\log ZR$ : evidence ratio; positive values in favour of combined model.

## 12.4 Likelihood ratio calculations for Normal and Log-Normal models of layer and lineage contribution

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 5.7](#) 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.

#### 13.1.3 Morpholino injections

Morpholino injections were performed by Maria Augusta Sartori by methods previously described [[Won15](#)].

#### 13.1.4 Morphological photography

The photographs presented in [Figure 5.1](#) were obtained with a Zeiss fluorescent stereoscope (SteREO Lumar.V12) and 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 5.2](#) 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 5.4](#).

### 13.1.6 BrdU pulse-chase assays

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

### 13.1.8 Progenitor identity marker immunohistochemistry

### 13.1.9 Caspase-3 immunohistochemistry

Caspase-3 immunohistochemistry was performed by Monica Dixon.

### 13.1.10 Pax6 immunohistochemistry

### 13.1.11 In situ hybridization

### 13.1.12 Electron microscopy

### 13.1.13 RT-PCR analysis of *rys* npat expression

### 13.1.14 *rys* MNase digestions and NGS

### 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]. The pipeline is depicted in figureautoref. Node settings are largely default and are available in the `knime_workspace` 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: 'AGATCGGAAGAGC' (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 5.14 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 16.9.23, for the refinement task in Section 16.9.22. EM execution code is found in Section 16.9.24 and Section 16.9.26. Analysis code is given in Section 16.9.25 and Section 16.9.21.

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

## 13.2 Supplementary figures

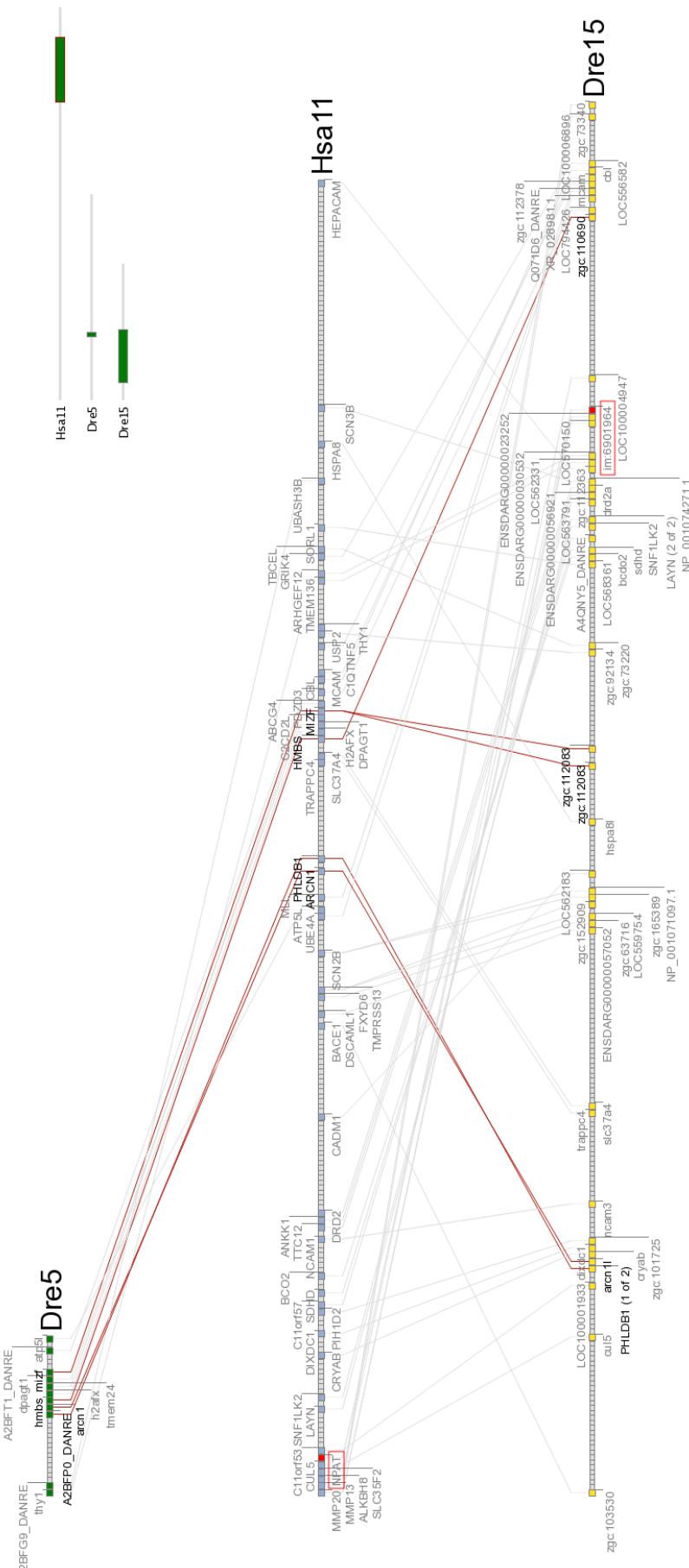


Figure 13.1: **Synteny Database output for the syntenic region containing *D. rerio* npat**

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.

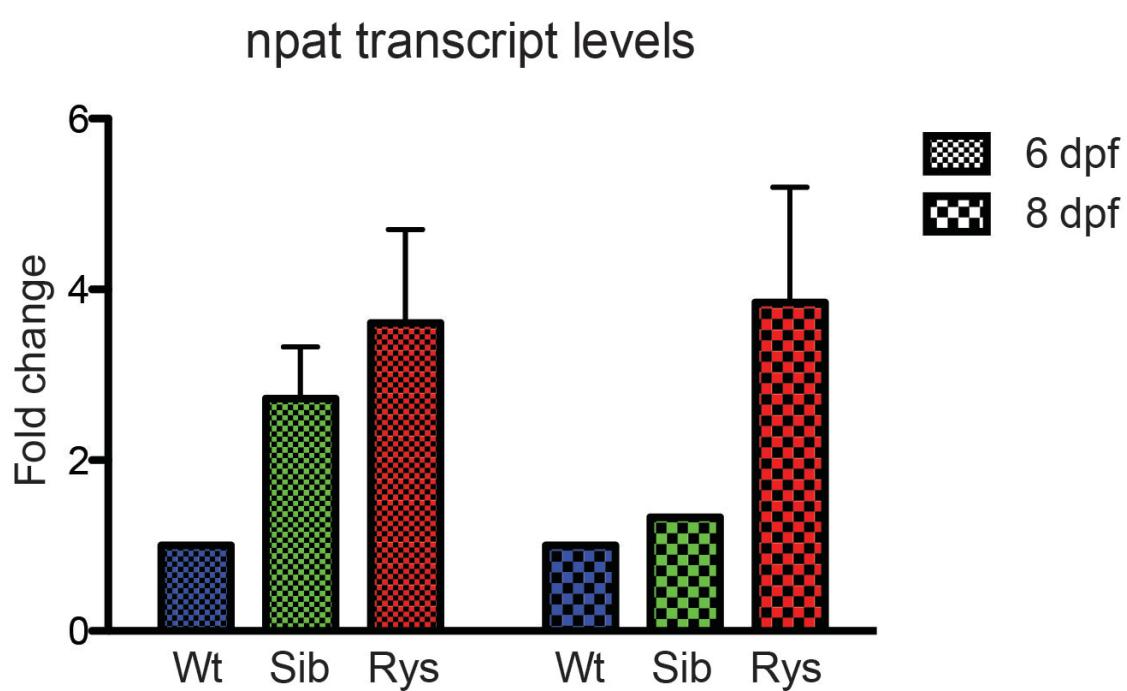


Figure 13.2: npat is overexpressed in rys

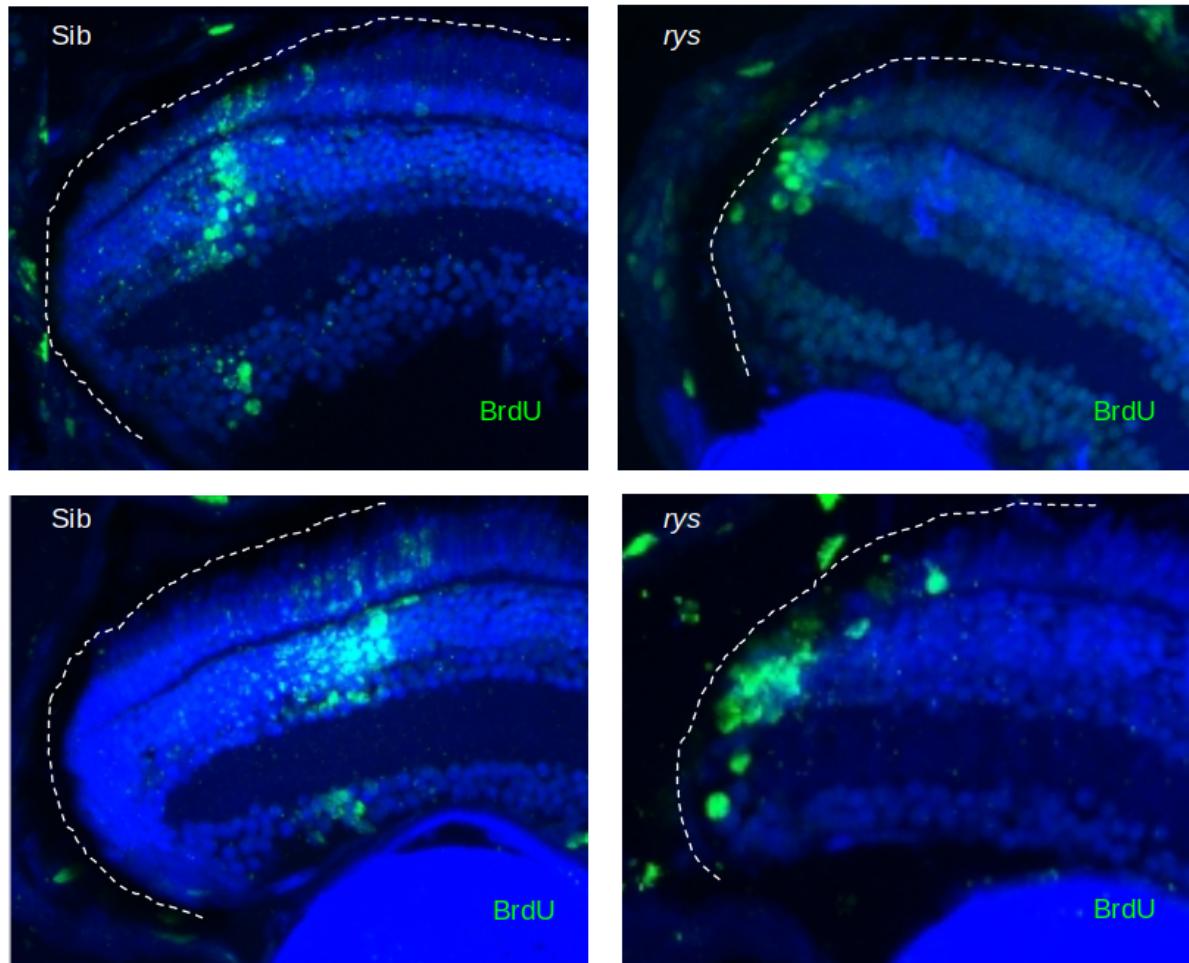


Figure 13.3: 10dpf mutant *rys* RPCs are mitotic

MIP 14 $\mu$ m coronal cryosections through representative sib (left panels) and *rys* eyes at 10dpf, 7 days after an 8hr BrdU pulse at 3dpf. Note that few labelled *rys* cells have entered the specified retinal layers.

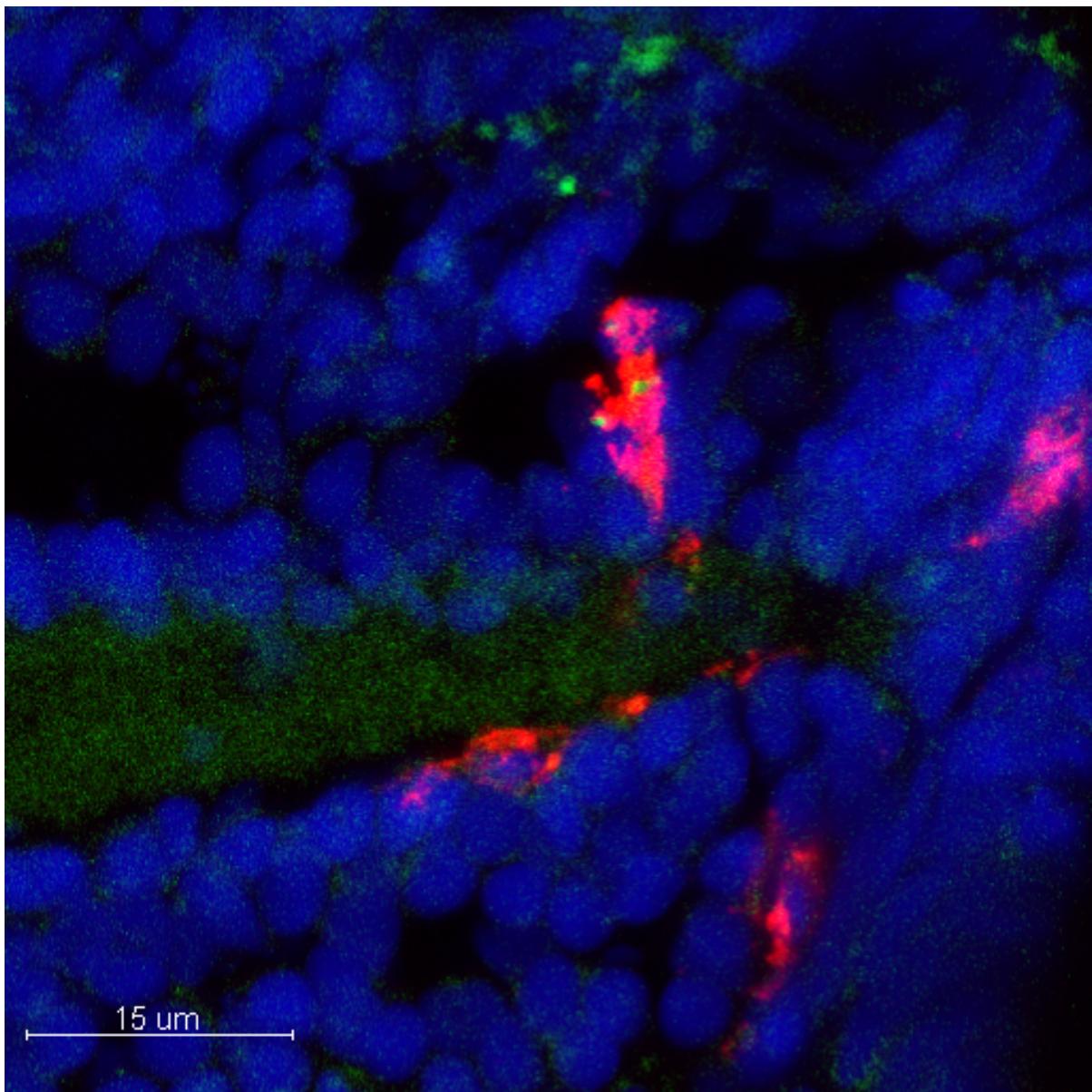


Figure 13.4: 4C4-positive microglia engulf *rys* mutant RPCs

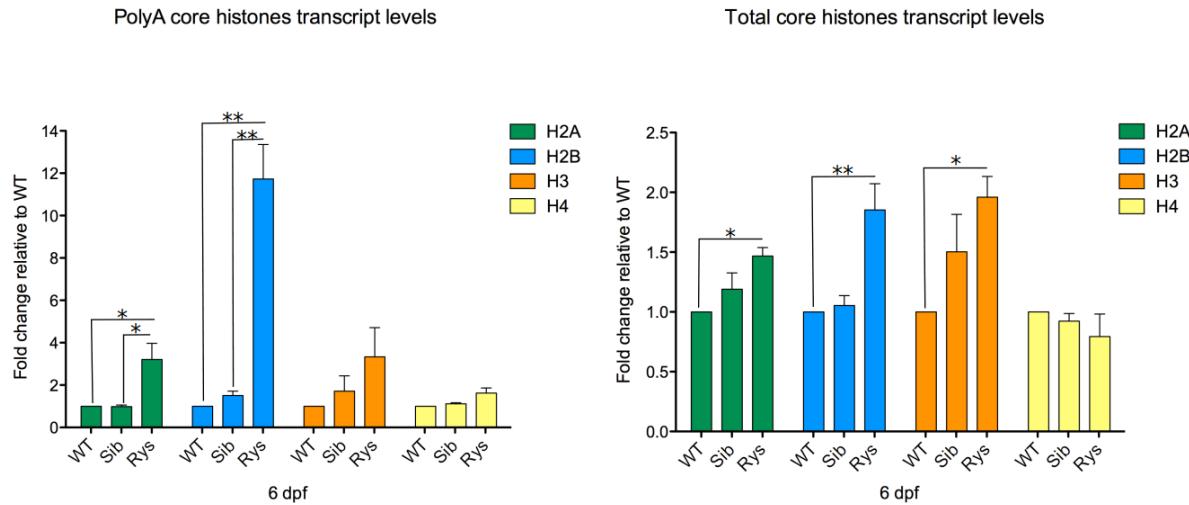


Figure 13.5: Morpholinos directed to npat result in histone transcript overexpression similar to *rys*

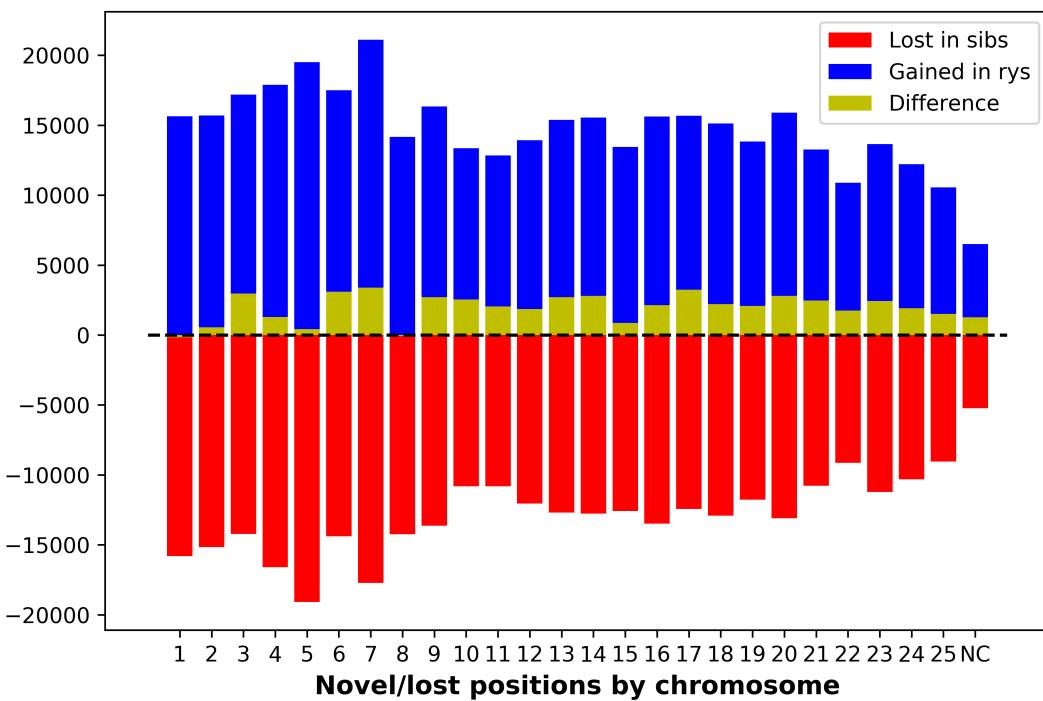
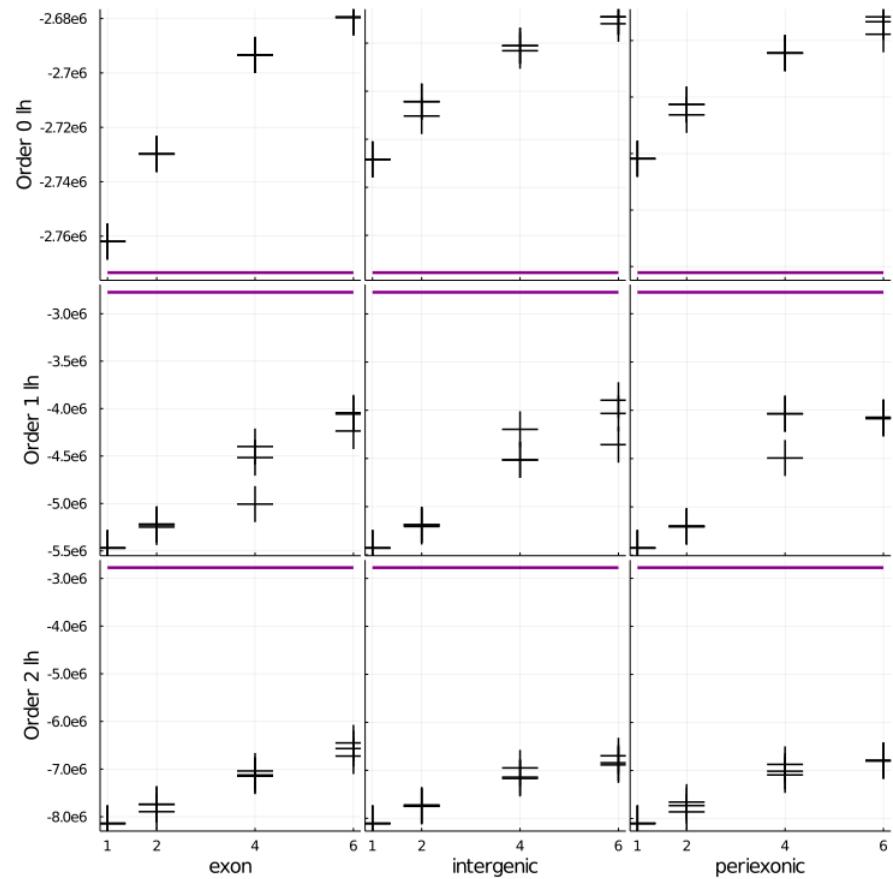
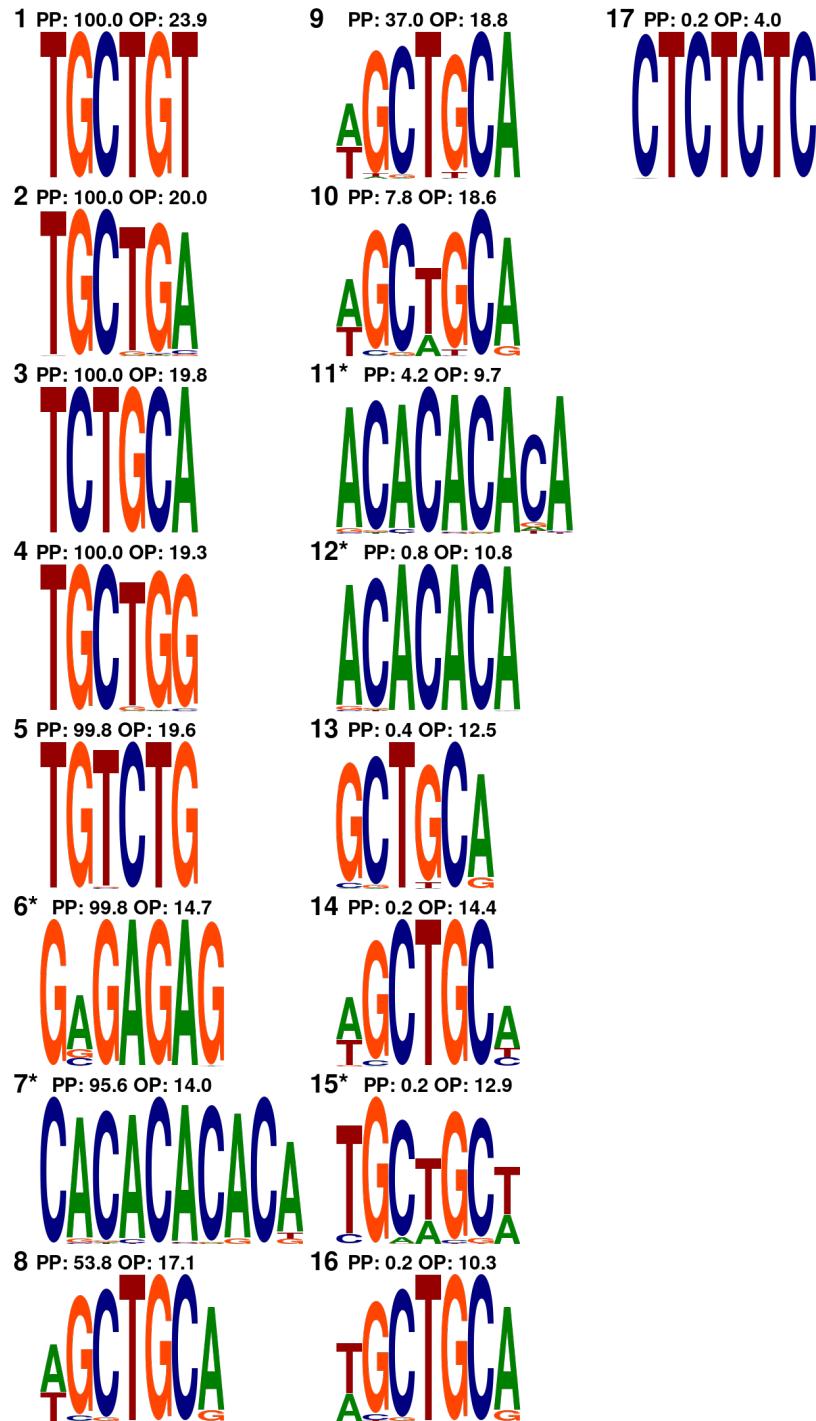


Figure 13.6: Novel nucleosome positions in *rys* occur in similar numbers to those lost from sibs.

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 13.7: Test likelihoods for background HMMs trained on *D. rerio* genome partitions



**Figure 13.8: 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 Normal vs. Log-Normal models of layer and lineage contribution

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.

# Chapter 14

## Theoretical Appendix A: Model theory and statistical methods

### 14.1 Model Theory

This thesis is mainly concerned with a particular kind of scientific model: those that are mathematically expressible, and therefore tractable subjects for numerical simulation techniques. Most biological models do not fit this description, but increasing computing power has greatly 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. Sometimes, simple heuristics are adequate biological explanation; much of differential diagnosis consists of reliable descriptive relations between qualitatively characterised signs in patients, and the presence of a pathogen. 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 make reference to the use of the model in practical domains where the failure of the model to reflect reality results in its disuse. 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 and handwaving. This type of “model comparison” necessarily devolves into the type of anarchic Feyerabendian discursive chaos discussed in [Section 15.2.1](#). Feyerabend is correct to point 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 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, like Feyerabend, we abandon the study and practice of science for poetry.

The support of complex biological systems for multiple explanations often gives rise to calls for explanatory pluralism [[Bri10](#)]. It should be emphasized that very differently parameterised models can be adequate explanations for the same system. This is particularly common for explanations at different descriptive depths. For instance, the models used in [Chapter 4](#) make no reference to particular macromolecules, but 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. 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, and crystallographic studies of the same phenomenon, are 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 usually result in descriptive chaos [Res00, p.65-6]. Rescher explains how the descriptive adequacy of some model in its local domain usually requires the airbrushing out of at least some pertinent details of the system that could have been included; it can be added that the computational tractability of the model requires the same.

If we virtually all accept this form 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 scientific rationation. We cannot accept logically contradictory notions about the structure of reality without precluding cognitive harmony arising from a broadly consistent view of the way the world works [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 the phenomena they represent, as encoded by observational datasets. This requires the selection of a loss or likelihood function. Loss functions, like AIC, used in Chapter 4, express the relative amount of information in the dataset lost by the model, given a set of parameters. Likelihood functions, used elsewhere, express the likelihood of the data given the model and parameters. The parameters of the model thus define an n-dimensional “parameter space”; the loss or likelihood 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, more likely parameter space locations to sample from, in search of the least lossy/most likely parameterisation of the model (model optimisation); we may derive marginal likelihoods for particular parameter values (parameter estimation), or we may estimate the marginal probability of the model over all sampled parameterisations (evidence estimation). These topics are discussed below.

### 14.1.1 Bayesian Epistemological View on Model Comparison

The analyses presented in the data chapters express two views on how model comparison should be approached. The first, expressed in Chapter 2, is drawn from information theory, while the second, taken up in Chapter 4, is a relatively conventional Bayesian view, albeit with more sophisticated tools and more computing power than has been available to Bayesians in the past. In the same way that frequentist analyses may be expressed as a subset of Bayesian analyses (i.e. they normally seek to express the 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 overcome the limitations of the maximum-likelihood approach by using both the maximum-likelihood value (or maximum a priori score, in the case of BIC), and the number of parameters, to produce a score

---

<sup>1</sup>That is, they have incompatible metaphysical content; they are therefore subject to counterinduction as explained in Section 15.2.1

which penalizes models with more free parameters.

The general approach of optimizing a model for a loss function against a dataset, then penalizing the best model loss score by the parameterisation of the model, allows us to overcome the most important problem with frequentist approaches to model selection: the requirement for models to be parametrically nested. Model nesting fundamentally precludes [counter-induction](#); we cannot compare the adequacy of models which express different views of how the described system is parameterised, only whether adding more parameters improves a particular view. Escaping this limitation is what allows us to compare stochastic and deterministic mitotic mode models in [Chapter 2](#); these models express fundamentally different views of how reality is organised. Still, in important ways, this approach shares the basic problem of simply calculating the MLE: the score in no way accounts for the relative 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, will appear to be a better explanation than a model which is a broadly good description over the whole parameter space. Moreover, simply using the number of parameters to penalize the best model found in some sampling procedure fails to express the extent to which the inclusion of those parameters is justified 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 reveals how it specifies the 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 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,  $Pr(data|x)$ , given the proposition, 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 posterior probability of a model, 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:

$$Pr(x)Pr(data|x) = Pr(data)Pr(x|data)$$

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, ( $Pr(data|x)$ ). In return we receive, 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 this proposition about the data is 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

---

<sup>2</sup>If nested sampling is not *the* completion of the Bayesian system, it is at least *a* complete Bayesian system.

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 credibility of the model ( $Pr(data)$ ), ie. a better model gives higher marginal probability to observations than a worse one), 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. A 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. 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.

This view dispenses with typical frequentist 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 refuted in [Section 15.1](#)). 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.

### 14.1.2 Model sampling and optimization

There are now a fairly large number of widely-used techniques intended to sample the parameter space of a statistical model, given an objective function which scores the model. These have a variety of purposes. Most 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, with 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 engineering domains for control systems, where one wishes to maintain the consistency of some process by estimating the inputs most likely to achieve a setpoint, given a process model, for instance. More sophisticated applications of sampling involve the estimation of the distribution of objective function values over the parameter space, rather than focusing solely on finding the optimal value.

The process of model sampling requires a type of algorithmic logic additional to the objective function, which is a means to generate parameter vectors to be supplied to the function. Most effective sampling methods operate by performing an initial sample from some type of prior information<sup>3</sup>, and then iteratively generating new proposals from these initial ones. This iterative proposal generation

---

<sup>3</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 an algorithmically defined prior distribution.

forms a chain of linked positions within parameter space. Generally, new proposals are made considering information related only to the last accepted proposal<sup>4</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 success of any sampling procedure depends critically on the properties of these algorithms. Most of these offer a physical interpretation of the surface formed by the objective function in parameter space. Basic random-walk proposal generation, like the original Metropolis-Hastings implementations of MCMC, is generally considered to be 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”. By proposing a new location up- or down-slope, for objective functions that are maximized or minimized, respectively, 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 fix problems with random-walk proposal generation by mechanical interpretations of proposal movement through parameter space, like Hamiltonian Monte Carlo, or thermodynamic interpretations, like simulated annealing. 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.

### 14.1.3 Overfitting and underfitting

Overfitting is among the most common problems in optimization and statistical estimation. This problem typically arises 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 gives the appearance of a model which is 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

A second source of overfitting is the failure to properly account for uncertainty and prior information. For instance, if a set of 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) is, typically, an overfit to the data. In this simple context, this usually means that the MLE Normal is overly influenced by “outliers”. Arbitrary outlier exclusion is one possible response to this problem; as this thesis demonstrates, this is unnecessary, given 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 serve this comparative role).

---

<sup>4</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 probable than the last one.

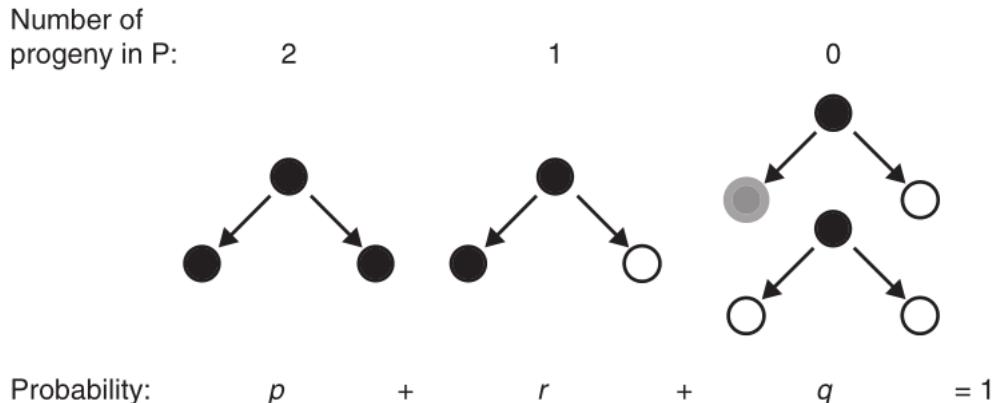


Figure 14.1: Simple stochastic stem cell model, representing probabilities of cell division events, excerpted from Fagan 2013 pg. 61. Black circles denote proliferative cells, while white and grey circles denote different types of postmitotic offspring. “Number of progeny in P” is the number of mitotic 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.

There is a further sense in which models may be overfit to data, even given full Bayesian accounting for prior information and posterior uncertainty. This arises, in particular, from the use of uninformative prior distributions in cases where they are not especially appropriate. This thesis makes extensive use of uninformative priors. In some cases, this is well-justified. The PWM source ICA models

#### 14.1.4 Monte Carlo simulations

Monte Carlo refers to the repeated sampling of some statistical model in order 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, which is the iterative, sequential sampling of parameter space. The second use is the estimation of variability in model outcomes, which involves repetitively sampling the same position in parameter space. I

#### 14.1.5 Simple Stochastic Models

The Simple Stochastic Model is schematically summarised in Figure 14.1. This is the basic structure of the great 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 divisions can be categorised by the number of progeny which remain mitotic after the division. It is important to note that a mitotic event cannot presently be categorized in this fashion except retrospectively. This must be kept in mind when analysing 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.

In effect, then, 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 particular cells as “stem cells”, this may make models derived from the SSM inappropriate for proliferative populations with a large degree of heterogeneity. One may think here of the classic idea of a small number of slowly proliferating “true” stem cells and a larger population of rapidly dividing “transit amplifying” progenitors- this type of internal structure within the proliferating population can only be represented by multiple, 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>5</sup>[Fag13, p.60]

Typically, this type of model has been employed to describe population dynamics of proliferating cells in assays generating ostensibly *clonal* data, where a “clone” here refers to the population constituted by all of the offspring descended from some particular (usually “initial” and sometimes therefore taken for “stem”) proliferative cell. This population is the *lineage* generated by some particular dividing cell. The mitotic events which give rise to a lineage are represented as a Galton-Watson branching process, a stochastic process originally intended to model 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 each cell within the lineage, with this mode being 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.

### 14.1.6 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 define the value of some model output value at a time  $t$ , given the value(s) of the output(s) at some time  $t - p$ ,  $p$  time units in the past, plus the parameter vector for the model. 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 value which is generated by 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 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

---

<sup>5</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.

law model of lens growth.

#### 14.1.7 Independent Component Analysis models of sequence emission

The models of *rys* nucleosome position sequence emission presented in Chapter 5 are of a form that is more familiar to psychologists and signal processing engineers than molecular biologists. Independent Component Analysis (ICA) is 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. 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. 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 a model susceptible to ICA is made both conceptually, and at the level of the numerical representation of these parts of the overall channel. I have closely followed the early, pioneering example of Down et al. [DH05] in this regard. Down et al. use Hidden Markov Models optimized on promoter sequences as background models for promoter sequences in general, against which a foreground of Position Weight Matrix signals, representing binding motifs within the promoter region, is inferred by nested sampling.

#### 14.1.8 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 base emission at the position represented by row  $\lambda^6$ . 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>7</sup>. A computationally useful conjugate prior distribution for discrete categoricals is the Dirichlet distribution [Min00], and 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  $\lambda$ , where 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.

---

<sup>6</sup>This base ordering is computationally significant, since it allows for trivial reverse complementing by reversing the matrix values across both dimensions.

<sup>7</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

### 14.1.9 Hidden Markov Models

## 14.2 Statistical Methods

### 14.2.1 Simultaneous Perturbation Stochastic Approximation (SPSA)

### 14.2.2 Bayesian parameter estimation

#### 14.2.2.1 Problems with frequentist inference using normal models of sample data

Typical biological practice is to report the mean and variance of a sample, assuming a normal distribution of the error around the mean. In other words, the sample is taken to be representative of a larger population; that population is modelled by a normal distribution with mean  $\mu$  and variance  $\sigma^2$ ; the parameters of the maximum likelihood estimate (MLE) for the normal model are the values reported. A slightly more sophisticated approach is to report the standard error of the mean of the sampling distribution the sample is taken to be drawn from, if more than one sample can be obtained, although this usually plays no role in hypothesis testing (often conducted by t-test).

This approach has a number of defects which follow from one another. We are reporting MLE parameters without any account of our uncertainty about those parameters. There is no way to incorporate prior information we have about the parameters (even just to admit total ignorance about them). This leads to *overfitting* of our estimates to the sample data. Practically speaking, this means our estimate of the mean is stated too precisely, and the variance is too sensitive to outliers.

Additionally, the plain-sense interpretation of the estimates are often unclear. Means are usually reported plus-minus variance,  $\mu \pm \sigma$ , and  $\sigma$  is often erroneously interpreted as uncertainty about  $\mu$  rather than an estimate of a second parameter, the variance of the normal population model. If the frequentist confidence interval for  $\mu$  is reported, it is explicitly not understood as the interval in which we have e.g. 95% confidence that  $\mu$  lies, but rather as the interval in which, in the case we repeat the experiment indefinitely,  $\mu$  will be found in 95% of samples. Hypothesis tests are given similarly confusing interpretations involving long-run repeated experiments. These interpretations are widely, if not ubiquitously, misunderstood or ignored in favour of technically incorrect but comprehensible ones [?, ?].

#### 14.2.2.2 The Bayesian approach to normal models of unknown mean and variance

Bayesian methods rectify these problems by understanding the normal model as a model of our information about the population and not of the population itself. This epistemological view of statistics is explicated in ?. Normal gaussian distributions 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 the population. However, we wish to express our uncertainty about the parameters of a normal gaussian distribution by giving further distributions over the mean  $m$  and variance of the normal distribution, with variance usually expressed as precision,  $\lambda = 1/\sigma^2$ . Typically, this is done by assuming normally distributed uncertainty on  $\mu$  and gamma distributed uncertainty on  $\lambda$ , giving rise to a joint normal-gamma (NG) distribution [?]:

An NG distribution may thus serve as a model of our prior information about the population being measured. Because my estimates are the first ones I have made about the relevant populations, and I

have no specific guide as to the actual numbers of cells to expect, I have chosen to use the uninformative NG prior:

$$p(m, \lambda)$$

lies within that range, but is rather understood as the probability that, if the experiment were repeated indefinitely, 95

The appropriateness of the normal model is often in question because it is taken to represent some actually-existing population (which are often not well modelled by normal gaussians). Comparisons of these models using t-tests are given complex interpretations involving long-run rates of error

In Bayesian statistics, available information about a parameter is often modelled by a gaussian distribution over possible values of the parameter.

#### 14.2.3 Empirical Bayes linear regression

#### 14.2.4 Expectation Maximization optimization of HMMs

#### 14.2.5 Galilean Monte Carlo

#### 14.2.6 Nested sampling

## Chapter 15

# Theoretical Appendix B: Metaphysical Arguments

### 15.1 Chance is not a valid explanation for biological phenomena; Randomness is a measure of property of sequences and not an explanation

Substantial portions of this document are dedicated to an examination of 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. It may not be immediately obvious why this should be the case; 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 consider the classic 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 whether they refer 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 possible senses in which we may legitimately describe a process as "stochastic": referring to an epistemic dimension (we may describe some process as stochastic because we are unable to predict its outcome *a priori*), or referring to an ontological dimension (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. In general, we may say that stochasticity, for Harris, applies to at least the following entities:

1. The population-level phenomenal outcome 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

It is, moreover, hardly fair to expect Harris 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 convoluted case of *model overfitting*, if the “stochasticity” in question does not have a material biological referent. Before considering this, we need to define our terms more carefully to avoid the pervasive confusion mentioned above.

#### 15.1.0.1 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, with reference to Kolmogorov complexity:

Chance is mainly 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 multiple instances of some process. It turns out to be challenging merely to define what a “random” binary sequence might be (perhaps generated by a series of coin flips). However, 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.

#### 15.1.0.2 Chance in molecular mechanisms

I turn first to consider what it might 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

---

<sup>1</sup>The Kolmogorov complexity of a sequence can be estimated, contrary to common belief[LV08]. Minimum Message Length expresses a similar concept. 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.

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 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]

Examining the example of the coin toss, we find a fairly simple answer to the question posed earlier: 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 therefore 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 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. This is a common method of making covert appeals to the second line of argument for objective chance, which is the existence of orthodox quantum theory. 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. Rather, 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.

#### 15.1.0.3 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. Unfortunately, a significant misrepresentation of the history of quantum theory has given rise to the impression that the Copenhagen theory is the unanimous or best articulation of quantum theory. We must briefly examine this misrepresentation before we can understand whether Eagle’s argument makes sense.

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 (strangely, without any acknowledgement) in another section of the very pages Eagle’s argument appears in, this is a highly partisan and misleading view. Eminent Bohmian theorist Sheldon Goldstein conclusively demonstrates that Bell was an advocate of the 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 reason to prefer the Copenhagen approach, given that the Bohmian theory explains Copenhagen’s paradoxical results “without further ado”[Gol17]<sup>3</sup>, deals with empirically verified phenomena of physical and biological interest that Copenhagen does not (eg. electron tunneling), and was the preferred approach of the man who likely understood better than anyone his own results, J.S. Bell.

Therefore, Eagle’s argument is incorrect. There is no reason to suppose that the existence of quantum theoretical models that posit objective chance is good evidence for the *reality* of objective chance.

---

<sup>3</sup>Bizarrely, Copenhagen partisans claim that Bohmian mechanics is formally equivalent to the Copenhagen approach. If this is the case, chance is *clearly* a function of model choices and not of any underlying ontological reality. However, Bohmian mechanics is, in fact, substantially more complete than its Copenhagen equivalent, which, by Eagle’s (defective) logic, suggests reality is more likely to reflect the deterministic rather than the chancy approach.

Moreover, there are good reasons to suppose that the converse is true. In sum, then, we may say that there is no reason to consider Harris' argument to refer to *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.

#### 15.1.0.4 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 must ask whether these sequences meet any reasonable technical definition of "random".

Harris' own model proves that RPC fate outcomes are not algorithmically random. 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 *axiomatically assume* a tripartite temporal structure to the differentiation process<sup>4</sup>. This is precisely the type of sequential bias which allows efficient compression of a non-random sequence of outcomes by an algorithm. Therefore, Harris himself concedes that RPC fate specification is not an algorithmically random process<sup>5</sup>.

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. It has long been recognised that classical and quantum physical systems which have algorithmically random initial conditions do not spontaneously evolve to a state of order. 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?"

#### 15.1.0.5 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 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:

---

<sup>4</sup>That is, an early bias in RPC production is produced in these models by the a priori commitment to a "rule" which results in early RPC production.

<sup>5</sup>Having debunked the lay sense of "random" being equivalent to "chancy" above, there is no reason to consider these other, confused, non-technical definitions of randomness. They are neither logically defensible nor practically quantifiable, and therefore are of no scientific interest.

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>6</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 stochastic modelling techniques should not be taken to suggest that on some level 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 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, and explain departures from pure incompressibility.

## 15.2 Macromolecular mechanistic explanations in the Systems era

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 particular 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 (allegedly) sophisticated numerical techniques into our laboratories, only to have their expert 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

---

<sup>6</sup>That is, the mean and the variance of the Normal Gaussian model are said to determine the process being modeled.

been slain by the flurry of t-tests and ANOVAe 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 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 interventions are effective at all [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 15.1.

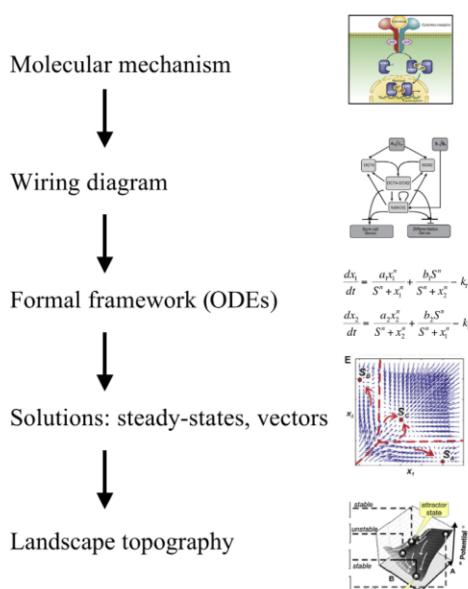


Figure 15.1: Cellular systems model-construction, excerpted from [Fag15, p.7].

The system of equations and subsequent steps are based on a 2-element wiring diagram.

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].

Moreover, much of the sophisticated literature on mathematical models of stem cell variability are based on im-

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 so concerned with. Other approaches rely on control systems theory [SK15, YSK15], many other gaily 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 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*.

### 15.2.1 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, highlighting the sense in which the entities to which the Stochastic Mitotic Mode Explanation (SMME) refer become progressively more abstract and restricted to the mitotic mode concept. Mitotic mode is not a physical existent, but rather an abstraction compounding 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 Paul Feyerabend’s *Against Method* is that the observed historical succession of scientific theories occurs

by counterpositional advancement of incompatible opposing theories, because 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. 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>7</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 dishonesty and 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. Similar points can be made about aspects of Einstein or Bohr's scientific programmes, both scientists viewing themselves as outsiders attacking an established orthodoxy.

Because of his concern to attack what he sees as the overweening social influence of scientific and technical experts, Feyerabend puts a great deal of emphasis on the discursive process of establishing orthodoxy within a field, and the sense in which this is common to scientific and nonscientific domains. That is, the practice of the natural sciences are susceptible to the same irreducibly subjective and historically contingent social, political, economic, etc. forces as all other domains of human culture, and 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

---

<sup>7</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.

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. He was also wrong that there is nothing particular to the practice of natural sciences that is not present in other domains of human activity; statistical descriptions of collections of outcomes and of uncertainty, and a self-reinforcing dynamic of scaffolding methodological complexity, are 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 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 mendacious insistence on the actual reality of chanciness by Copenhagen theorists, and the broken and illogical statistical approaches commanded 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 show up the fundamental problems with Harris' theory.

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 16

## Code Appendix

### 16.1 Code and Output Archive

### 16.2 SMME

#### 16.2.1 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.
```

```
20
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', ''}
```

```
56     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)
```

```
87
88     # Amend the components
88     components_list = []
```

```
89
```

```

90     if ask_for_response("Does this project depend on the cell_based
91         ↵ component? [Y/n] "):
92             components_list.append('cell_based')
93
94     if ask_for_response("Does this project depend on the crypt component?
95         ↵ [Y/n] "):
96             components_list.append('crypt')
97
98     if ask_for_response("Does this project depend on the heart component?
99         ↵ [Y/n] "):
100            components_list.append('heart')
101
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
108
109     if __name__ == "__main__":
110         main()

```

---

### 16.2.2 /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     }
49     return exit_code;
50 }
51
52 /*****
53 * SIMULATOR PARAMETERS
54 *****/
55
56 std::string directoryString, filenameString;

```

```

47     int outputMode; //0 = counts; 1 = mitotic mode events; 2 = mitotic
48         ↵ mode sequence sampling
49     bool debugOutput;
50     unsigned startSeed, endSeed, endGeneration, phase2Generation,
51         ↵ phase3Generation;
52     double pAtoh7, pPtf1a, png; //stochastic model parameters

53
54     //PARSE ARGUMENTS
55     directoryString = argv[1];
56     filenameString = argv[2];
57     outputMode = std::stoi(argv[3]);
58     debugOutput = std::stoul(argv[4]);
59     startSeed = std::stoul(argv[5]);
60     endSeed = std::stoul(argv[6]);
61     endGeneration = std::stoul(argv[7]);
62     phase2Generation = std::stoul(argv[8]);
63     phase3Generation = std::stoul(argv[9]);
64     pAtoh7 = std::stod(argv[10]);
65     pPtf1a = std::stod(argv[11]);
66     png = std::stod(argv[12]);

67     /*****
68     * PARAMETER/ARGUMENT SANITY CHECK
69     *****/
70
71     bool sane = 1;

72     if (outputMode != 0 && outputMode != 1 && outputMode != 2)
73     {
74         ExecutableSupport::PrintError(
75             "Bad outputMode (argument 3). Must be 0 (counts) 1
76                 ↵ (mitotic events) or 2 (sequence sampling)");
77         sane = 0;
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     if (endGeneration <= 0)
86     {

```

```
86     ExecutableSupport::PrintError("Bad endGeneration (argument 7).  
87         ↪   endGeneration must be > 0");  
88     sane = 0;  
89 }  
90  
91 if (phase3Generation < phase2Generation)  
92 {  
93     ExecutableSupport::PrintError(  
94         "Bad phase2Generation or phase3Generation (arguments 8,  
95             ↪   9). phase3Generation must be > phase2Generation. Both  
96             ↪   must be >0");  
97     sane = 0;  
98 }  
99  
100 if (pAtoh7 < 0 || pAtoh7 > 1)  
101 {  
102     ExecutableSupport::PrintError("Bad pAtoh7 (argument 10). Must be  
103         ↪   0-1");  
104     sane = 0;  
105 }  
106  
107 if (pPtf1a < 0 || pPtf1a > 1)  
108 {  
109     ExecutableSupport::PrintError("Bad pPtf1a (argument 11). Must be  
110         ↪   0-1");  
111     sane = 0;  
112 }  
113  
114 if (png < 0 || png > 1)  
115 {  
116     ExecutableSupport::PrintError("Bad png (argument 12). Must be  
117         ↪   0-1");  
118     sane = 0;  
119 }  
120  
121 if (sane == 0)  
122 {  
123     ExecutableSupport::PrintError("Exiting with bad arguments. See  
124         ↪   errors for details");  
125     exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;  
126     return exit_code;  
127 }  
128  
129  
130  
131
```

```

122 //*****
123 * SIMULATOR OUTPUT SETUP
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

```

```
160  
161     //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
```

```

197     //Generate 1x1 mesh for single-cell colony
198     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     //Setup simulator & run simulation
207     boost::shared_ptr<OffLatticeSimulationPropertyStop<2>>
208         ↪ p_simulator(
209             new
210                 ↪ OffLatticeSimulationPropertyStop<2>(*cell_population));
211     p_simulator->SetStopProperty(p_Mitotic); //simulation to stop if
212         ↪ no mitotic cells are left
213     p_simulator->SetDt(0.25);
214     p_simulator->SetEndTime(endGeneration);
215     p_simulator->SetOutputDirectory("UnusedSimOutput" +
216         ↪ filenameString); //unused output
217     p_simulator->Solve();

218     //Count lineage size
219     unsigned count = cell_population->GetNumRealCells();

220
221     if (outputMode == 0) *p_log << entry_number << "\t" << seed <<
222         ↪ "\t" << count << "\n";
223     if (outputMode == 2) *p_log << "\n";

224     //Reset for next simulation
225     SimulationTime::Destroy();
226     delete cell_population;
227     entry_number++;

228     if (debugOutput)
229     {
230         ↪ debugWriter->Close();
231     }

232 }
233
234     p_RNG->Destroy();

```

```

234     LogFile::Close();
235
236     return exit_code;
237 }
238 ;

```

---

### 16.2.3 /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      exit_code = ExecutableSupport::EXIT_BAD_ARGUMENTS;
49      return exit_code;
50  }

51  //*****
52  * SIMULATOR PARAMETERS
53  *****/
54  std::string directoryString, filenameString;
55  int outputMode; //0 = counts; 1 = mitotic mode events; 2 = mitotic
56  → mode sequence sampling
57  bool debugOutput;
58  unsigned startSeed, endSeed;
59  double endTime;
60  double normalMu, normalSigma, pPP, pPD, pBC, pAC, pMG; //stochastic
61  → model parameters

62  //PARSE ARGUMENTS
63  directoryString = argv[1];
64  filenameString = argv[2];
65  outputMode = std::stoi(argv[3]);
66  debugOutput = std::stoul(argv[4]);
67  startSeed = std::stoul(argv[5]);
68  endSeed = std::stoul(argv[6]);
69  endTime = std::stod(argv[7]);
70  normalMu = std::stod(argv[8]);
71  normalSigma = std::stod(argv[9]);
72  pPP = std::stod(argv[10]);
73  pPD = std::stod(argv[11]);
74  pBC = std::stod(argv[12]);
75  pAC = std::stod(argv[13]);

```

```

67     pMG = std::stod(argv[14]);
68
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    }

```

```

105
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 //Set up singleton LogFile
125    LogFile* p_log = LogFile::Instance();
126     p_log→Set(0, directoryString, filenameString);
127
128 ExecutableSupport::Print("Simulator writing file " + filenameString +
129     ↪    " to directory " + directoryString);
130
131 //Log entry counter
132     unsigned entry_number = 1;
133
134 //Write appropriate headers to log
135     if (outputMode == 0) *p_log << "Entry\tSeed\tCount\n";
136     if (outputMode == 1) *p_log << "Time (hpf)\tSeed\tCellID\tMitotic
137         ↪    Mode (0=PP;1=PD;2=DD)\n";
138     if (outputMode == 2) *p_log << "Entry\tSeed\tSequence\n";
139
140 //Instance RNG
141     RandomNumberGenerator* p_RNG = RandomNumberGenerator::Instance();

```

```

141 //Initialise pointers to relevant singleton ProliferativeTypes and
142   ↳ Properties
143     MAKE_PTR(WildTypeCellMutationState, p_state);
144     MAKE_PTR(TransitCellProliferativeType, p_Mitotic);
145     MAKE_PTR(DifferentiatedCellProliferativeType, p_PostMitotic);
146     MAKE_PTR(RodPhotoreceptor, p_RPh_fate);
147     MAKE_PTR(AmacrineCell, p_AC_fate);
148     MAKE_PTR(BipolarCell, p_BC_fate);
149     MAKE_PTR(MullerGlia, p_MG_fate);
150     MAKE_PTR(CellLabel, p_label);

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;

```

```

179     }
180
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();

```

```

213
214     //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
232     p_RNG→Destroy();
233     LogFile::Close();
234
235     return exit_code;
236 }
237 ;

```

---

#### 16.2.4 /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 }
```

```
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  
108 if (fixture != 0 && fixture != 1 && fixture != 2)  
109 {  
110     ExecutableSupport::PrintError("Bad fixture (argument 5). Must be  
111         ↳ 0 (He), 1 (Wan), or 2 (validation/test)");  
112     sane = 0;  
113 }  
114  
115 if (ath5founder != 0 && ath5founder != 1)  
116 {  
117     ExecutableSupport::PrintError("Bad ath5founder (argument 6). Must  
118         ↳ be 0 (wild type) or 1 (ath5 mutant)");  
119     sane = 0;  
120 }  
121  
122 if (endSeed < startSeed)  
123 {  
124     ExecutableSupport::PrintError("Bad start & end seeds (arguments,  
125         ↳ 8, 9). endSeed must not be < startSeed");  
126     sane = 0;  
127 }  
128  
129 if (inductionTime ≥ endTime)  
130 {
```

```

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     if (pPP1 + pPD1 > 1 || pPP1 > 1 || pPP1 < 0 || pPD1 > 1 || pPD1 <
161         ↵ 0)
162     {
163         ExecutableSupport::PrintError(
164             "Bad phase 1 probabilities (arguments 16, 17). pPP1 +
165                 ↵ pPD1 should be ≥0, ≤1, sum should not exceed
166                 ↵ 1");
167         sane = 0;
168     }
169 }
```

```

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         {

```

```

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     }

```

```

268     double currTiL; //Time in Lineage offset for lineages induced
269     ↵ after first mitosis
270     double lineageStartTime; //first mitosis time (hpf)
271     double currSimEndTime; //simulation end time (hpf);

272

273

274     ↵ ****
275     * Time in Lineage Generation Fixtures & Cell Cycle Model Setup
276
277     → ****
278
279     if (fixture == 0) //He 2012-type fixture - even distribution
280     ↵ across nasal-temporal axis
281     {
282         //generate lineage start time from even random distro across
283         ↵ earliest-latest start time figures
284         lineageStartTime = (p_RNG→ranf() * (latestLineageStartTime -
285         ↵ earliestLineageStartTime))
286             + earliestLineageStartTime;
287         //this reflects induction of cells after the lineages' first
288         ↵ mitosis
289         if (lineageStartTime < inductionTime)
290         {
291             currTiL = inductionTime - lineageStartTime;
292             currSimEndTime = endTime - inductionTime;
293             if (outputMode == 1)
294                 ↵ p_cycle_model→EnableModeEventOutput(inductionTime,
295                 ↵ seed);
296         }
297         //if the lineage starts after the induction time, give it
298         ↵ zero & TiL run the appropriate-length simulation
299         //((ie. the endTime is reduced by the amount of time after
300         ↵ induction that the first mitosis occurs)
301         if (lineageStartTime ≥ inductionTime)
302         {
303             currTiL = 0.0;
304             currSimEndTime = endTime - lineageStartTime;
305             if (outputMode == 1)
306                 ↵ p_cycle_model→EnableModeEventOutput(lineageStartTime,
307                 ↵ seed);
308         }
309     }

```

```

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 ;

```

---

### 16.2.5 /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    p_stem_model→EnableExpandingStemPopulation(1, cell_population);  
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 ;
```

### 16.2.6 /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 std::vector<double> stemOffspringParams = { mitoticModePhase2,
82   ↪ mitoticModePhase2 + mitoticModePhase3, pPP1, pPD1,
83   ↪ pPP2, pPD2, pPP3, pPD3,
84   ↪ progenitorGammaShift,
85   ↪ progenitorGammaShape,
86   ↪ progenitorGammaScale,
87   ↪ progenitorGammaSister
88   ↪ };

89 ****
90 * PARAMETER/ARGUMENT SANITY CHECK
91 ****
92 bool sane = 1;

93 if (endSeed < startSeed)
94 {
95     ExecutableSupport::PrintError("Bad start & end seeds (arguments,
96   ↪ 3, 4). endSeed must not be < startSeed");
97     sane = 0;
98 }
99 if (cmzResidencyTime < 0)
100 {
101     ExecutableSupport::PrintError("Bad CMZ residency time (argument
102   ↪ 5). cmzResidencyTime must be positive-valued");

```

```

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
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
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
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
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
1577
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
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
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
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
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
1777
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
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
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
1877
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
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
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
1977
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
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
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
2077
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
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
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
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2247
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
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2295
2296
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2347
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
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
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
24
```



```

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     //Setup simulator & run simulation
244     boost::shared_ptr<OffLatticeSimulationPropertyStop<2>>
245         → p_simulator(
246             new
247                 → OffLatticeSimulationPropertyStop<2>(*cell_population));
248     p_simulator→SetStopProperty(p_Transit); //simulation to stop if
249         → no RPCs are left
250     p_simulator→SetDt(1);
251     p_simulator→SetOutputDirectory(directoryString + "/Seed" +
252         → std::to_string(seed) + "Results");
253     p_simulator→SetEndTime(8568); // 360dpf - 3dpf simulation start
254         → time
255     p_simulator→Solve();
256
257     //Reset for next simulation
258     SimulationTime::Destroy();
259     cell_population.reset();
260 }
261
262 p_RNG→Destroy();
263
264 return exit_code;
265 }
266 ;

```

---

### 16.2.7 /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()

```

---

### 16.2.8 /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()
```

---

### 16.2.9 /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     # Use processes equal to the number of cpus available
442     cpu_count = multiprocessing.cpu_count()
443
444     print("Starting simulations for iterate " + str(k) + " with " +
445           ↳ str(cpu_count) + " processes, " + str(end_seed+1) + " lineages
446           ↳ simulated for counts, " + str(rate_end_seed+1) + " lineages for
447           ↳ events")
448
449     log.flush() #required to prevent pool from jamming up log for some
450           ↳ reason
451
452     # Generate a pool of workers
453     pool = multiprocessing.Pool(processes=cpu_count)
454
455     # Pass the list of bash commands to the pool, block until pool is
456       ↳ complete
457     pool.map(execute_command, command_list, 1)
458
459     #these numpy arrays hold the individual RSS vals for timepoints +
460       ↳ rates
461     rss_plus = np.zeros(len(induction_times)+3)
462     rss_minus = np.zeros(len(induction_times)+3)
463
464     #clear the plots on the interactive figure
465     for i in range(0, len(plot_list)):
466         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 #data plotter function for monitoring SPSA results
537 def plotter(subplot,plus,minus,empirical_prob,samples,mode):
538     global plt0,plt1,plt2,plt3,plt4,plt5, prob_histo_plus,
539         → prob_histo_minus
540     bin_sequence = []
541     x_sequence = []
542     trim_value = 0
543
544     if mode == 0:
545         bin_sequence = count_bin_sequence
546         x_sequence = count_x_sequence
547         trim_value = count_trim_value
548         prob_histo_plus, bin_edges = np.histogram(plus, bin_sequence,
549             → density=True)
550         prob_histo_minus, bin_edges = np.histogram(minus, bin_sequence,
551             → density=True)
552
553     if mode == 1:
554         bin_sequence = rate_bin_sequence
555         x_sequence = rate_x_sequence
556         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()

```

---

### 16.2.10 /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"
88         ↵ WanOutput 18.0 24.0 17.0 10 792 160 4 6.5 4 4 2 1 1 8 15 1.0 0.0
89         ↵ 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()
```

---

### 16.2.11 /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()

```

---

### 16.2.12 /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
94 if __name__ == "__main__":
95     main()

```

---

### 16.2.13 /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 operatinos
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())
42 fit_results = model.params
43
44 tc = 1/fit_results[1]
45 ts = tc * fit_results[0]
46
47 sns.regplot(totals_frame.Time,totals_frame.labelled_fraction)
48
49 plt.text(7,.2,"Tc: " + f'{tc:.2f}')
50 plt.text(7,.1,"Ts: " + f'{ts:.2f}')
51
52 plt.xlabel("Time (h)")
53 plt.ylabel("Fraction of CMZ RPCs labelled by EdU")
54
55 plt.savefig("cumulative_edu.png")
56 png_memory = BytesIO()
57 plt.savefig(png_memory, format='png', dpi=600)
58 PILpng = Image.open(png_memory)
59 PILpng.save('cumulative_edu.tiff')
60 png_memory.close()

```

```
57
58     plt.show()
59
60 if __name__ == "__main__":
61     main()
```

---

### 16.2.14 /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.cs'
    ↵ 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.cs'
    ↵ skiprows=1, usecols=(3,5,8))
59 E0_events = raw_events[np.where(raw_events[:,2]==1)] #exclude any mitosis
    ↵ whose time was too early for recording
60
61 E0_PP = E0_events[np.where(E0_events[:,0]==0)]
62 E0_PD = E0_events[np.where(E0_events[:,0]==1)]
63 E0_DD = E0_events[np.where(E0_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,
   ↳ ath5_mean_clone_size, wt_even_odd_ratio, ath5_even_odd_ratio]

85
86 def main():
87     #LOAD & PARSE HE MODEL OUTPUT FILES
88     #original fit stochastic mitotic mode files
89     o_counts_24 =
       ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
       ↳ skiprows=1, usecols=3)
90     o_counts_32 =
       ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
       ↳ skiprows=1, usecols=3)
91     o_counts_48 =
       ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
       ↳ skiprows=1, usecols=3)
92     o_events =
       ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM
       ↳ skiprows=1, usecols=(0,3))
93     o_events_PP = np.array(o_events[np.where(o_events[:,1]==0)])

```

```
94     o_events_PD = np.array(o_events[np.where(o_events[:,1]==1)])
95     o_events_DD = np.array(o_events[np.where(o_events[:,1]==2)])
96     o_counts_wan_no_shadow =
97         ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
98             ↳ skiprows=1, usecols=3)
99
100    #refit stochastic mitotic mode files
101    s_counts_24 =
102        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
103            ↳ skiprows=1, usecols=3)
104    s_counts_32 =
105        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
106            ↳ skiprows=1, usecols=3)
107    s_counts_48 =
108        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
109            ↳ skiprows=1, usecols=3)
110    s_events =
111        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM
112            ↳ skiprows=1, usecols=(0,3))
113    s_events_PP = np.array(s_events[np.where(s_events[:,1]==0)])
114    s_events_PD = np.array(s_events[np.where(s_events[:,1]==1)])
115    s_events_DD = np.array(s_events[np.where(s_events[:,1]==2)])
116    s_counts_wan_no_shadow =
117        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
118            ↳ skiprows=1, usecols=3)
119    s_counts_ath5 =
120        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
121            ↳ skiprows=1, usecols=3)
122    s_counts_validate =
123        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
124            ↳ skiprows=1, usecols=3)
125
126    #deterministic mitotic mode files
127    d_counts_24 =
128        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
129            ↳ skiprows=1, usecols=3)
```

```

114 d_counts_32 =
115     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
116     ↵ skiprows=1, usecols=3)
117 d_counts_48 =
118     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
119     ↵ skiprows=1, usecols=3)
120 d_events =
121     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM
122     ↵ skiprows=1, usecols=(0,3))
123 d_events_PP = np.array(d_events[np.where(d_events[:,1]==0)])
124 d_events_PD = np.array(d_events[np.where(d_events[:,1]==1)])
125 d_events_DD = np.array(d_events[np.where(d_events[:,1]==2)])
126 d_counts_wan_no_shadow =
127     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
128     ↵ skiprows=1, usecols=3)
129 d_counts_ath5 =
130     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
131     ↵ skiprows=1, usecols=3)
132 d_counts_validate =
133     ↵ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeC
134     ↵ skiprows=1, usecols=3)

135 #calculate Ath5 clone size and even-odd ratios
136 o_wt_mean_clone_size = np.mean(o_counts_validate)
137 o_ath5_mean_clone_size = np.mean(o_counts_ath5)

138 s_wt_mean_clone_size = np.mean(s_counts_validate)
139 s_ath5_mean_clone_size = np.mean(s_counts_ath5)

140 d_wt_mean_clone_size = np.mean(d_counts_validate)
141 d_ath5_mean_clone_size = np.mean(d_counts_ath5)

142 o_wt_even_odd_ratio = len(np.where(o_counts_validate % 2 ==
143     ↵ 0)[0])/len(np.where(o_counts_validate % 2 == 1)[0])
144 o_ath5_even_odd_ratio = len(np.where(o_counts_ath5 % 2 ==
145     ↵ 0)[0])/len(np.where(o_counts_ath5 % 2 == 1)[0])

146 s_wt_even_odd_ratio = len(np.where(s_counts_validate % 2 ==
147     ↵ 0)[0])/len(np.where(s_counts_validate % 2 == 1)[0])
148 s_ath5_even_odd_ratio = len(np.where(s_counts_ath5 % 2 ==
149     ↵ 0)[0])/len(np.where(s_counts_ath5 % 2 == 1)[0])

```

```

140 d_wt_even_odd_ratio = len(np.where(d_counts_validate % 2 ==
141   ↵ 0)[0])/len(np.where(d_counts_validate % 2 == 1)[0])
d_ath5_even_odd_ratio = len(np.where(d_counts_ath5 % 2 ==
142   ↵ 0)[0])/len(np.where(d_counts_ath5 % 2 == 1)[0])

143 o_fit_output_list = [o_counts_24, o_counts_32, o_counts_48,
144   ↵ o_events_PP, o_events_PD, o_events_DD]
o_test_output_list = [o_counts_wan_no_shadow, o_wt_mean_clone_size,
145   ↵ o_ath5_mean_clone_size, o_wt_even_odd_ratio,
146   ↵ o_ath5_even_odd_ratio]

147 s_fit_output_list = [s_counts_24, s_counts_32, s_counts_48,
148   ↵ s_events_PP, s_events_PD, s_events_DD]
s_test_output_list = [s_counts_wan_no_shadow, s_wt_mean_clone_size,
149   ↵ s_ath5_mean_clone_size, s_wt_even_odd_ratio,
150   ↵ s_ath5_even_odd_ratio]

151 d_fit_output_list = [d_counts_24, d_counts_32, d_counts_48,
152   ↵ d_events_PP, d_events_PD, d_events_DD]
d_test_output_list = [d_counts_wan_no_shadow, d_wt_mean_clone_size,
153   ↵ d_ath5_mean_clone_size, d_wt_even_odd_ratio,
154   ↵ d_ath5_even_odd_ratio]

155 #SETUP FIGURES
156 #4 figures to be generated; 3 show output of individual model modes,
157   ↵ 4th overlays stochastic refit on deterministic fit
158 o_fig, ((o_24,o_32),(o_48,o_PP),(o_PD,o_DD),(o_wan_noshad,
159   ↵ o_ath5size),(o_ratio,o_unused)) =
160   ↵ plt.subplots(5,2,figsize=(7.5,8.75))
161 s_fig, ((s_24,s_32),(s_48,s_PP),(s_PD,s_DD),(s_wan_noshad,
162   ↵ s_ath5size),(s_ratio,s_unused)) =
163   ↵ plt.subplots(5,2,figsize=(7.5,8.75))
164 d_fig, ((d_24,d_32),(d_48,d_PP),(d_PD,d_DD),(d_wan_noshad,
165   ↵ d_ath5size),(d_ratio,d_unused)) =
166   ↵ plt.subplots(5,2,figsize=(7.5,8.75))
167 sd_fig, ((sd_24,sd_32),(sd_48,sd_PP),(sd_PD,sd_DD),(sd_wan_noshad,
168   ↵ sd_ath5size),(sd_ratio,sd_unused)) =
169   ↵ plt.subplots(5,2,figsize=(7.5,8.75))

170 #turn off unused axes
171
172   ↵ o_unused.axis('off');s_unused.axis('off');d_unused.axis('off');sd_unused.axis('off')

```

```
162 #set xlabel
163
164     → o_24.set_xlabel('Count');o_32.set_xlabel('Count');o_48.set_xlabel('Count');o_
165     → _PP.set_xlabel('Age (hpf)');o_PD.set_xlabel('Age
166     → (hpf)');o_DD.set_xlabel('Age (hpf)')
167
168
169     → s_24.set_xlabel('Count');s_32.set_xlabel('Count');s_48.set_xlabel('Count');s_
170     → _PP.set_xlabel('Age (hpf)');s_PD.set_xlabel('Age
171     → (hpf)');s_DD.set_xlabel('Age (hpf)')
172
173     → d_24.set_xlabel('Count');d_32.set_xlabel('Count');d_48.set_xlabel('Count');d_
174     → _PP.set_xlabel('Age (hpf)');d_PD.set_xlabel('Age
175     → (hpf)');d_DD.set_xlabel('Age (hpf)')
176
177 #set ylabel
178
179     → o_24.set_ylabel('Probability');o_32.set_ylabel('Probability');o_48.set_ylabel(
180     → o_PP.set_ylabel('PP Probability');o_PD.set_ylabel('PD
181     → Probability');o_DD.set_ylabel('DD Probability');
182     → o_at5size.set_ylabel('Clone size');o_ratio.set_ylabel('Clone
183     → even/odd ratio')
184
185     → s_24.set_ylabel('Probability');s_32.set_ylabel('Probability');s_48.set_ylabel(
186     → s_PP.set_ylabel('PP Probability');s_PD.set_ylabel('PD
187     → Probability');s_DD.set_ylabel('DD Probability');
188     → s_at5size.set_ylabel('Clone size');s_ratio.set_ylabel('Clone
189     → even/odd ratio')
190
191     → d_24.set_ylabel('Probability');d_32.set_ylabel('Probability');d_48.set_ylabel(
192     → d_PP.set_ylabel('PP Probability');d_PD.set_ylabel('PD
193     → Probability');d_DD.set_ylabel('DD Probability');
194     → d_at5size.set_ylabel('Clone size');d_ratio.set_ylabel('Clone
195     → even/odd ratio')
```

```

188     → sd_24.set_ylabel('Probability');sd_32.set_ylabel('Probability');sd_48.set_yla
189     sd_PP.set_ylabel('PP Probability');sd_PD.set_ylabel('PD
     → Probability');sd_DD.set_ylabel('DD Probability');
190     sd_ath5size.set_ylabel('Clone size');sd_ratio.set_ylabel('Clone
     → even/odd ratio')

191 #PLOT DATA
192 #plot lines, CIs, and bars
193 o_objects = line_plotter(o_24, o_counts_24, bin_sequence_24_32,
194     → x_sequence_24_32, count_seeds, lineages_sampled_24,
195     → prob_empirical_24, 'b-', '#000080', '#0000FF', 'y+', 0)
196 line_plotter(o_32, o_counts_32, bin_sequence_24_32, x_sequence_24_32,
197     → count_seeds, lineages_sampled_32, prob_empirical_32, 'b-',
198     → '#000080', '#0000FF', 'y+', 0)
199 line_plotter(o_48, o_counts_48, bin_sequence_48, x_sequence_48,
200     → count_seeds, lineages_sampled_48, prob_empirical_48, 'b-',
201     → '#000080', '#0000FF', 'y+', 0)
202 line_plotter(o_PP, o_events_PP[:,0], bin_sequence_events,
203     → x_sequence_events, event_seeds,
204     → lineages_sampled_events,prob_empirical_PP,'b-', '#000080',
205     → '#0000FF', 'y+', 1)
206 line_plotter(o_PD, o_events_PD[:,0], bin_sequence_events,
207     → x_sequence_events, event_seeds,
208     → lineages_sampled_events,prob_empirical_PD,'b-', '#000080',
209     → '#0000FF', 'y+', 1)
210 line_plotter(o_DD, o_events_DD[:,0], bin_sequence_events,
211     → x_sequence_events, event_seeds,
212     → lineages_sampled_events,prob_empirical_DD,'b-', '#000080',
213     → '#0000FF', 'y+', 1)
214 line_plotter(o_wan_noshad, o_counts_wan_no_shadow, bin_sequence_wan,
215     → x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
216     → 'b-', '#000080', '#0000FF', 'y+', 2)
217 o_ath5size.bar([0,1,2,3], [(wt_mean_clone_size/wt_mean_clone_size),
218     → (o_wt_mean_clone_size/wt_mean_clone_size),
219     → (ath5_mean_clone_size/wt_mean_clone_size),
220     → (o_ath5_mean_clone_size/wt_mean_clone_size)],color=['#000000','#000080','#000
221     → WT','OSM WT','EO Ath5mo','OSM Ath5mo'])
222 o_ratio.bar([0,1,2,3], [wt_even_odd_ratio, o_wt_even_odd_ratio,
223     → ath5_even_odd_ratio,
224     → o_ath5_even_odd_ratio],color=['#000000','#000080','#000000','#000080'],edgeco
225     → WT,'OFit WT','EO Ath5mo','OFit Ath5mo'])
226 plt.setp(o_ath5size.get_xticklabels(), rotation=25, ha='right')

```

```

204     plt.setp(o_ratio.get_xticklabels(), rotation=25, ha='right')
205
206     s_objects = line_plotter(s_24, s_counts_24, bin_sequence_24_32,
207         ↳ x_sequence_24_32, count_seeds, lineages_sampled_24,
208         ↳ prob_empirical_24, 'm-', '#800080', '#FF00FF', 'k+', 0)
209     line_plotter(s_32, s_counts_32, bin_sequence_24_32, x_sequence_24_32,
210         ↳ count_seeds, lineages_sampled_32, prob_empirical_32, 'm-',
211         ↳ '#800080', '#FF00FF', 'k+', 0)
212     line_plotter(s_48, s_counts_48, bin_sequence_48, x_sequence_48,
213         ↳ count_seeds, lineages_sampled_48, prob_empirical_48, 'm-',
214         ↳ '#800080', '#FF00FF', 'k+', 0)
215     line_plotter(s_PP, s_events_PP[:,0], bin_sequence_events,
216         ↳ x_sequence_events, event_seeds,
217         ↳ lineages_sampled_events,prob_empirical_PP,'m-', '#800080',
218         ↳ '#FF00FF', 'k+', 1)
219     line_plotter(s_PD, s_events_PD[:,0], bin_sequence_events,
220         ↳ x_sequence_events, event_seeds,
221         ↳ lineages_sampled_events,prob_empirical_PD,'m-', '#800080',
222         ↳ '#FF00FF', 'k+', 1)
223     line_plotter(s_DD, s_events_DD[:,0], bin_sequence_events,
224         ↳ x_sequence_events, event_seeds,
225         ↳ lineages_sampled_events,prob_empirical_DD,'m-', '#800080',
226         ↳ '#FF00FF', 'k+', 1)
227     line_plotter(s_wan_noshad, s_counts_wan_no_shadow, bin_sequence_wan,
228         ↳ x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
229         ↳ 'm-', '#800080', '#FF00FF', 'k+', 2)
230     s_ath5size.bar([0,1,2,3], [(wt_mean_clone_size/wt_mean_clone_size),
231         ↳ (s_wt_mean_clone_size/wt_mean_clone_size),(ath5_mean_clone_size/wt_mean_clone_
232         ↳ '#000000', '#800080'],edgecolor='#000000',tick_label=['EO WT','SM
233         ↳ WT','EO Ath5mo','SM Ath5mo'])
234     s_ratio.bar([0,1,2,3], [wt_even_odd_ratio, s_wt_even_odd_ratio,
235         ↳ ath5_even_odd_ratio,
236         ↳ s_ath5_even_odd_ratio],color=['#000000','#000080','#000000','#000080'],edgeco
237         ↳ WT,'SM WT','EO Ath5mo','SM Ath5mo'])
238     plt.setp(s_ath5size.get_xticklabels(), rotation=25, ha='right')
239     plt.setp(s_ratio.get_xticklabels(), rotation=25, ha='right')

240
241     d_objects = line_plotter(d_24, d_counts_24, bin_sequence_24_32,
242         ↳ x_sequence_24_32, count_seeds, lineages_sampled_24,
243         ↳ prob_empirical_24, 'g-', '#008000', '#00FF00', 'k+', 0)

```

```

220     line_plotter(d_32, d_counts_32, bin_sequence_24_32, x_sequence_24_32,
221         ↳ count_seeds, lineages_sampled_32, prob_empirical_32, 'g-',
222         ↳ '#008000', '#00FF00', 'k+', 0)
223     line_plotter(d_48, d_counts_48, bin_sequence_48, x_sequence_48,
224         ↳ count_seeds, lineages_sampled_48, prob_empirical_48, 'g-',
225         ↳ '#008000', '#00FF00', 'k+', 0)
226     line_plotter(d_PP, d_events_PP[:,0], bin_sequence_events,
227         ↳ x_sequence_events, event_seeds,
228         ↳ lineages_sampled_events,prob_empirical_PP,'g-', '#008000',
229         ↳ '#00FF00', 'k+', 1)
230     line_plotter(d_PD, d_events_PD[:,0], bin_sequence_events,
231         ↳ x_sequence_events, event_seeds,
232         ↳ lineages_sampled_events,prob_empirical_PD,'g-', '#008000',
233         ↳ '#00FF00', 'k+', 1)
234     line_plotter(d_DD, d_events_DD[:,0], bin_sequence_events,
235         ↳ x_sequence_events, event_seeds,
236         ↳ lineages_sampled_events,prob_empirical_DD,'g-', '#008000',
237         ↳ '#00FF00', 'k+', 1)
238     line_plotter(d_wan_noshad, d_counts_wan_no_shadow, bin_sequence_wan,
239         ↳ x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
240         ↳ 'g-', '#008000', '#00FF00', 'k+', 2)
241     d_ath5size.bar([0,1,2,3], [(wt_mean_clone_size/wt_mean_clone_size),
242         ↳ (d_wt_mean_clone_size/wt_mean_clone_size),(ath5_mean_clone_size/wt_mean_clone_
243         ↳ WT','DM WT', 'EO Ath5mo', 'DM Ath5mo')])
244     d_ratio.bar([0,1,2,3], [wt_even_odd_ratio, d_wt_even_odd_ratio,
245         ↳ ath5_even_odd_ratio,
246         ↳ d_ath5_even_odd_ratio],color=[ '#000000', '#008000', '#000000', '#008000'],edgeco
247         ↳ WT','DM WT', 'EO Ath5mo', 'DM Ath5mo'])
248     plt.setp(d_ath5size.get_xticklabels(), rotation=25, ha='right')
249     plt.setp(d_ratio.get_xticklabels(), rotation=25, ha='right')
250
251
252     sd_objects_s = line_plotter(sd_24, s_counts_24, bin_sequence_24_32,
253         ↳ x_sequence_24_32, count_seeds, lineages_sampled_24,
254         ↳ prob_empirical_24, 'm-', '#800080', '#FF00FF', 'k+', 0)
255     sd_objects_d = line_plotter(sd_24, d_counts_24, bin_sequence_24_32,
256         ↳ x_sequence_24_32, count_seeds, lineages_sampled_24,
257         ↳ prob_empirical_24, 'g-', '#008000', '#00FF00', 'k+', 0)
258     line_plotter(sd_32, s_counts_32, bin_sequence_24_32,
259         ↳ x_sequence_24_32, count_seeds, lineages_sampled_32,
260         ↳ 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)
238   line_plotter(sd_48, s_counts_48, bin_sequence_48, x_sequence_48,
239     ↳ count_seeds, lineages_sampled_48, prob_empirical_48, 'm-',
240     ↳ '#800080', '#FF00FF', 'k+', 0)
241   line_plotter(sd_48, d_counts_48, bin_sequence_48, x_sequence_48,
242     ↳ count_seeds, lineages_sampled_48, prob_empirical_48, 'g-',
243     ↳ '#008000', '#00FF00', 'k+', 0)
244   line_plotter(sd_PP, s_events_PP[:,0], bin_sequence_events,
245     ↳ x_sequence_events, event_seeds,
246     ↳ lineages_sampled_events,prob_empirical_PP,'m-','#800080',
247     ↳ '#FF00FF', 'k+', 1)
248   line_plotter(sd_PP, d_events_PP[:,0], bin_sequence_events,
249     ↳ x_sequence_events, event_seeds,
250     ↳ lineages_sampled_events,prob_empirical_PP,'g-','#008000',
251     ↳ '#00FF00', 'k+', 1)
252   line_plotter(sd_PD, s_events_PD[:,0], bin_sequence_events,
253     ↳ x_sequence_events, event_seeds,
254     ↳ lineages_sampled_events,prob_empirical_PD,'m-','#800080',
255     ↳ '#FF00FF', 'k+', 1)
256   line_plotter(sd_PD, d_events_PD[:,0], bin_sequence_events,
257     ↳ x_sequence_events, event_seeds,
258     ↳ lineages_sampled_events,prob_empirical_PD,'g-','#008000',
259     ↳ '#00FF00', 'k+', 1)
260   line_plotter(sd_DD, s_events_DD[:,0], bin_sequence_events,
261     ↳ x_sequence_events, event_seeds,
262     ↳ lineages_sampled_events,prob_empirical_DD,'m-','#800080',
263     ↳ '#FF00FF', 'k+', 1)
264   line_plotter(sd_DD, d_events_DD[:,0], bin_sequence_events,
265     ↳ x_sequence_events, event_seeds,
266     ↳ lineages_sampled_events,prob_empirical_DD,'g-','#008000',
267     ↳ '#00FF00', 'k+', 1)
268   line_plotter(sd_wan_noshad, s_counts_wan_no_shadow, bin_sequence_wan,
269     ↳ x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
270     ↳ 'm-','#800080', '#FF00FF', 'k+', 2)
271   line_plotter(sd_wan_noshad, d_counts_wan_no_shadow, bin_sequence_wan,
272     ↳ x_sequence_wan, count_seeds, lineages_sampled_wan, prob_wan,
273     ↳ 'g-','#008000', '#00FF00', 'k+', 2)
274   sd_ath5size.bar([0,1,2,3,4,5], [(wt_mean_clone_size),
275     ↳ (s_wt_mean_clone_size),(d_wt_mean_clone_size),(ath5_mean_clone_size),(s_ath5_
276     ↳ WT','SM WT','DM WT', 'EO Ath5mo', 'SM Ath5Mo', 'DM Ath5mo')])

```

```

247     sd_ratio.bar([0,1,2,3,4,5], [wt_even_odd_ratio, s_wt_even_odd_ratio,
248     ↵ d_wt_even_odd_ratio, ath5_even_odd_ratio, s_ath5_even_odd_ratio,
249     ↵ d_ath5_even_odd_ratio],color=['#000000','#800080','#008000','#000000','#80008
250     ↵ WT','SM WT','DM WT', 'EO Ath5mo', 'SM Ath5Mo', 'DM Ath5mo'])
251
252 plt.setp(sd_ath5size.get_xticklabels(), rotation=25, ha='right')
253 plt.setp(sd_ratio.get_xticklabels(), rotation=25, ha='right')
254
255 #calculate AICs, write table to file
256 o_fit_AIC = calculate_fit_AIC(o_fit_output_list, empirical_fit_list,
257     ↵ he_model_params)
258 o_test_AIC = calculate_test_AIC(o_test_output_list,
259     ↵ empirical_fit_list, he_model_params)
260
261 s_fit_AIC = calculate_fit_AIC(s_fit_output_list, empirical_fit_list,
262     ↵ he_model_params)
263 s_test_AIC = calculate_test_AIC(s_test_output_list,
264     ↵ empirical_test_list, he_model_params)
265
266 d_fit_AIC = calculate_fit_AIC(d_fit_output_list, empirical_fit_list,
267     ↵ det_model_params)
268 d_test_AIC = calculate_test_AIC(d_test_output_list,
269     ↵ empirical_test_list, det_model_params)
270
271 AIC_table('AIC_table.tex', o_fit_AIC, o_test_AIC, s_fit_AIC,
272     ↵ s_test_AIC, d_fit_AIC, d_test_AIC)
273
274 #plot annotations
275 plot_annotater(s_fig,s_objects,'Stochastic Model')
276 plot_annotater(o_fig,o_objects,'Stochastic Model (original fit)')
277 plot_annotater(d_fig,d_objects, 'Deterministic Model')
278 plot_annotater(sd_fig,sd_objects_s, 'Stochastic Model', sd_objects_d,
279     ↵ 'Deterministic Model')
280
281 #export as .tiff, .png
282 o_fig.savefig('o_fig.png',dpi=600)
283 png_memory = BytesIO()
284 o_fig.savefig(png_memory, format='png', dpi=600)
285 PILpng = Image.open(png_memory)
286 PILpng.save('o_fig.tiff')
287 png_memory.close()
288
289 s_fig.savefig('s_fig.png',dpi=600)
290 png_memory = BytesIO()

```

```
279     s_fig.savefig(png_memory, format='png', dpi=600)
280     PILpng = Image.open(png_memory)
281     PILpng.save('s_fig.tiff')
282     png_memory.close()
283
284     d_fig.savefig('d_fig.png',dpi=600)
285     png_memory = BytesIO()
286     d_fig.savefig(png_memory, format='png', dpi=600)
287     PILpng = Image.open(png_memory)
288     PILpng.save('d_fig.tiff')
289     png_memory.close()
290
291     sd_fig.savefig('sd_fig.png',dpi=600)
292     png_memory = BytesIO()
293     sd_fig.savefig(png_memory, format='png', dpi=600)
294     PILpng = Image.open(png_memory)
295     PILpng.save('sd_fig.tiff')
296     png_memory.close()
297
298     plt.show()
299
300
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     for i in range(0,3):
310         output_counts = output_list[i]
311         empirical_prob = empirical_list[i]
312
313         output_histo,bin_edges = np.histogram(output_counts,
314             np.arange(1,len(empirical_prob)+2), density = False)
315         output_prob = np.array(output_histo/count_seeds)
316
317         residual = np.array(output_prob- empirical_prob)
318         number_comparisons += len(residual)
319         rss[i] = np.sum(np.square(residual))
320
321     for i in range (0,3):
```

```
319     output_events = output_list[i+3]
320     empirical_prob = empirical_list[i+3]
321
322     histo_events,bin_edges = np.histogram(output_events,
323         ↳ bin_sequence_events, density=True)
324     output_prob = np.array(histo_events/event_seeds)
325
326     residual = np.array(output_prob - empirical_prob)
327     number_comparisons += len(residual)
328     rss[i+3] = np.sum(np.square(residual))
329
330     AIC = 2 * number_params + number_comparisons * np.log(np.sum(rss))
331
332     return AIC
333
334 def calculate_test_AIC(output_list, empirical_list, number_params):
335     #function calculates AIC for He mutant data & Wan counts
336     rss=np.zeros(3)
337     number_comparisons = 0
338
339     #Wan data
340     for i in range (0,1):
341         output_counts = output_list[i]
342         empirical_prob = empirical_list[i]
343
344         output_histo,bin_edges = np.histogram(output_counts,
345             ↳ np.arange(2,len(empirical_prob)+3), density = False)
346         output_prob = np.array(output_histo/count_seeds)
347
348         residual = np.array(output_prob- empirical_prob)
349         number_comparisons += len(residual)
350         rss[i] = np.sum(np.square(residual))
351
352     #He data
353     for i in range(1,3):
354         output_measure = output_list[i]
355         empirical_measure = empirical_list[i]
356
357         residual = np.array(output_measure - empirical_measure)
358         number_comparisons += 1
359         rss[i] = np.sum(np.square(residual))
360
361     AIC = 2 * number_params + number_comparisons * np.log(np.sum(rss))
```

```
360
361     return AIC
362
363 #line plotter plots average counts or event event +- 2 standard
364 #    deviations (He et al's "95% probability interval")
365 # "mode" 1 gives correct hourly output for 5-hour event bins. mode 2
366 #    begins Wan plots at clone size 2, reflecting unavailable data
367 def line_plotter(subplot, data, bin_sequence, x_sequence, seeds, samples,
368 #    empirical_prob, line_colour_string, fill_edge_colour_string,
369 #    fill_face_colour_string, empirical_colour_string, mode):
370
371     legend_objects = []
372
373     prob_histo, bin_edges = np.histogram(data, bin_sequence,
374 #        density=True)
375
376     #estimate of 95% CI for empirical observations of model output from
377 #    repeated sampling of the data w/ the appropriate number of
378 #    empirically observed lineages
379     interval = sampler(data,samples,bin_sequence)
380     legend_objects.append(subplot.plot(x_sequence,prob_histo,
381 #        line_colour_string))
382     subplot.fill_between(x_sequence, (prob_histo - interval), (prob_histo
383 #        + interval), alpha=0.1, edgecolor=fill_edge_colour_string,
384 #        facecolor=fill_face_colour_string)
385
386     if mode == 0:
387
388         legend_objects.append(subplot.plot(np.arange(1,len(empirical_prob)+1),empirical_prob,
389 #            empirical_colour_string))
390
391     if mode == 1:
392
393         legend_objects.append(subplot.plot(np.arange(30,80,5),empirical_prob,
394 #            empirical_colour_string))
395
396     if mode == 2:
397
398         legend_objects.append(subplot.plot(np.arange(2,len(empirical_prob)+2,1),
399 #            empirical_prob, empirical_colour_string))
400
401     return legend_objects
402
```

```

387 def sampler(data,samples,bin_sequence):
388
389     if len(data) == 0: #catches edge case w/ no entries for a mitotic
        ↵ mode
390         data=[0]
391
392     base_sample=np.zeros((error_samples,len(bin_sequence)-1))
393
394     for i in range(0,error_samples):
395         new_data_sample = np.random.choice(data,samples)
396         new_histo_prob, bin_edges = np.histogram(new_data_sample,
            ↵ bin_sequence, density=True)
397         base_sample[i,:] = new_histo_prob
398
399     sample_95CI = np.array(2 * (np.std(base_sample,0)))
400
401     return sample_95CI
402
403 def plot_annotater(fig_to_ann, legend_objects_1, legend_objects_1_name,
    ↵ legend_objects_2=None, legend_objects_2_name=None):
404     #subplot labels
405     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,
            ↵ fontweight='bold', va='top')
409
410         time_rectangle_x = .70
411         time_rectangle_y = .65
412         time_rectangle_width = .2
413         time_rectangle_height = .03
414         time_caret_half_width = .02
415         time_caret_height = np.sqrt(3)*(time_caret_half_width*2)
416
417         if i < 3:
418             #add the time bar and label 72 hr caret on appropriate plots
419
            ↵ ax.add_patch(plt.Rectangle((time_rectangle_x,time_rectangle_y),time_r
            ↵ color='k', transform=ax.transAxes))
420
            ↵ ax.add_patch(plt.Polygon(((time_rectangle_x+time_rectangle_width, time_
            ↵ fill=False, transform=ax.transAxes))

```

```
421     ↵ ax.text(time_rectangle_x+time_rectangle_width,time_rectangle_y+time_rect
        ↵ transform=ax.transAxes,
        ↵ fontsize='8',va='bottom',ha='center')
422
423 if i == 0:
424     #add 24hr caret
425
426     ↵ ax.add_patch(plt.Polygon(((time_rectangle_x,time_rectangle_y+time_rect
        ↵ fill=False, transform=ax.transAxes))
427
428 if i == 1:
429     #add 32hr caret
430
431     ↵ ax.add_patch(plt.Polygon(((time_rectangle_x+(8/48)*time_rectangle_wid
        ↵ fill=False, transform=ax.transAxes))
432
433 if i == 2:
434
435     ↵ ax.add_patch(plt.Polygon(((time_rectangle_x+(24/48)*time_rectangle_wi
        ↵ 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     ↵ ax.add_patch(plt.Polygon((parent_circle_coord),child_1_coord),color=
```

```
445      ↪ ax.add_patch(plt.Polygon((parent_circle_coord),child_2_coord),color=
446      ↪ ax.add_patch(patches.Ellipse(parent_circle_coord,ellipse_width,ellipse_
      ↪ color='k', transform=ax.transAxes))
447      ↪ ax.add_patch(patches.Ellipse(child_1_coord,ellipse_width,ellipse_height,
      ↪ color='k', transform=ax.transAxes))
448      ↪ ax.add_patch(patches.Ellipse(child_2_coord,ellipse_width,ellipse_height,
      ↪ color='k', transform=ax.transAxes))
449      ax.text(.91, .7, 'PP', transform=ax.transAxes, va='center',
      ↪ ha='left')
450
451 if i == 4:
452
453      ↪ ax.add_patch(plt.Polygon((parent_circle_coord),child_1_coord),color=
454      ↪ ax.add_patch(plt.Polygon((parent_circle_coord),child_2_coord),color=
455      ↪ ax.add_patch(patches.Ellipse(parent_circle_coord,ellipse_width,ellipse_
      ↪ color='k', transform=ax.transAxes))
456      ↪ ax.add_patch(patches.Ellipse(child_1_coord,ellipse_width,ellipse_height,
      ↪ color='k', transform=ax.transAxes))
457      ↪ ax.add_patch(patches.Ellipse(child_2_coord,ellipse_width,ellipse_height,
      ↪ edgecolor='k', facecolor='w', transform=ax.transAxes))
458      ax.text(.91, .7, 'PD', transform=ax.transAxes, va='center',
      ↪ ha='left')
459
460 if i == 5:
461
462      ↪ ax.add_patch(plt.Polygon((parent_circle_coord),child_1_coord),color=
463      ↪ ax.add_patch(plt.Polygon((parent_circle_coord),child_2_coord),color=
464      ↪ ax.add_patch(patches.Ellipse(parent_circle_coord,ellipse_width,ellipse_
      ↪ color='k', transform=ax.transAxes))
465      ↪ ax.add_patch(patches.Ellipse(child_1_coord,ellipse_width,ellipse_height,
      ↪ edgecolor='k', facecolor='w', transform=ax.transAxes))
```

```
465     ↵ ax.add_patch(patches.Ellipse(child_2_coord, ellipse_width, ellipse_height,
466     ↵ edgecolor='k', facecolor='w', transform=ax.transAxes))
467
468
469     if i == 6:
470         ax.text(.80,.85,'CMZ', transform=ax.transAxes,
471             ↵ fontweight='bold', va='top')
472
473     if legend_objects_2 == None:
474
475         ↵ fig_to_ann.legend((legend_objects_1[1][0],legend_objects_1[0][0]),('Observer',
476         ↵ bbox_to_anchor=(.59,.10,.1,.1), loc='upper left')
477     else:
478
479         ↵ fig_to_ann.legend((legend_objects_1[1][0],legend_objects_1[0][0],legend_objects_2_name),
480         ↵ legend_objects_2_name, bbox_to_anchor=(.59,.1,.1,.1),
481         ↵ loc='upper left')
482
483
484     #data group box annotations
485     fig_to_ann.patches.extend([plt.Rectangle((0.005,0.41),0.99,0.58,
486         ↵ fill=False, color='k', linestyle='-', transform=fig_to_ann.transFigure,
487         ↵ figure=fig_to_ann)])
488
489     fig_to_ann.patches.extend([plt.Polygon(((0.005, .4),(.995,.4),
490         ↵ (.995,.21), (.5,.21), (.5, .01), (.005, .01)),
491         ↵ fill=False, color='k', linestyle=':', transform=fig_to_ann.transFigure,
492         ↵ figure=fig_to_ann)])
493
494
495     fig_to_ann.text(.605, .075, 'Training Data', fontsize=12, transform =
496         ↵ fig_to_ann.transFigure, figure=fig_to_ann)
497     fig_to_ann.patches.extend([plt.Rectangle((0.6,0.07),0.143,0.029,
498         ↵ fill=False, color='k', linestyle='-', transform=fig_to_ann.transFigure,
499         ↵ figure=fig_to_ann)])
500
501
502     fig_to_ann.text(.605, .025, 'Test Data', fontsize=12, transform =
503         ↵ fig_to_ann.transFigure, figure=fig_to_ann)
504     fig_to_ann.patches.extend([plt.Rectangle((0.6,0.02),0.111,0.029,
```

```

493         fill=False, color='k', linestyle=':',  

494         transform=fig_to_ann.transFigure,  

495         → figure=fig_to_ann))  

496  

497     fig_to_ann.subplots_adjust(wspace=0.4, hspace=0.3)  

498     fig_to_ann.tight_layout()  

499  

500 def AIC_table (filename, o_training_AIC, o_test_AIC, s_training_AIC,  

501   → s_test_AIC, d_training_AIC, d_test_AIC):  

502     #writes LaTeX tabular snippet with AIC values  

503     table_file = open(filename, 'w')  

504     table_file.write(r'\begin{tabular}{l|l|l}'+ '\n')  

505     table_file.write(r'\hline'+ '\n')  

506     table_file.write(r'{\bf Model} & {\bf Training AIC} & {\bf Test AIC}  

507     → \\ \thickhline'+ '\n')  

508     table_file.write(r'Stochastic (He fit) &  

509     → $'+f'{o_training_AIC:.2f}'+r'$ & $' + f'{o_test_AIC:.2f}' + r'$\\  

510     → \hline'+ '\n')  

511     table_file.write(r'Stochastic (SPSA fit) &  

512     → $'+f'{s_training_AIC:.2f}'+r'$ & $' + f'{s_test_AIC:.2f}' + r'$\\  

513     → \hline'+ '\n')  

514     table_file.write(r'Deterministic (SPSA fit) &  

515     → $'+f'{d_training_AIC:.2f}'+r'$ & $' + f'{d_test_AIC:.2f}' + r'$\\  

516     → \hline'+ '\n')  

517     table_file.write(r'\end{tabular}'+ '\n')  

518  

519 if __name__ == "__main__":  

520     main()

```

---

### 16.2.15 /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/KolmogorovComplexity/gomes_kol.txt')
13         skiprows=1, usecols=1)
14     he_kol =
15         np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/KolmogorovComplexity/he_kol.txt')
16         skiprows=1, usecols=1)
17     he_refit_kol =
18         np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/KolmogorovComplexity/he_refit_kol.txt')
19         skiprows=1, usecols=1)
20     deterministic_kol =
21         np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/KolmogorovComplexity/deterministic_kol.txt')
22         skiprows=1, usecols=1)
23     boije_kol =
24         np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/KolmogorovComplexity/boije_kol.txt')
25         skiprows=1, usecols=1)
26     eo_kol =
27         np.loadtxt('/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/KolmogorovComplexity/eo_kol.txt')
28         skiprows=1, usecols=1)

29     data_sequence =
30         [gomes_kol,he_kol,he_refit_kol,deterministic_kol,boije_kol,eo_kol]
31
32     plt.violinplot(data_sequence, showextrema=True)
33
34     plt.ylabel ('Estimated Kolmogorov Complexity')
35
36     plt.show()

37
38 if __name__ == "__main__":
39     main()

```

---

### 16.2.16 /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
    ↳ plausibility interval for simulations

16
17 bin_sequence_events = np.arange(30,85,5)
18 x_sequence_events = np.arange(30,80,5)

19
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 E0 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.')
    ↳ skiprows=1, usecols=(3,5,8))
33 E0_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(E0_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():
    #LOAD & PARSE HE MODEL OUTPUT FILES
    #original fit stochastic mitotic mode files
40     o_events =
        ↳ np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM")
        ↳ skiprows=1, usecols=(0))

41
42     #refit stochastic mitotic mode files

```

```

45     s_events =
46         np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM
47         skiprows=1, usecols=(0))
48
49     #deterministic mitotic mode files
50     d_events =
51         np.loadtxt("/home/main/git/chaste/projects/ISP/python_fixtures/testoutput/HeM
52         skiprows=1, usecols=(0))
53
54     original = line_plotter(o_events,bin_sequence_events,
55         x_sequence_events, event_seeds, lineages_sampled_events, 'b--',
56         '#000080', '#0000FF')
57     refit = line_plotter(s_events,bin_sequence_events, x_sequence_events,
58         event_seeds, lineages_sampled_events, 'm-', '#800080', '#FF00FF')
59     deterministic = line_plotter(d_events,bin_sequence_events,
60         x_sequence_events, event_seeds, lineages_sampled_events, 'g-',
61         '#008000', '#00FF00')
62
63     empirical = plt.plot(np.arange(30,80,5),empirical_events_per_lineage,
64         'k+')
65
66     plt.legend((original[0], refit[0], deterministic[0], empirical[0]),
67         ('SM (He fit)', 'SM (SPSA fit)', 'DM', 'Observed'))
68
69     plt.xlabel ('Age (hpf)')
70     plt.ylabel ('Probability of mitotic event per lineage per hour')
71
72     plt.savefig("per_lineage_mitotic_rate.png")
73     png_memory = BytesIO()
74     plt.savefig(png_memory, format='png', dpi=600)
75     PILpng = Image.open(png_memory)
76     PILpng.save('per_lineage_mitotic_rate.tiff')
77     png_memory.close()
78
79     plt.show()
80
81 #line plotter plots average counts or event event +- 2 standard
82     deviations (He et al's "95% probability interval")
83 # "mode" 1 gives correct hourly output for 5-hour event bins. mode 2
84     begins Wan plots at clone size 2, reflecting unavailable data

```

```
73 def line_plotter(data, bin_sequence, x_sequence, seeds, samples,
74     ↵ line_colour_string, fill_edge_colour_string,
75     ↵ fill_face_colour_string):
76
77
78     histo, bin_edges = np.histogram(data, bin_sequence, density=False)
79     prob_histo = np.array(histo / (seeds * 5))
80
81
82     #estimate of 95% CI for empirical observations of model output from
83     ↵ repeated sampling of the data w/ the appropriate number of
84     ↵ empirically observed lineages
85     interval = sampler(data,samples,bin_sequence)
86     line = plt.plot(x_sequence,prob_histo, line_colour_string)
87     plt.fill_between(x_sequence, (prob_histo - interval), (prob_histo +
88         ↵ interval), alpha=0.1, edgecolor=fill_edge_colour_string,
89         ↵ facecolor=fill_face_colour_string)
90
91
92     return line
93
94
95 def sampler(data,samples,bin_sequence):
96
97     if len(data) == 0: #catches edge case w/ no entries for a mitotic
98         ↵ mode
99         data=[0]
100
101
102     base_sample=np.zeros((error_samples,len(bin_sequence)-1))
103
104
105     for i in range(0,error_samples):
106         new_data_sample = np.random.choice(data,samples)
107         new_histo, bin_edges = np.histogram(new_data_sample,
108             ↵ bin_sequence, density=False)
109         new_histo_prob = np.array(new_histo/samples)
110         base_sample[i,:] = new_histo_prob
111
112
113     sample_95CI = np.array(2 * (np.std(base_sample,0)))
114
115
116     return sample_95CI
117
118
119 if __name__ == "__main__":
120     main()
```

### 16.2.17 /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-')

```

```

37     plt.fill_between(empirical_x_seq, (empirical_pop_mean -
38         ↵ empirical_pop_95CI), (empirical_pop_mean + empirical_pop_95CI),
39         ↵ alpha=0.1, edgecolor='#008000', facecolor='#00FF00')
40
41     wan_simulator = plt.plot(wan_sim_x_seq, wan_sim_mean, 'm-')
42     plt.fill_between(wan_sim_x_seq, (wan_sim_mean - wan_sim_95CI),
43         ↵ (wan_sim_mean + wan_sim_95CI), alpha=0.1, edgecolor='#800080',
44         ↵ facecolor='#FF00FF')
45
46     plt.ylim(bottom=0)
47     plt.xlim(left=0)
48
49     plt.ylabel("CMZ population size")
50     plt.xlabel("Retina age (dpf)")
51     ax.xaxis.set_major_locator(ticker.MultipleLocator(40))
52     ax.legend((empirical[0], wan_simulator[0]), ('Observed CMZ
53         ↵ population', '"Wan-type" simulated CMZ population'))
54
55     png_memory = BytesIO()
56     fig.savefig(png_memory, format='png', dpi=600)
57     PILpng = Image.open(png_memory)
58     PILpng.save('wan_fig.tiff')
59     png_memory.close()
60
61     plt.show()
62
63 if __name__ == "__main__":
64     main()

```

---

### 16.2.18 /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(

```

```

7             false), mMitoticMode(0), mSeed(0), mp_PostMitoticType(),
8     ↳ mp_RGC_Type(), mp_AC_HC_Type(), mp_PR_BC_Type(),
9     ↳ mp_label_Type()
10    }
11
12 BoijeCellCycleModel::BoijeCellCycleModel(const BoijeCellCycleModel&
13   ↳ rModel) :
14     AbstractSimpleCellCycleModel(rModel), mOutput(rModel.mOutput),
15     ↳ mEventStartTime(rModel.mEventStartTime), mSequenceSampler(
16       rModel.mSequenceSampler),
17       ↳ mSeqSamplerLabelSister(rModel.mSeqSamplerLabelSister),
18       ↳ mDebug(rModel.mDebug), mTimeID(
19         rModel.mTimeID), mVarIDs(rModel.mVarIDs),
20         ↳ mDebugWriter(rModel.mDebugWriter), mGeneration(
21           rModel.mGeneration), mPhase2gen(rModel.mPhase2gen),
22           ↳ mPhase3gen(rModel.mPhase3gen), mprobAtoh7(
23             rModel.mprobAtoh7), mprobPtf1a(rModel.mprobPtf1a),
24             ↳ mprobng(rModel.mprobng), mAtoh7Signal(
25               rModel.mAtoh7Signal), mPtf1aSignal(rModel.mPtf1aSignal),
26               ↳ mNgSignal(rModel.mNgSignal), mMitoticMode(
27                 rModel.mMitoticMode), mSeed(rModel.mSeed),
28                 ↳ mp_PostMitoticType(rModel.mp_PostMitoticType),
29                 ↳ mp_RGC_Type(
30                   rModel.mp_RGC_Type), mp_AC_HC_Type(rModel.mp_AC_HC_Type),
31                   ↳ mp_PR_BC_Type(rModel.mp_PR_BC_Type), mp_label_Type(
32                     rModel.mp_label_Type)
33
34 {
35 }
```

23

```

24 AbstractCellCycleModel* BoijeCellCycleModel::CreateCellCycleModel()
25 {
26     return new BoijeCellCycleModel(*this);
27 }
```

28

```

29 void BoijeCellCycleModel::SetCellCycleDuration()
30 {
31
32     if
33     ↳ (mpCell->GetCellProliferativeType()->IsType<DifferentiatedCellProliferativeTy
34     {
35         mCellCycleDuration = DBL_MAX;
36     }
```

```

36     else
37     {
38         mCellCycleDuration = 1.0;
39     }
40 }
41 }
42
43 void BoijeCellCycleModel::ResetForDivision()
44 {
45     mGeneration++; //increment generation counter
46     //the first division is ascribed to generation "1"
47
48     RandomNumberGenerator* p_random_number_generator =
49     → RandomNumberGenerator::Instance();
50
51     mMitoticMode = 0; //0=PP;1=PD;2=DD
52
53     /*****
54      * TRANSCRIPTION FACTOR RANDOM VARIABLES & RULES
55      * 1st phase: only PP divisions (symmetric proliferative) permitted
56      * 2nd phase: all division modes permitted, all TF signals avail
57      * 3rd phase: only PP/DD modes permitted, only ng has nonzero signal
58      *****/
59
60     //reset RVs and signals
61     double atoh7RV = 1, ptf1aRV = 1, ngRV = 1;
62     mAtoh7Signal = mPtf1aSignal = mNgSignal = false;
63
64     //PHASE & TF SIGNAL RULES
65     if (mGeneration > mPhase2gen && mGeneration ≤ mPhase3gen) //if the
66     → cell is in the 2nd model phase, all signals have nonzero
67     → probabilities at each division
68     {
69         //RVs take values evenly distributed across 0-1
70         atoh7RV = p_random_number_generator→ranf();
71         ptf1aRV = p_random_number_generator→ranf();
72         ngRV = p_random_number_generator→ranf();
73
74         if (atoh7RV < mprobAtoh7)
75         {
76             mAtoh7Signal = true;
77         }

```

```

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
87   ↪ phase, only ng signal has a nonzero probability
87 {
88     ngRV = p_random_number_generator→ranf();
89     //roll a probability die for the ng signal
90     if (ngRV < mprobng)
91     {
92         mNgSignal = true;
93     }
94 }
95
96 /*****
97 * MITOTIC MODE & SPECIFICATION RULES
98 * -(additional asymmetric postmitotic rules specified in
99 ↪ InitialiseDaughterCell());
100 * *****/
100
101 if(mAtoh7Signal == true)
102 {
103     mMitoticMode = 1;
104 }
105
106 if (mPtf1aSignal == true && mAtoh7Signal == false) //Ptf1A alone
107   ↪ gives a symmetrical postmitotic AC/HC division
107 {
108     mMitoticMode = 2;
109     mpCell→SetCellProliferativeType(mp_PostMitoticType);
110     mpCell→AddCellProperty(mp_AC_HC_Type);
111 }
112
113 if (mPtf1aSignal == false && mAtoh7Signal == false && mNgSignal ==
114   ↪ true) //ng alone gives a symmetrical postmitotic PR/BC division
114 {

```

```
115     mMitoticMode = 2;
116     mpCell->SetCellProliferativeType(mp_PostMitoticType);
117     mpCell->AddCellProperty(mp_PR_BC_Type);
118 }
119
120 /*****
121 * Write mitotic event to file if appropriate
122 * mDebug: many files: detailed per-lineage info switch; intended for
123 * TestHeInductionCountFixture
124 * mOutput: 1 file: time, seed, cellID, mitotic mode, intended for
125 * TestHeMitoticModeRateFixture
126 * *****/
127
128 if (mDebug)
129 {
130     WriteDebugData(atoh7RV, ptf1aRV, ngRV);
131 }
132
133 if (mOutput)
134 {
135     WriteModeEventOutput();
136 }
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     }
157 }
```

```
154         else
155     {
156         mSeqSamplerLabelSister = false;
157     }
158 }
159 else
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
225     mp_RGC_Type = p_RGC_Type;
226     mp_AC_HC_Type = p_AC_HC_Type;
227     mp_PR_BC_Type = p_PR_BC_Type;
228 }
229
230
231 void BoijeCellCycleModel::SetModelParameters(unsigned phase2gen, unsigned
232     ← phase3gen, double probAtoh7, double probPtf1a,
233                                     double probng)
234 {
235
236     mPhase2gen = phase2gen;
237     mPhase3gen = phase3gen;
238     mprobAtoh7 = probAtoh7;
```

```
231     mprobPtf1a = probPtf1a;
232     mprobng = probng;
233 }
234
235 void BoijeCellCycleModel::EnableModeEventOutput(double eventStart,
236     ↪ unsigned seed)
237 {
238     mOutput = true;
239     mEventStartTime = eventStart;
240     mSeed = seed;
241 }
242
243 void BoijeCellCycleModel::WriteModeEventOutput()
244 {
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
262 void
263     ↪ BoijeCellCycleModel::EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
264         ↪ debugWriter)
265 {
266     mDebug = true;
267     mDebugWriter = debugWriter;
268
269     mTimeID = mDebugWriter→DefineUnlimitedDimension("Time", "h");
270     mVarIDs.push_back(mDebugWriter→DefineVariable("CellID", "No"));
271     mVarIDs.push_back(mDebugWriter→DefineVariable("Generation", "No"));
272     mVarIDs.push_back(mDebugWriter→DefineVariable("MitoticMode",
273         ↪ "Mode"));
```

```

266     mVarIDs.push_back(mDebugWriter->DefineVariable("atoh7Set",
267         "Percentile"));
268     mVarIDs.push_back(mDebugWriter->DefineVariable("atoh7RV",
269         "Percentile"));
270     mVarIDs.push_back(mDebugWriter->DefineVariable("ptf1aSet",
271         "Percentile"));
272     mVarIDs.push_back(mDebugWriter->DefineVariable("ptf1aRV",
273         "Percentile"));
274     mVarIDs.push_back(mDebugWriter->DefineVariable("ngSet",
275         "Percentile"));
276     mVarIDs.push_back(mDebugWriter->DefineVariable("ngRV",
277         "Percentile"));

278     mDebugWriter->EndDefineMode();
279 }

280 void BoijeCellCycleModel::WriteDebugData(double atoh7RV, double ptf1aRV,
281     double ngRV)
282 {
283     double currentTime = SimulationTime::Instance()->GetTime();
284     CellPtr currentCell = GetCell();
285     double currentCellID = (double) currentCell->GetCellId();

286     mDebugWriter->PutVariable(mTimeID, currentTime);
287     mDebugWriter->PutVariable(mVarIDs[0], currentCellID);
288     mDebugWriter->PutVariable(mVarIDs[1], mGeneration);
289     mDebugWriter->PutVariable(mVarIDs[2], mMitoticMode);
290     mDebugWriter->PutVariable(mVarIDs[3], mprobAtoh7);
291     mDebugWriter->PutVariable(mVarIDs[4], atoh7RV);
292     mDebugWriter->PutVariable(mVarIDs[5], mprobPtf1a);
293     mDebugWriter->PutVariable(mVarIDs[6], ptf1aRV);
294     mDebugWriter->PutVariable(mVarIDs[7], mprobng);
295     mDebugWriter->PutVariable(mVarIDs[8], ngRV);
296     mDebugWriter->AdvanceAlongUnlimitedDimension();
297 }

298 ****
299 * UNUSED FUNCTIONS (required for class nonvirtuality, do not remove)
300 ****/
301
302 double BoijeCellCycleModel::GetAverageTransitCellCycleTime()
303 {
304     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
316
317     ↵ AbstractSimpleCellCycleModel::OutputCellCycleModelParameters(rParamsFile);
318
319 // Serialization for Boost ≥ 1.36
320 #include "SerializationExportWrapperForCpp.hpp"
321 CHASTE_CLASS_EXPORT(BoijeCellCycleModel)

```

---

### 16.2.19 /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:
→ 10.1016/j.devcel.2015.08.011
19 *
20 * USE: By default, BoijeCellCycleModels are constructed with the
→ parameters reported in [Boije2015].
21 * In normal use, the model steps through three phases of probabilistic
→ transcription factor signalling.
22 * These signals determine the mitotic mode and the fate of offspring.
23 *
24 * NB: the Boije model is purely generational and assumes a generation
→ time of 1; time in Boije simulations
25 * thus represents generation number rather than proper time. Refactor for
→ use with simulations that refer
26 * to clocktime!!
27 *
28 * 2 per-model-event output modes:
29 * EnableModeEventOutput() enables mitotic mode event logging-all cells
→ will write to the singleton log file
30 * EnableModelDebugOutput() enables more detailed debug output, each seed
→ will have its own file written to
31 * by a ColumnDataWriter passed to it from the test
32 * (eg. by the SetupDebugOutput helper function in the project simulator)
33 *
34 * 1 mitotic-event-sequence sampler (only samples one "path" through the
→ lineage):
35 * EnableSequenceSampler() - one "sequence" of progenitors writes mitotic
→ event type to a string in the singleton log file
36 *
37 *****/
38
39 class BoijeCellCycleModel : public AbstractSimpleCellCycleModel
40 {
41     friend class TestSimpleCellCycleModels;
42
43 private:
44
45     /** Needed for serialization.*/
46     friend class boost::serialization::access;
47     /**
48      * Archive the cell-cycle model and random number generator, never
→ used directly - boost uses this.
49      *
50      * @param archive the 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             → RandomNumberGenerator::Instance()→GetSerializationWrapper();
61         archive & p_wrapper;
62         archive & mCellCycleDuration;
63         archive & mGeneration;
64     }
65
66     //Private write functions for models
67     void WriteModeEventOutput();
68     void WriteDebugData(double atoh7RV, double ptf1aRV, double ngRV);
69
70 protected:
71     //mode/output variables
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     unsigned mGeneration;
83     unsigned mPhase2gen;
84     unsigned mPhase3gen;
85     double mprobAtoh7;
86         double mprobPtf1a;
87         double mprobng;
88         bool mAtoh7Signal;
89         bool mPtf1aSignal;
90         bool mNgSignal;
91     unsigned mMitoticMode;
92     unsigned mSeed;
93     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    ← *
107    ← * This method is called by child classes to set member variables for
108    ← a daughter cell upon cell division.
109    ← * Note that the parent cell cycle model will have had
110    ← ResetForDivision() called just before CreateCellCycleModel() is
111    ← called,
112    ← * so performing an exact copy of the parent is suitable behaviour.
113    ← Any daughter-cell-specific initialisation
114    ← * can be done in InitialiseDaughterCell().
115    ← *
116    ← * @param rModel the cell cycle model to copy.
117    ← */
118 BoijeCellCycleModel(const BoijeCellCycleModel& rModel);
119
120 public:
121
122 /**
123  * Constructor - just a default, mBirthTime is set in the
124  ← AbstractCellCycleModel class.
125  */
126 BoijeCellCycleModel();
127
128 /**
129  * SetCellCycleDuration() method to set length of cell cycle (default
130  ← 1.0 as Boije's model is purely generational)
131  */
132 void SetCellCycleDuration();
133
134 /**
135  *
```

```
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 /**
134  * Overridden ResetForDivision() method.
135  * Contains general mitotic mode logic
136  */
137 void ResetForDivision();
138
139 /**
140  * 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 ****/
```

```

166     void SetModelParameters(unsigned phase2gen = 3, unsigned phase3gen =
167     ↵ 5, double probAtoh7 = 0.32, double probPtf1a = 0.3, double probng
168     ↵ = 0.8);
169
170     void SetPostMitoticType(boost::shared_ptr<AbstractCellProperty>
171     ↵ p_PostMitoticType);
172
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
178 //Functions to enable per-cell mitotic mode logging for mode rate &
179 //sequence sampling fixtures
180 //Uses singleton logfile
181
182     void EnableModeEventOutput(double eventStart, unsigned seed);
183
184     void EnableSequenceSampler(boost::shared_ptr<AbstractCellProperty>
185     ↵ label);
186
187
188 //More detailed debug output. Needs a ColumnDataWriter passed to it
189 //Only declare ColumnDataWriter directory, filename, etc; do not set
190 //up otherwise
191
192     void EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
193     ↵ debugWriter);
194
195
196 /**
197 * Overridden GetAverageTransitCellCycleTime() method.
198 *
199 * @return the average of mMinCellCycleDuration and
200 mMaxCellCycleDuration
201 */
202
203 double GetAverageTransitCellCycleTime();
204
205
206 /**
207 * Overridden GetAverageStemCellCycleTime() method.
208 *
209 * @return the average of mMinCellCycleDuration and
210 mMaxCellCycleDuration
211 */
212
213 double GetAverageStemCellCycleTime();
214
215
216 /**
217 * Overridden OutputCellCycleModelParameters() method.
218 *
219 * @param rParamsFile the file stream to which the parameters are
220 output

```

```

197     */
198     virtual void OutputCellCycleModelParameters(out_stream& rParamsFile);
199 };
200
201 #include "SerializationExportWrapper.hpp"
202 // Declare identifier for the serializer
203 CHASTE_CLASS_EXPORT(BoijeCellCycleModel)
204
205 #endif /*BOIJECELLCYCLEMODEL_HPP_*/

```

---

### 16.2.20 /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)

```

---

### 16.2.21 /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

```

```
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 RetinalGanglion(unsigned colour=3);
42 /**
43  * Destructor.
44  */
45 virtual ~RetinalGanglion();
46
47 /**
48  * @return #mColour.
49  */
50 unsigned GetColour() const;
51 };
52
53 class AmacrinoHorizontal : public AbstractCellProperty
54 {
55 private:
56 /**
57  * 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;
76     }
77
78 public:
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
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
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_ */
```

## 16.2.22 /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

```

```

17             rModel.mSeed),
18             ↳ mp_PostMitoticType(rModel.mp_PostMitoticType),
19             ↳ mp_RPh_Type(rModel.mp_RPh_Type), mp_BC_Type(
20             rModel.mp_BC_Type), mp_AC_Type(rModel.mp_AC_Type),
21             ↳ mp_MG_Type(rModel.mp_MG_Type), mp_label_Type(
22             rModel.mp_label_Type)
23 {
24 }
25
26 AbstractCellCycleModel* GomesCellCycleModel::CreateCellCycleModel()
27 {
28     return new GomesCellCycleModel(*this);
29 }
30
31 /**
32 * CELL CYCLE DURATION RANDOM VARIABLE
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 void GomesCellCycleModel::ResetForDivision()
44 {
45 /**
46 * Mitotic mode rules
47 */
48 RandomNumberGenerator* p_random_number_generator =
49     ↳ RandomNumberGenerator::Instance();
50
51 /**
52 * MITOTIC MODE RANDOM VARIABLE

```

```
52     *****/
53     //initialise mitoticmode random variable, set mitotic mode
54     // appropriately after comparing to mode probability array
55     double mitoticModeRV = p_random_number_generator->ranf();
56
57     if (mitoticModeRV > mPP && mitoticModeRV ≤ mPP + mPD)
58     {
59         mMitoticMode = 1;
60     }
61     if (mitoticModeRV > mPP + mPD)
62     {
63         mMitoticMode = 2;
64     }
65
66     /**
67      * Write mitotic event to file if appropriate
68      * mDebug: many files: detailed per-lineage info switch; intended for
69      * TestHeInductionCountFixture
70      * mOutput: 1 file: time, seed, cellID, mitotic mode, intended for
71      * TestHeMitoticModeRateFixture
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)
```

```

90  {
91      mpCell→SetCellProliferativeType(mp_PostMitoticType);
92      mCellCycleDuration = DBL_MAX;
93      /*****
94      * SPECIFICATION RANDOM VARIABLE
95      *****/
96      double specificationRV = p_random_number_generator→ranf();
97      if (specificationRV ≤ mpMG)
98      {
99          mpCell→AddCellProperty(mp_MG_Type);
100     }
101    if (specificationRV > mpMG && specificationRV ≤ mpMG + mpAC)
102    {
103        mpCell→AddCellProperty(mp_AC_Type);
104    }
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>();

```



```
171         mpCell→AddCellProperty(mp_AC_Type);
172     }
173     if (specificationRV > mpMG + mpAC && specificationRV ≤ mpMG +
174         → mpAC + mpBC)
175     {
176         mpCell→AddCellProperty(mp_BC_Type);
177     }
178     if (specificationRV > mpMG + mpAC + mpBC)
179     {
180         mpCell→AddCellProperty(mp_RPh_Type);
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     }
```

```

210     }
211
212     /*****
213     * SEQUENCE SAMPLER
214     *****/
215     if (mSequenceSampler)
216     {
217         if (mSeqSamplerLabelSister)
218         {
219             mpCell->AddCellProperty(mp_label_Type);
220             mSeqSamplerLabelSister = false;
221         }
222         else
223         {
224             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<AbstractCellProper
248                                         → p_AC_Type,
```



```

276     mSequenceSampler = true;
277     mp_label_Type = label;
278 }
279
280 void
281   ↵ GomesCellCycleModel :: EnableModelDebugOutput(boost :: shared_ptr<ColumnDataWriter>
282   ↵ debugWriter)
283 {
284
285     mDebug = true;
286     mDebugWriter = debugWriter;
287
288     mTimeID = mDebugWriter->DefineUnlimitedDimension("Time", "h");
289     mVarIDs.push_back(mDebugWriter->DefineVariable("CellID", "No"));
290     mVarIDs.push_back(mDebugWriter->DefineVariable("CycleDuration",
291       ↵ "h"));
292     mVarIDs.push_back(mDebugWriter->DefineVariable("PP", "Percentile"));
293     mVarIDs.push_back(mDebugWriter->DefineVariable("PD", "Percentile"));
294     mVarIDs.push_back(mDebugWriter->DefineVariable("Dieroll",
295       ↵ "Percentile"));
296     mVarIDs.push_back(mDebugWriter->DefineVariable("MitoticMode",
297       ↵ "Mode"));
298
299     mDebugWriter->EndDefineMode();
300 }
301
302
303 void GomesCellCycleModel :: WriteDebugData(double percentileRoll)
304 {
305     double currentTime = SimulationTime::Instance()->GetTime();
306     CellPtr currentCell = GetCell();
307     double currentCellID = (double) currentCell->GetCellId();
308
309     mDebugWriter->PutVariable(mTimeID, currentTime);
310     mDebugWriter->PutVariable(mVarIDs[0], currentCellID);
311     mDebugWriter->PutVariable(mVarIDs[1], mCellCycleDuration);
312     mDebugWriter->PutVariable(mVarIDs[2], mPP);
313     mDebugWriter->PutVariable(mVarIDs[3], mPD);
314     mDebugWriter->PutVariable(mVarIDs[4], percentileRoll);
315     mDebugWriter->PutVariable(mVarIDs[5], mMitoticMode);
316     mDebugWriter->AdvanceAlongUnlimitedDimension();
317 }
318
319 ****
320 * 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";
331
332     // Call method on direct parent class
333
334     ↪ AbstractSimpleCellCycleModel::OutputCellCycleModelParameters(rParamsFile);
335
336 // Serialization for Boost ≥ 1.36
337 #include "SerializationExportWrapperForCpp.hpp"
338 CHASTE_CLASS_EXPORT(GomesCellCycleModel)

```

---

### 16.2.23 /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();
31 *
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
48 class GomesCellCycleModel : public AbstractSimpleCellCycleModel
49 {
50     friend class TestSimpleCellCycleModels;
51
52 private:
53
54     /** Needed for serialization.*/
55     friend class boost::serialization::access;
56     /**

```

```

47     * Archive the cell-cycle model and random number generator, never
48     → used directly - boost uses this.
49
50     *
51     * @param archive the archive
52     * @param version the current version of this class
53     */
54
55     template<class Archive>
56     void serialize(Archive & archive, const unsigned int version)
57     {
58         archive &
59         → boost::serialization::base_object<AbstractSimpleCellCycleModel>(*this);
60
61         SerializableSingleton<RandomNumberGenerator>* p_wrapper =
62
63             → RandomNumberGenerator::Instance()→GetSerializationWrapper();
64         archive & p_wrapper;
65         archive & mCellCycleDuration;
66     }
67
68     //Private write functions for models
69     void WriteModeEventOutput();
70     void WriteDebugData(double percentile);
71
72 protected:
73     //mode/output variables
74     bool mOutput;
75     double mEventStartTime;
76     bool mSequenceSampler;
77     bool mSeqSamplerLabelSister;
78     //debug writer stuff
79     bool mDebug;
80     int mTimeID;
81     std::vector<int> mVarIDs;
82     boost::shared_ptr<ColumnDataWriter> mDebugWriter;
83     //model parameters and state memory vars
84     double mNormalMu;
85     double mNormalSigma;
86     double mPP;
87     double mPD;
88     double mpBC;
89     double mpAC;
90     double mpMG;
91     unsigned mMitoticMode;

```

```

87     unsigned mSeed;
88     boost::shared_ptr<AbstractCellProperty> mp_PostMitoticType;
89     boost::shared_ptr<AbstractCellProperty> mp_RPh_Type;
90     boost::shared_ptr<AbstractCellProperty> mp_BC_Type;
91     boost::shared_ptr<AbstractCellProperty> mp_AC_Type;
92     boost::shared_ptr<AbstractCellProperty> mp_MG_Type;
93     boost::shared_ptr<AbstractCellProperty> mp_label_Type;
94
95     /**
96      * Protected copy-constructor for use by CreateCellCycleModel().
97      *
98      * The only way for external code to create a copy of a cell cycle
99      * model
100     * is by calling that method, to ensure that a model of the correct
101     * subclass is created.
102     * This copy-constructor helps subclasses to ensure that all member
103     * variables are correctly copied when this happens.
104     *
105     * This method is called by child classes to set member variables for
106     * a daughter cell upon cell division.
107     * Note that the parent cell cycle model will have had
108     * ResetForDivision() called just before CreateCellCycleModel() is
109     * called,
110     * so performing an exact copy of the parent is suitable behaviour.
111     * Any daughter-cell-specific initialisation
112     * can be done in InitialiseDaughterCell().
113     */
114     GomesCellCycleModel(const GomesCellCycleModel& rModel);
115
116 public:
117
118     /**
119      * Constructor - just a default, mBirthTime is set in the
120      * AbstractCellCycleModel class.
121      */
122     GomesCellCycleModel();
123
124     /**
125      * SetCellCycleDuration() method to set length of cell cycle
126      * (lognormal distribution as specified in [Gomes2011])
127      */
128
129
130

```

```

121 void SetCellCycleDuration();
122
123 /**
124 * Overridden builder method to create new copies of
125 * this cell-cycle model.
126 *
127 * @return new cell-cycle model
128 */
129 AbstractCellCycleModel* CreateCellCycleModel();
130
131 /**
132 * Overridden ResetForDivision() method.
133 * Contains general mitotic mode logic
134 */
135 void ResetForDivision();
136
137 /** Overridden InitialiseDaughterCell() method. Used to implement
138 * asymmetric mitotic mode*/
139 void InitialiseDaughterCell();
140
141 /* Model setup functions
142 * Set lognormal cell cycle curve properties with *mean and std of
143 * corresponding NORMAL curve*
144 * Model requires valid AbstractCellProperties to assign postmitotic
145 * fate;
146 * Defaults are found in GomesRetinalNeuralFates.hpp
147 * (RodPhotoreceptor, AmacrineCell, BipolarCell, MullerGlia)
148 */
149
150 void SetModelParameters(const double normalMu = 3.9716, const double
151 * normalSigma = .32839, const double PP = .055,
152 * const double PD = .221, const double pBC =
153 * .128, const double pAC = .106, const
154 * double pMG =
155 * .028);
156
157 void SetModelProperties(boost::shared_ptr<AbstractCellProperty>
158 * p_RPh_Type,
159 * boost::shared_ptr<AbstractCellProperty>
160 * p_AC_Type,
161 * boost::shared_ptr<AbstractCellProperty>
162 * p_BC_Type,
163 * boost::shared_ptr<AbstractCellProperty>
164 * p_MG_Type);

```

```

154
155 //This should normally be a DifferentiatedCellProliferativeType
156 void SetPostMitoticType(boost::shared_ptr<AbstractCellProperty>
157   → p_PostMitoticType);
158
159 //Functions to enable per-cell mitotic mode logging for mode rate &
160   → sequence sampling fixtures
161 //Uses singleton logfile
162 void EnableModeEventOutput(double eventStart, unsigned seed);
163 void EnableSequenceSampler(boost::shared_ptr<AbstractCellProperty>
164   → label);
165
166 //More detailed debug output. Needs a ColumnDataWriter passed to it
167 //Only declare ColumnDataWriter directory, filename, etc; do not set
168   → up otherwise
169 void EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
170   → debugWriter);
171
172 //Not used, but must be overwritten lest GomesCellCycleModels be
173   → abstract
174 double GetAverageTransitCellCycleTime();
175 double GetAverageStemCellCycleTime();
176
177 /**
178 * Overridden OutputCellCycleModelParameters() method.
179 *
180 * @param rParamsFile the file stream to which the parameters are
181   → output
182 */
183 virtual void OutputCellCycleModelParameters(out_stream& rParamsFile);
184 };
185
186 #include "SerializationExportWrapper.hpp"
187 // Declare identifier for the serializer
188 CHASTE_CLASS_EXPORT(GomesCellCycleModel)
189
190 #endif /*GOMESCELLCYCLEMODEL_HPP_*/

```

---

### 16.2.24 /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)

```

---

### 16.2.25 /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     /**
22      * Archive the member variables.
23      *
24      * @param archive the archive
25      * @param version the current version of this class
26      */
27     template<class Archive>
28     void serialize(Archive & archive, const unsigned int version)
29     {
30         archive &
31             boost::serialization::base_object<AbstractCellProperty>(*this);
32         archive & mColour;
33     }
34
35 public:
36
37     /**
38      * Constructor.
39      *
40      * @param colour what colour cells with this property should be in
41      * the visualizer (defaults to 6)
42      */
43     RodPhotoreceptor(unsigned colour=3);
44
45     virtual ~RodPhotoreceptor();
46
47     /**
48      * @return #mColour.
49      */
50     unsigned GetColour() const;
51 };
52
53 class AmacrineCell : 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      * 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;  
76     }  
77  
78 public:  
79  
80     /**  
81      * Constructor.  
82      *  
83      * @param colour what colour cells with this property should be in  
84      * the visualizer (defaults to 6)  
85      */  
86     AmacrineCell(unsigned colour=4);  
87  
88     /**  
89      * Destructor.  
90      */  
91     virtual ~AmarineCell();  
92  
93     /**  
94      * @return #mColour.  
95      */  
96     unsigned GetColour() const;  
97 };  
98  
99 class BipolarCell : public AbstractCellProperty  
100 {
```

```
99 private:
100
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
123 public:
124
125     /**
126      * Constructor.
127      *
128      * @param colour what colour cells with this property should be in
129      * the visualizer (defaults to 6)
130      */
131     BipolarCell(unsigned colour=5);
132
133     /**
134      * Destructor.
135      */
136     virtual ~BipolarCell();
137
138     /**
139      * @return #mColour.
140      */
141     unsigned GetColour() const;
142
143 };
```

```
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.
```

```

181     */
182     unsigned GetColour() const;
183 };
184 #include "SerializationExportWrapper.hpp"
185 // Declare identifier for the serializer
186 CHASTE_CLASS_EXPORT(RodPhotoreceptor)
187 CHASTE_CLASS_EXPORT(AmacrineCell)
188 CHASTE_CLASS_EXPORT(BipolarCell)
189 CHASTE_CLASS_EXPORT(MullerGlia)
190
191 #endif /* GOMESRETINALNEURALFATES_HPP_ */

```

---

### 16.2.26 /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)

```

---

### 16.2.27 /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     /**
22      * Archive the member variables.
23      *
24      * @param archive the archive
25      * @param version the current version of this class
26      */
27     template<class Archive>
28     void serialize(Archive & archive, const unsigned int version)
29     {
30         archive &
31             boost::serialization::base_object<AbstractCellProperty>(*this);
32         archive & mColour;
33     }
34
35 public:
36
37     /**
38      * Constructor.
39      *
40      * @param colour what colour cells with this property should be in
41      * the visualizer (defaults to 6)
42      */
43     Ath5Mo(unsigned colour=3);
44
45     /**
46      * Destructor.
47      */
48     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 */

```

---

### 16.2.28 /src/HeCellCycleModel.cpp

```

1 #include "HeCellCycleModel.hpp"
2
3 HeCellCycleModel::HeCellCycleModel() :
4     AbstractSimpleCellCycleModel(), mKillSpecified(false),
5     mDeterministic(false), mOutput(false), mEventStartTime(
6         24.0), mSequenceSampler(false),
7     mSeqSamplerLabelSister(false), mDebug(false),
8     mTimeID(), mVarIDs(), mDebugWriter(), mTiLOffset(
9         0.0), mGammaShift(4.0), mGammaShape(2.0),
10    mGammaScale(1.0), mSisterShiftWidth(1),
11    mMitoticModePhase2(
12        8.0), mMitoticModePhase3(15.0), mPhaseShiftWidth(2.0),
13    mPhase1PP(1.0), mPhase1PD(0.0), mPhase2PP(0.2),
14    mPhase2PD(
15        0.4), mPhase3PP(0.2), mPhase3PD(0.0), mMitoticMode(0),
16    mSeed(0), mTimeDependentCycleDuration(false),
17    mPeakRateTime(), mIncreasingRateSlope(),
18    mDecreasingRateSlope(), mBaseGammaScale()
19 {
20     mReadyToDivide = true; //He model begins with a first division
21 }
22
23 HeCellCycleModel::HeCellCycleModel(const HeCellCycleModel& rModel) :
24     AbstractSimpleCellCycleModel(rModel),
25     mKillSpecified(rModel.mKillSpecified), mDeterministic(

```

```

15         rModel.mDeterministic), mOutput(rModel.mOutput),
16         ↳ mEventStartTime(rModel.mEventStartTime),
17         ↳ mSequenceSampler(
18             rModel.mSequenceSampler),
19             ↳ mSeqSamplerLabelSister(rModel.mSeqSamplerLabelSister),
20             ↳ mDebug(rModel.mDebug), mTimeID(
21                 rModel.mTimeID), mVarIDs(rModel.mVarIDs),
22                 ↳ mDebugWriter(rModel.mDebugWriter), mTiLOffset(
23                     rModel.mTiLOffset), mGammaShift(rModel.mGammaShift),
24                     ↳ mGammaShape(rModel.mGammaShape), mGammaScale(
25                         rModel.mGammaScale),
26                         ↳ mSisterShiftWidth(rModel.mSisterShiftWidth),
27                         ↳ mMitoticModePhase2(
28                             rModel.mMitoticModePhase2),
29                             ↳ mMitoticModePhase3(rModel.mMitoticModePhase3),
30                             ↳ mPhaseShiftWidth(
31                                 rModel.mPhaseShiftWidth), mPhase1PP(rModel.mPhase1PP),
32                                 ↳ mPhase1PD(rModel.mPhase1PD), mPhase2PP(
33                                     rModel.mPhase2PP), mPhase2PD(rModel.mPhase2PD),
34                                     ↳ mPhase3PP(rModel.mPhase3PP), mPhase3PD(
35                                         rModel.mPhase3PD), mMitoticMode(rModel.mMitoticMode),
36                                         ↳ mSeed(rModel.mSeed), mTimeDependentCycleDuration(
37                                             rModel.mTimeDependentCycleDuration),
38                                             ↳ mPeakRateTime(rModel.mPeakRateTime),
39                                             ↳ mIncreasingRateSlope(
40                                                 rModel.mIncreasingRateSlope),
41                                                 ↳ mDecreasingRateSlope(rModel.mDecreasingRateSlope),
42                                                 ↳ mBaseGammaScale(
43                                                     rModel.mBaseGammaScale)
44     {
45 }
46
47 AbstractCellCycleModel* HeCellCycleModel::CreateCellCycleModel()
48 {
49     return new HeCellCycleModel(*this);
50 }
51
52 void HeCellCycleModel::SetCellCycleDuration()
53 {
54     RandomNumberGenerator* p_random_number_generator =
55         ↳ RandomNumberGenerator::Instance();
56
57 /*****
58 *****/

```

```

40     * CELL CYCLE DURATION RANDOM VARIABLE
41     *****/
42
43     if (!mTimeDependentCycleDuration) //Normal operation, cell cycle
44         → length stays constant
45     {
46         //He cell cycle length determined by shifted gamma distribution
47         → reflecting 4 hr refractory period followed by gamma pdf
48         mCellCycleDuration = mGammaShift +
49             → p_random_number_generator→GammaRandomDeviate(mGammaShape,
50                 → mGammaScale);
51
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         {
66             mGammaScale = std::max((mBaseGammaScale - currTime *
67             → mIncreasingRateSlope), .000000000001);
68         }
69         if (currTime > mPeakRateTime)
70         {
71             mGammaScale = std::max(
72                 ((mBaseGammaScale - mPeakRateTime *
73                     → mIncreasingRateSlope)
74                     + (mBaseGammaScale + (currTime -
75                         → mPeakRateTime) * mDecreasingRateSlope)),
76                     .000000000001);
77         }
78         mCellCycleDuration = mGammaShift +
79             → p_random_number_generator→GammaRandomDeviate(mGammaShape,
80                 → mGammaScale);
81     }
82
83 }
84
85 }
```

```

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
93     ****
94     if (currentTiL > mMitoticModePhase2 && currentTiL <
95         → mMitoticModePhase3)
96     {
97         //if current TiL is > phase 2 boundary time, set the currentPhase
98         → appropriately
99         currentPhase = 2;
100
101         //if deterministic mode is enabled, PD divisions are guaranteed
102         → unless this is an Ath5 morphant
103         if (mDeterministic)
104         {
105             mMitoticMode = 1; //0=PP;1=PD;2=DD
106             if (mpCell→HasCellProperty<Ath5Mo>()) //Ath5 morphants
107                 → undergo PP rather than PD divisions in 80% of cases
108             {
109                 double ath5RV = p_random_number_generator→ranf();
110                 if (ath5RV ≤ .8)
111                 {
112                     mMitoticMode = 0;
113                 }
114             }
115         }
116     }
117 }
```

```

109 }
110
111 if (currentTiL > mMitoticModePhase3)
112 {
113     //if current TiL is > phase 3 boundary time, set the currentPhase
114     //appropriately
115     currentPhase = 3;
116     if (mDeterministic)
117     {
118         //if deterministic mode is enabled, DD divisions are
119         //guaranteed
120         mMitoticMode = 2;
121     }
122
123 /**
124 * MITOTIC MODE RANDOM VARIABLE
125 */
126 //initialise mitoticmode random variable, set mitotic mode
127 //appropriately after comparing to mode probability matrix
128 double mitoticModeRV = p_random_number_generator->ranf(); //0-1
129 //evenly distributed RV
130
131 if (!mDeterministic)
132 {
133     //construct 3x2 matrix of mode probabilities arranged by phase
134     double modeProbabilityMatrix[3][2] = { { mPhase1PP, mPhase1PD },
135     { mPhase2PP, mPhase2PD }, { mPhase3PP,
136
137     //if the RV is > currentPhasePP && <= currentPhasePD, change
138     //mMitoticMode from PP to PD
139     if (mitoticModeRV > modeProbabilityMatrix[currentPhase - 1][0]
140         && mitoticModeRV
141             <= modeProbabilityMatrix[currentPhase - 1][0] +
142                 modeProbabilityMatrix[currentPhase - 1][1])
143     {
144         mMitoticMode = 1;
145         if (mpCell->HasCellProperty<Ath5Mo>()) //Ath5 morphants
146             undergo PP rather than PD divisions in 80% of cases

```

```

141         {
142             double ath5RV = p_random_number_generator->ranf();
143             if (ath5RV <=.8)
144             {
145                 mMitoticMode = 0;
146             }
147         }
148     }
149     //if the RV is > currentPhasePP + currentPhasePD, change
150     //→ mMitoticMode from PP to DD
151     if (mitoticModeRV > modeProbabilityMatrix[currentPhase - 1][0] +
152         modeProbabilityMatrix[currentPhase - 1][1])
153     {
154         mMitoticMode = 2;
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 * Symmetric postmitotic specification rule
174 * -(asymmetric postmitotic rule specified in
175 //→ InitialiseDaughterCell());
176 */
177 if (mMitoticMode == 2)
178 {
179     boost::shared_ptr<AbstractCellProperty> p_PostMitoticType =

```

```

179             → mpCell→rGetCellPropertyCollection().GetCellPropertyRegistry()→G
180             mpCell→SetCellProliferativeType(p_PostMitoticType);
181             mCellCycleDuration = DBL_MAX;
182
183         if(mKillSpecified)
184         {
185             mpCell→Kill();
186         }
187     }
188
189     /*****
190      * SEQUENCE SAMPLER
191      *****/
192     //if the sequence sampler has been turned on, check for the label &
193     → write mitotic mode to log
194     //50% chance of each daughter cell from a mitosis inheriting the
195     → label
196     if (mSequenceSampler)
197     {
198         if (mpCell→HasCellProperty<CellLabel>())
199         {
200             (*LogFile::Instance()) << mMitoticMode;
201             double labelRV = p_random_number_generator→ranf();
202             if (labelRV ≤ .5)
203             {
204                 mSeqSamplerLabelSister = true;
205                 mpCell→RemoveCellProperty<CellLabel>();
206             }
207             else
208             {
209                 mSeqSamplerLabelSister = false;
210             }
211         }
212         else
213         {
214             //prevents lost-label cells from labelling their progeny
215             mSeqSamplerLabelSister = false;
216         }
217     }
218 void HeCellCycleModel::Initialise()

```



```

252     mCellCycleDuration = (mGammaShift +
253         ↳ p_random_number_generator→GammaRandomDeviate(mGammaShape,
254             ↳ mGammaScale))
255         + c;
256     }
257
258 void HeCellCycleModel::InitialiseDaughterCell()
259 {
260     RandomNumberGenerator* p_random_number_generator =
261         ↳ RandomNumberGenerator::Instance();
262
263     /*****
264      * PD-type division & shifted sister cycle length & boundary
265      * adjustments
266      *****/
267
268     if (mMitoticMode == 1) //RPC becomes specified retinal neuron in
269         ↳ asymmetric PD mitosis
270     {
271         boost::shared_ptr<AbstractCellProperty> p_PostMitoticType =
272
273             ↳ mpCell→rGetCellPropertyCollection().GetCellPropertyRegistry()→G
274         mpCell→SetCellProliferativeType(p_PostMitoticType);
275         mCellCycleDuration = DBL_MAX;
276
277         if(mKillSpecified)
278         {
279             mpCell→Kill();
280         }
281
282         //daughter cell's mCellCycleDuration is copied from parent; modified
283         ↳ by a normally distributed shift if it remains proliferative
284     if (mMitoticMode == 0)
285     {
286         double sisterShift =
287             ↳ p_random_number_generator→NormalRandomDeviate(0,
288                 ↳ mSisterShiftWidth); //random variable mean 0 SD 1 by default
289         mCellCycleDuration = std::max(mGammaShift, mCellCycleDuration +
290             ↳ sisterShift); // sister shift respects 4 hour refractory
291             ↳ period

```

```

284 }
285
286 //deterministic model phase boundary division shift for daughter
287 // cells
288 if (mDeterministic)
289 {
290     //shift phase boundaries to reflect error in "timer" after
291     // division
292     double phaseShift =
293         → p_random_number_generator→NormalRandomDeviate(0,
294         → mPhaseShiftWidth);
295     mMitoticModePhase2 = mMitoticModePhase2 + phaseShift;
296     mMitoticModePhase3 = mMitoticModePhase3 + phaseShift;
297 }
298
299 /**
300 * SEQUENCE SAMPLER
301 *****/
302 if (mSequenceSampler)
303 {
304     if (mSeqSamplerLabelSister)
305     {
306         boost::shared_ptr<AbstractCellProperty> p_label_type =
307
308             → mpCell→rGetCellPropertyCollection().GetCellPropertyRegistry(
309             mpCell→AddCellProperty(p_label_type);
310             mSeqSamplerLabelSister = false;
311     }
312     else
313     {
314         mpCell→RemoveCellProperty<CellLabel>();
315     }
316 }
317
318 if (mMitoticMode == 2 && mKillSpecified) mpCell→Kill();
319 }
320
321 void HeCellCycleModel::SetModelParameters(double t1Offset, double
322     mitoticModePhase2, double mitoticModePhase3,
323
324     double phase1PP, double
325     → phase1PD, double phase2PP,
326     → double phase2PD,
327
328     double phase3PP, double
329     → phase3PD, double phase4PP,
330     → double phase4PD,
331
332     double phase5PP, double
333     → phase5PD, double phase6PP,
334     → double phase6PD,
335
336     double phase7PP, double
337     → phase7PD, double phase8PP,
338     → double phase8PD,
339
340     double phase9PP, double
341     → phase9PD, double phase10PP,
342     → double phase10PD,
343
344     double phase11PP, double
345     → phase11PD, double phase12PP,
346     → double phase12PD,
347
348     double phase13PP, double
349     → phase13PD, double phase14PP,
350     → double phase14PD,
351
352     double phase15PP, double
353     → phase15PD, double phase16PP,
354     → double phase16PD,
355
356     double phase17PP, double
357     → phase17PD, double phase18PP,
358     → double phase18PD,
359
360     double phase19PP, double
361     → phase19PD, double phase20PP,
362     → double phase20PD,
363
364     double phase21PP, double
365     → phase21PD, double phase22PP,
366     → double phase22PD,
367
368     double phase23PP, double
369     → phase23PD, double phase24PP,
370     → double phase24PD,
371
372     double phase25PP, double
373     → phase25PD, double phase26PP,
374     → double phase26PD,
375
376     double phase27PP, double
377     → phase27PD, double phase28PP,
378     → double phase28PD,
379
380     double phase29PP, double
381     → phase29PD, double phase30PP,
382     → double phase30PD,
383
384     double phase31PP, double
385     → phase31PD, double phase32PP,
386     → double phase32PD,
387
388     double phase33PP, double
389     → phase33PD, double phase34PP,
390     → double phase34PD,
391
392     double phase35PP, double
393     → phase35PD, double phase36PP,
394     → double phase36PD,
395
396     double phase37PP, double
397     → phase37PD, double phase38PP,
398     → double phase38PD,
399
400     double phase39PP, double
401     → phase39PD, double phase40PP,
402     → double phase40PD,
403
404     double phase41PP, double
405     → phase41PD, double phase42PP,
406     → double phase42PD,
407
408     double phase43PP, double
409     → phase43PD, double phase44PP,
410     → double phase44PD,
411
412     double phase45PP, double
413     → phase45PD, double phase46PP,
414     → double phase46PD,
415
416     double phase47PP, double
417     → phase47PD, double phase48PP,
418     → double phase48PD,
419
420     double phase49PP, double
421     → phase49PD, double phase50PP,
422     → double phase50PD,
423
424     double phase51PP, double
425     → phase51PD, double phase52PP,
426     → double phase52PD,
427
428     double phase53PP, double
429     → phase53PD, double phase54PP,
430     → double phase54PD,
431
432     double phase55PP, double
433     → phase55PD, double phase56PP,
434     → double phase56PD,
435
436     double phase57PP, double
437     → phase57PD, double phase58PP,
438     → double phase58PD,
439
440     double phase59PP, double
441     → phase59PD, double phase60PP,
442     → double phase60PD,
443
444     double phase61PP, double
445     → phase61PD, double phase62PP,
446     → double phase62PD,
447
448     double phase63PP, double
449     → phase63PD, double phase64PP,
450     → double phase64PD,
451
452     double phase65PP, double
453     → phase65PD, double phase66PP,
454     → double phase66PD,
455
456     double phase67PP, double
457     → phase67PD, double phase68PP,
458     → double phase68PD,
459
460     double phase69PP, double
461     → phase69PD, double phase70PP,
462     → double phase70PD,
463
464     double phase71PP, double
465     → phase71PD, double phase72PP,
466     → double phase72PD,
467
468     double phase73PP, double
469     → phase73PD, double phase74PP,
470     → double phase74PD,
471
472     double phase75PP, double
473     → phase75PD, double phase76PP,
474     → double phase76PD,
475
476     double phase77PP, double
477     → phase77PD, double phase78PP,
478     → double phase78PD,
479
480     double phase79PP, double
481     → phase79PD, double phase80PP,
482     → double phase80PD,
483
484     double phase81PP, double
485     → phase81PD, double phase82PP,
486     → double phase82PD,
487
488     double phase83PP, double
489     → phase83PD, double phase84PP,
490     → double phase84PD,
491
492     double phase85PP, double
493     → phase85PD, double phase86PP,
494     → double phase86PD,
495
496     double phase87PP, double
497     → phase87PD, double phase88PP,
498     → double phase88PD,
499
500     double phase89PP, double
501     → phase89PD, double phase90PP,
502     → double phase90PD,
503
504     double phase91PP, double
505     → phase91PD, double phase92PP,
506     → double phase92PD,
507
508     double phase93PP, double
509     → phase93PD, double phase94PP,
510     → double phase94PD,
511
512     double phase95PP, double
513     → phase95PD, double phase96PP,
514     → double phase96PD,
515
516     double phase97PP, double
517     → phase97PD, double phase98PP,
518     → double phase98PD,
519
520     double phase99PP, double
521     → phase99PD, double phase100PP,
522     → double phase100PD,
523
524     double phase101PP, double
525     → phase101PD, double phase102PP,
526     → double phase102PD,
527
528     double phase103PP, double
529     → phase103PD, double phase104PP,
530     → double phase104PD,
531
532     double phase105PP, double
533     → phase105PD, double phase106PP,
534     → double phase106PD,
535
536     double phase107PP, double
537     → phase107PD, double phase108PP,
538     → double phase108PD,
539
540     double phase109PP, double
541     → phase109PD, double phase110PP,
542     → double phase110PD,
543
544     double phase111PP, double
545     → phase111PD, double phase112PP,
546     → double phase112PD,
547
548     double phase113PP, double
549     → phase113PD, double phase114PP,
550     → double phase114PD,
551
552     double phase115PP, double
553     → phase115PD, double phase116PP,
554     → double phase116PD,
555
556     double phase117PP, double
557     → phase117PD, double phase118PP,
558     → double phase118PD,
559
560     double phase119PP, double
561     → phase119PD, double phase120PP,
562     → double phase120PD,
563
564     double phase121PP, double
565     → phase121PD, double phase122PP,
566     → double phase122PD,
567
568     double phase123PP, double
569     → phase123PD, double phase124PP,
570     → double phase124PD,
571
572     double phase125PP, double
573     → phase125PD, double phase126PP,
574     → double phase126PD,
575
576     double phase127PP, double
577     → phase127PD, double phase128PP,
578     → double phase128PD,
579
580     double phase129PP, double
581     → phase129PD, double phase130PP,
582     → double phase130PD,
583
584     double phase131PP, double
585     → phase131PD, double phase132PP,
586     → double phase132PD,
587
588     double phase133PP, double
589     → phase133PD, double phase134PP,
590     → double phase134PD,
591
592     double phase135PP, double
593     → phase135PD, double phase136PP,
594     → double phase136PD,
595
596     double phase137PP, double
597     → phase137PD, double phase138PP,
598     → double phase138PD,
599
599 }
```

```

318                               double phase3PP, double
319                               ↵ phase3PD, double
320                               ↵ gammaShift, double
321                               ↵ gammaShape,
322                               double gammaScale, double
323                               ↵ sisterShift)
324 {
325     mTiLOffset = tiLOffset;
326     mMitoticModePhase2 = mitoticModePhase2;
327     mMitoticModePhase3 = mitoticModePhase3;
328     mPhase1PP = phase1PP;
329     mPhase1PD = phase1PD;
330     mPhase2PP = phase2PP;
331     mPhase2PD = phase2PD;
332     mPhase3PP = phase3PP;
333     mPhase3PD = phase3PD;
334     mGammaShift = gammaShift;
335     mGammaShape = gammaShape;
336     mGammaScale = gammaScale;
337     mSisterShiftWidth = sisterShift;
338 }
339
340 void HeCellCycleModel::SetDeterministicMode(double tiLOffset, double
341     ↵ mitoticModePhase2, double mitoticModePhase3,
342                               double phaseShiftWidth,
343                               ↵ double gammaShift, double
344                               ↵ gammaShape,
345                               double gammaScale, double
346                               ↵ sisterShift)
347 {
348     mDeterministic = true;
349     mTiLOffset = tiLOffset;
350     mMitoticModePhase2 = mitoticModePhase2;
351     mMitoticModePhase3 = mitoticModePhase3;
352     mPhaseShiftWidth = phaseShiftWidth;
353     mGammaShift = gammaShift;
354     mGammaShape = gammaShape;
355     mGammaScale = gammaScale;
356     mSisterShiftWidth = sisterShift;
357 }
358
359 void HeCellCycleModel::SetTimeDependentCycleDuration(double peakRateTime,
360     ↵ double increasingSlope,

```

```

352                               double
353                               ↵   decreasingSlope)
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
367     ↵   seed)
368 {
369     mOutput = true;
370     mEventStartTime = eventStart;
371     mSeed = seed;
372 }
373
374 void HeCellCycleModel::WriteModeEventOutput()
375 {
376     double currentTime = SimulationTime::Instance()->GetTime() +
377         ↵   mEventStartTime;
378     CellPtr currentCell = GetCell();
379     double currentCellID = (double) currentCell->GetCellId();
380     (*LogFile::Instance()) << currentTime << "\t" << mSeed << "\t" <<
381         ↵   currentCellID << "\t" << mMitoticMode << "\n";
382 }
383
384 void HeCellCycleModel::EnableSequenceSampler()
385 {
386     mSequenceSampler = true;
387     boost::shared_ptr<AbstractCellProperty> p_label_type =
388
389         ↵   mpCell->rGetCellPropertyCollection().GetCellPropertyRegistry()->Get<C
390     mpCell->AddCellProperty(p_label_type);
391 }
392
393

```

```

389 void
400   → HeCellCycleModel::PassDebugWriter(boost::shared_ptr<ColumnDataWriter>
401   → debugWriter, int timeID,
402                                         std::vector<int> varIDs)
403 {
404   mDebug = true;
405   mDebugWriter = debugWriter;
406   mTimeID = timeID;
407   mVarIDs = varIDs;
408 }
409
410 void
411   → HeCellCycleModel::EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
412   → debugWriter)
413 {
414   mDebug = true;
415   mDebugWriter = debugWriter;
416
417   mTimeID = mDebugWriter->DefineUnlimitedDimension("Time", "h");
418
419   mVarIDs.push_back(mDebugWriter->DefineVariable("CellID", "No"));
420   mVarIDs.push_back(mDebugWriter->DefineVariable("TiL", "h"));
421   mVarIDs.push_back(mDebugWriter->DefineVariable("CycleDuration",
422     → "h"));
423   mVarIDs.push_back(mDebugWriter->DefineVariable("Phase2Boundary",
424     → "h"));
425   mVarIDs.push_back(mDebugWriter->DefineVariable("Phase3Boundary",
426     → "h"));
427   mVarIDs.push_back(mDebugWriter->DefineVariable("Phase", "No"));
428   mVarIDs.push_back(mDebugWriter->DefineVariable("MitoticModeRV",
429     → "Percentile"));
430   mVarIDs.push_back(mDebugWriter->DefineVariable("MitoticMode",
431     → "Mode"));
432   mVarIDs.push_back(mDebugWriter->DefineVariable("Label", "binary"));
433
434   mDebugWriter->EndDefineMode();
435 }
436
437 void HeCellCycleModel::WriteDebugData(double currentTime, unsigned phase,
438   → double mitoticModeRV)
439 {
440   double currentTiL = SimulationTime::Instance()->GetTime();
441   double currentCellID = mpCell->GetCellId();

```

```
422     unsigned label = 0;
423     if (mpCell->HasCellProperty<CellLabel>()) label = 1;
424
425     mDebugWriter->PutVariable(mTimeID, currentTime);
426     mDebugWriter->PutVariable(mVarIDs[0], currentCellID);
427     mDebugWriter->PutVariable(mVarIDs[1], currentTiL);
428     mDebugWriter->PutVariable(mVarIDs[2], mCellCycleDuration);
429     mDebugWriter->PutVariable(mVarIDs[3], mMitoticModePhase2);
430     mDebugWriter->PutVariable(mVarIDs[4], mMitoticModePhase3);
431     mDebugWriter->PutVariable(mVarIDs[5], phase);
432     if (!mDeterministic)
433     {
434         mDebugWriter->PutVariable(mVarIDs[6], mitoticModeRV);
435     }
436     mDebugWriter->PutVariable(mVarIDs[7], mMitoticMode);
437     if (mSequenceSampler)
438     {
439         mDebugWriter->PutVariable(mVarIDs[8], label);
440     }
441     mDebugWriter->AdvanceAlongUnlimitedDimension();
442 }
443
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
```

```

463     ↳ AbstractSimpleCellCycleModel::OutputCellCycleModelParameters(rParamsFile);
464 }
465
466 // Serialization for Boost ≥ 1.36
467 #include "SerializationExportWrapperForCpp.hpp"
468 CHASTE_CLASS_EXPORT(HeCellCycleModel)

```

---

### 16.2.29 /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 ****
16 * HE CELL CYCLE MODEL
17 * As described in He et al. 2012 [He2012] doi:
18 *   ↳ 10.1016/j.neuron.2012.06.033
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>);

```

```

27  *
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  * by a ColumnDataWriter passed to it from the test
34  * (eg. by the SetupDebugOutput helper function in the project simulator)
35  *
36  *
37  *****/
38
39 class HeCellCycleModel : public AbstractSimpleCellCycleModel
40 {
41     friend class TestSimpleCellCycleModels;
42
43 private:
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
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         archive & p_wrapper;
64         archive & mCellCycleDuration;
65     }

```

```
63 //Private write functions for models
64 void WriteModeEventOutput();
65 void WriteDebugData(double currTiL, unsigned phase, double
66 → percentile);
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 public:  
128  
129     /**  
130     * Constructor - just a default, mBirthTime is set in the  
131     * AbstractCellCycleModel class.  
132     */  
133 HeCellCycleModel();  
134  
135     /**  
136     * SetCellCycleDuration() method to set length of cell cycle  
137     */  
138 void SetCellCycleDuration();  
139  
140     /**  
141     * Overridden builder method to create new copies of  
142     * this cell-cycle model.  
143     *  
144     * @return new cell-cycle model  
145     */  
146 AbstractCellCycleModel* CreateCellCycleModel();  
147  
148
```

```

140  /**
141   * Overridden ResetForDivision() method.
142   * Contains general mitotic mode logic
143   */
144  void ResetForDivision();

145
146  /**
147   * Overridden Initialise() method
148   * Used to give an appropriate mCellCycleDuration to cells w/ TiL
149   * offsets
150   * sets mReadytoDivide to false as appropriate
151   * Initialises as transit proliferative type
152   */
153  void Initialise();

154  /**
155   * Overridden InitialiseDaughterCell() method.
156   * Used to apply sister-cell time shifting (cell cycle duration,
157   * deterministic phase boundaries)
158   * Used to implement asymmetric mitotic mode
159   */
160  void InitialiseDaughterCell();

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  void SetModelParameters(double tiLOffset = 0, double
170   * mitoticModePhase2 = 8, double mitoticModePhase3 = 15,
171   * double phase1PP = 1, double phase1PD = 0,
172   * double phase2PP = .2, double phase2PD =
173   * .4,
174   * double phase3PP = .2, double phase3PD = 0,
175   * double gammaShift = 4, double gammaShape
176   * = 2,
177   * double gammaScale = 1, double sisterShift =
178   * 1);
179  void SetDeterministicMode(double tiLOffset = 0, double
180   * mitoticModePhase2 = 8, double mitoticModePhase3 = 15,

```

```

171         double phaseShiftWidth = 1, double
172             ↵ gammaShift = 4, double gammaShape = 2,
173             double gammaScale = 1, double sisterShift =
174                 ↵ 1);
175     void SetTimeDependentCycleDuration(double peakRateTime, double
176         ↵ increasingSlope, double decreasingSlope);
177
178     //Function to set mKillSpecified = true; marks specified neurons for
179         ↵ death and removal from population
180     //Intended to help w/ resource consumption for WanSimulator
181     void EnableKillSpecified();
182
183     //Functions to enable per-cell mitotic mode logging for mode rate &
184         ↵ sequence sampling fixtures
185     //Uses singleton logfile
186     void EnableModeEventOutput(double eventStart, unsigned seed);
187     void EnableSequenceSampler();
188
189     //More detailed debug output. Needs a ColumnDataWriter passed to it
190     //Only declare ColumnDataWriter directory, filename, etc; do not set
191         ↵ up otherwise
192     //Use PassDebugWriter if the writer is already enabled elsewhere (ie.
193         ↵ in a Wan stem cell cycle model)
194     void EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
195         ↵ debugWriter);
196     void PassDebugWriter(boost::shared_ptr<ColumnDataWriter> debugWriter,
197         ↵ int timeID, std::vector<int> varIDs);
198
199     //Not used, but must be overwritten lest HeCellCycleModels be
200         ↵ abstract
201     double GetAverageTransitCellCycleTime();
202     double GetAverageStemCellCycleTime();
203
204     /**
205      * Overridden OutputCellCycleModelParameters() method.
206      *
207      * @param rParamsFile the file stream to which the parameters are
208          ↵ output
209      */
210     virtual void OutputCellCycleModelParameters(out_stream& rParamsFile);
211 };
212
213 #include "SerializationExportWrapper.hpp"

```

```

203 // Declare identifier for the serializer
204 CHASTE_CLASS_EXPORT(HeCellCycleModel)
205
206 #endif /*HECELLCYCLEMODEL_HPP_*/

```

---

### 16.2.30 /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>::OffLatticeSimulationPropertyStop(
11     rCellPopulation,
12     bool deleteCellPopulationInDestructor,
13     bool initialiseCells
14 ) :
15     AbstractCellBasedSimulation<ELEMENT_DIM,SPACE_DIM>(rCellPopulation,
16     deleteCellPopulationInDestructor, initialiseCells),
17     p_property()
18 {
19     if (
20         !dynamic_cast<AbstractOffLatticeCellPopulation<ELEMENT_DIM,SPACE_DIM>>(&rCellPopulation)
21     ) {
22         EXCEPTION("OffLatticeSimulations require a subclass of "
23             "AbstractOffLatticeCellPopulation.");
24     }
25 }
26
27 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
28 bool OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::StoppingEventHasOccurred()
29 {
30     if(p_property->GetCellCount()<1){
31         return true;
32     }
33     else{

```

```
30             return false;
31         }
32     }
33
34 //Public access to Stopping Event bool
35 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
36 bool
37     → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::HasStoppingEventOccurred
38 {
39     return StoppingEventHasOccurred();
40 }
41 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
42 void
43     → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::SetStopProperty(boost::shared_p
44     → stopPropertySetting)
45 {
46     p_property = stopPropertySetting;
47 }
48 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
49 void
50     → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::AddForce(boost::shared_p
51     → pForce)
52 {
53     mForceCollection.push_back(pForce);
54 }
55 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
56 void
57     → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::RemoveAllForces()
58 {
59     mForceCollection.clear();
60 }
61 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
62 void
63     → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::AddCellPopulationBoundaryC
64     → pBoundaryCondition)
65 {
66     mBoundaryConditions.push_back(pBoundaryCondition);
67 }
```

```

65 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
66 void
67   → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::RemoveAllCellPopulationB
68 {
69   mBoundaryConditions.clear();
70 }
71
72 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
73 void
74   → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::SetNumericalMethod(boost
75   → SPACE_DIM> > pNumericalMethod)
76 {
77   mpNumericalMethod = pNumericalMethod;
78 }
79
80 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
81 const boost::shared_ptr<AbstractNumericalMethod<ELEMENT_DIM, SPACE_DIM> >
82   → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::GetNumericalMethod()
83   → const
84 {
85   return mpNumericalMethod;
86 }
87
88 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
89 const std::vector<boost::shared_ptr<AbstractForce<ELEMENT_DIM, SPACE_DIM>
90   → > >&
91   → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::rGetForceCollection()
92   → const
93 {
94   return mForceCollection;
95 }
96
97 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
98 void
99   → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::UpdateCellLocationsAndTo
100 {
101   CellBasedEventHandler::BeginEvent(CellBasedEventHandler::POSITION);
102
103   double time_advanced_so_far = 0;
104   double target_time_step = this→mDt;
105   double present_time_step = this→mDt;
106
107   while (time_advanced_so_far < target_time_step)

```

```

99  {
100     // Store the initial node positions (these may be needed when
101     // applying boundary conditions)
102     std::map<Node<SPACE_DIM>*, c_vector<double, SPACE_DIM> >
103     old_node_locations;
104
105     for (typename AbstractMesh<ELEMENT_DIM, SPACE_DIM>::NodeIterator
106         node_iter =
107         this->mrCellPopulation.rGetMesh().GetNodeIteratorBegin();
108         node_iter !=
109         this->mrCellPopulation.rGetMesh().GetNodeIteratorEnd();
110         ++node_iter)
111     {
112         old_node_locations[&(*node_iter)] =
113             (node_iter)->rGetLocation();
114     }
115
116     // Try to update node positions according to the numerical method
117     try
118     {
119         mpNumericalMethod->UpdateAllNodePositions(present_time_step);
120         ApplyBoundaries(old_node_locations);
121
122         // Successful time step! Update time_advanced_so_far
123         time_advanced_so_far += present_time_step;
124
125         // If using adaptive timestep, then increase the
126         // present_time_step (by 1% for now)
127         if (mpNumericalMethod->HasAdaptiveTimestep())
128         {
129             ///\todo #2087 Make this a settable member variable
130             double timestep_increase = 0.01;
131             present_time_step =
132                 std::min((1+timestep_increase)*present_time_step,
133                         target_time_step - time_advanced_so_far);
134         }
135
136     }
137     catch (StepSizeException& e)
138     {
139         // Detects if a node has travelled too far in a single time
140         // step
141         if (mpNumericalMethod->HasAdaptiveTimestep())

```

```

132     {
133         // If adaptivity is switched on, revert node locations
134         → and choose a suitably smaller time step
135         RevertToOldLocations(old_node_locations);
136         present_time_step = std::min(e.GetSuggestedNewStep(),
137             → target_time_step - time_advanced_so_far);
138     }
139     else
140     {
141         // If adaptivity is switched off, terminate with an error
142         EXCEPTION(e.what());
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)
152 {
153     for (typename AbstractMesh<ELEMENT_DIM, SPACE_DIM>::NodeIterator
154         → node_iter =
155         → this→mrCellPopulation.rGetMesh().GetNodeIteratorBegin();
156         node_iter ≠
157         → this→mrCellPopulation.rGetMesh().GetNodeIteratorEnd();
158         ++node_iter)
159     {
160         (node_iter)→rGetModifiableLocation() =
161             → oldNodeLoctions[&(*node_iter)];
162     }
163 }
164
165 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
166 void
167     → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::ApplyBoundaries(std::map
168         → SPACE_DIM> > oldNodeLoctions)
169 {
170     // Apply any boundary conditions

```

```

163   for (typename
164     ↵ std::vector<boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT>>::iterator bcs_iter = mBoundaryConditions.begin();
165     ↵ bcs_iter != mBoundaryConditions.end();
166     ↵ ++bcs_iter)
167   {
168     (*bcs_iter)->ImposeBoundaryCondition(oldNodeLoctions);
169   }
170
171 // Verify that each boundary condition is now satisfied
172 for (typename
173   ↵ std::vector<boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT>>::iterator bcs_iter = mBoundaryConditions.begin();
174   ↵ bcs_iter != mBoundaryConditions.end();
175   ↵ ++bcs_iter)
176 {
177   if (!(*bcs_iter)->VerifyBoundaryCondition()))
178   {
179     EXCEPTION("The cell population boundary conditions are
180     ↵ incompatible.");
181   }
182 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
183 void
184   ↵ OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::WriteVisualizerSetupFile
185 {
186   if (PetscTools::AmMaster())
187   {
188     for (unsigned i=0; i<this->mForceCollection.size(); i++)
189     {
190       ↵ this->mForceCollection[i]->WriteDataToVisualizerSetupFile(this->mpViz)
191     }
192
193   ↵ this->mrCellPopulation.WriteDataToVisualizerSetupFile(this->mpVizSetupFile)
194 }
195
196 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
```

```

197 void
198 {
199     // Clear all forces
200     for (typename AbstractMesh<ELEMENT_DIM, SPACE_DIM>::NodeIterator
201         node_iter =
202             this->mrCellPopulation.rGetMesh().GetNodeIteratorBegin();
203         node_iter !=
204             this->mrCellPopulation.rGetMesh().GetNodeIteratorEnd();
205         ++node_iter)
206     {
207         node_iter->ClearAppliedForce();
208     }
209
210     // Use a forward Euler method by default, unless a numerical method
211     // has been specified already
212     if (mpNumericalMethod == nullptr)
213     {
214         mpNumericalMethod =
215             boost::make_shared<ForwardEulerNumericalMethod<ELEMENT_DIM,
216             SPACE_DIM>>();
217
218         mpNumericalMethod->SetCellPopulation(dynamic_cast<AbstractOffLatticeCellPopul
219         mpNumericalMethod->SetForceCollection(&mForceCollection);
220     }
221
222     template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
223     void
224     OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::OutputAdditionalSimulati
225     rParamsFile)
226     {
227         // Loop over forces
228         *rParamsFile << "\n\t<Forces>\n";
229         for (typename
230             std::vector<boost::shared_ptr<AbstractForce<ELEMENT_DIM,SPACE_DIM>
231             >>::iterator iter = mForceCollection.begin();
232             iter != mForceCollection.end();
233             ++iter)
234         {
235             // Output force details
236             (*iter)->OutputForceInfo(rParamsFile);
237         }

```

```

228     *rParamsFile << "\t</Forces>\n";
229
230     // Loop over cell population boundary conditions
231     *rParamsFile << "\n\t<CellPopulationBoundaryConditions>\n";
232     for (typename
233         → std::vector<boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT>
234         → >>::iterator iter = mBoundaryConditions.begin();
235         iter ≠ mBoundaryConditions.end();
236         ++iter)
237     {
238         // Output cell boundary condition details
239         (*iter)→OutputCellPopulationBoundaryConditionInfo(rParamsFile);
240     }
241     *rParamsFile << "\t</CellPopulationBoundaryConditions>\n";
242
243     // Output numerical method details
244     *rParamsFile << "\n\t<NumericalMethod>\n";
245     mpNumericalMethod→OutputNumericalMethodInfo(rParamsFile);
246     *rParamsFile << "\t</NumericalMethod>\n";
247 }
248
249 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM>
250 void
251     → OffLatticeSimulationPropertyStop<ELEMENT_DIM,SPACE_DIM>::OutputSimulationParameter
252     → rParamsFile)
253 {
254     // No new parameters to output, so just call method on direct parent
255     → class
256
257     → AbstractCellBasedSimulation<ELEMENT_DIM,SPACE_DIM>::OutputSimulationParameter
258 }
259
260 // Explicit instantiation
261 template class OffLatticeSimulationPropertyStop<1,1>;
262 template class OffLatticeSimulationPropertyStop<1,2>;
263 template class OffLatticeSimulationPropertyStop<2,2>;
264 template class OffLatticeSimulationPropertyStop<1,3>;
265 template class OffLatticeSimulationPropertyStop<2,3>;
266 template class OffLatticeSimulationPropertyStop<3,3>;
267
268 // Serialization for Boost ≥ 1.36
269 #include "SerializationExportWrapperForCpp.hpp"
270 EXPORT_TEMPLATE_CLASS_ALL_DIMS(OffLatticeSimulationPropertyStop)

```

---

### 16.2.31 /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.

```

```
34  */
35 template<unsigned ELEMENT_DIM, unsigned SPACE_DIM = ELEMENT_DIM>
36 class OffLatticeSimulationPropertyStop : public
37     AbstractCellBasedSimulation<ELEMENT_DIM, SPACE_DIM>
38 {
39
40     /** Needed for serialization. */
41     friend class boost::serialization::access;
42     friend class TestOffLatticeSimulation;
43     friend class TestOffLatticeSimulationWithNodeBasedCellPopulation;
44
45 /**
46 * Save or restore the simulation.
47 *
48 * @param archive the archive
49 * @param version the current version of this class
50 */
51 template<class Archive>
52 void serialize(Archive & archive, const unsigned int version)
53 {
54     archive &
55         boost::serialization::base_object<AbstractCellBasedSimulation<ELEMENT_DIM,
56                                         SPACE_DIM>>(*this);
57     archive & mForceCollection;
58     archive & mBoundaryConditions;
59     archive & mpNumericalMethod;
60 }
61
62 protected:
63     boost::shared_ptr<AbstractCellProperty> p_property;
64     /** The mechanics used to determine the new location of the cells, a
65      * list of the forces. */
66     std::vector<boost::shared_ptr<AbstractForce<ELEMENT_DIM, SPACE_DIM>>
67         > mForceCollection;
68
69     /** List of boundary conditions. */
70
71     std::vector<boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT_DIM,
72                                         SPACE_DIM>>> mBoundaryConditions;
73
74     /** The numerical method to use in this simulation. Defaults to the
75      * explicit forward Euler method. */
76 }
```

```
69     boost::shared_ptr<AbstractNumericalMethod<ELEMENT_DIM, SPACE_DIM>>
70     mpNumericalMethod;
71
72     /**
73      * Overridden UpdateCellLocationsAndTopology() method.
74      *
75      * Calculate forces and update node positions.
76      */
77
78     virtual void UpdateCellLocationsAndTopology();
79
80     /**
81      * Sends nodes back to the positions given in the input map. Used
82      * after a failed step
83      * when adaptivity is turned on.
84      *
85      * @param oldNodeLoctions A map linking nodes to their old positions.
86      */
87     void RevertToOldLocations(std::map<Node<SPACE_DIM>*, c_vector<double,
88                               > oldNodeLoctions);
89
90     /**
91      * Applies any boundary conditions.
92      *
93      * @param oldNodeLoctions Mapping between node indices and old node
94      * locations
95      */
96     void ApplyBoundaries(std::map<Node<SPACE_DIM>*, c_vector<double,
97                           > oldNodeLoctions);
98
99     /**
100      * Overridden SetupSolve() method to clear the forces applied to the
101      * nodes.
102      */
103
104     virtual void SetupSolve();
105
106     /**
107      * Overridden WriteVisualizerSetupFile() method.
108      */
109
110     virtual void WriteVisualizerSetupFile();
111
112     bool StoppingEventHasOccurred();
113
114 public:
```

```

106
107  /**
108   * Constructor.
109   *
110  * @param rCellPopulation Reference to a cell population object
111  * @param deleteCellPopulationInDestructor Whether to delete the cell
112  * population on destruction to
113  *     * free up memory (defaults to false)
114  * @param initialiseCells Whether to initialise cells (defaults to
115  * true, set to false when loading
116  *     * from an archive)
117  */
118 OffLatticeSimulationPropertyStop(AbstractCellPopulation<ELEMENT_DIM,
119  * SPACE_DIM>& rCellPopulation,
120  * bool
121  *     * deleteCellPopulationInDestructor
122  *     * = false, bool initialiseCells =
123  *     * true);
124
125
126
127
128 void SetStopProperty(boost::shared_ptr<AbstractCellProperty>
129  * stopPropertySetting);
130
131
132
133
134
135
136
137
138

```

```
139     */
140 void AddCellPopulationBoundaryCondition(
141     → boost::shared_ptr<AbstractCellPopulationBoundaryCondition<ELEMENT_DIM,
142     → SPACE_DIM> > pBoundaryCondition);
143 /**
144 * Method to remove all the cell population boundary conditions
145 */
146 void RemoveAllCellPopulationBoundaryConditions();
147 /**
148 * Set the numerical method to be used in this simulation (use this
149 → to solve the mechanics system).
150 *
151 * @param pNumericalMethod pointer to a numerical method object
152 */
153 void
154     → SetNumericalMethod(boost::shared_ptr<AbstractNumericalMethod<ELEMENT_DIM,
155     → SPACE_DIM> > pNumericalMethod);
156 /**
157 * @return the current numerical method.
158 */
159 const boost::shared_ptr<AbstractNumericalMethod<ELEMENT_DIM,
160     → SPACE_DIM> > GetNumericalMethod() const;
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 * Overridden OutputSimulationParameters() method.
173 *
174 * @param rParamsFile the file stream to which the parameters are
175 → output
```

```

173     */
174     virtual void OutputSimulationParameters(out_stream& rParamsFile);
175
176     /**
177      * Directly access the forces attached to this simulation, to allow
178      * their manipulation after archiving.
179      *
180      * @return mForceCollection the vector of pointers to forces attached
181      * to this simulation
182      */
183     const std::vector<boost::shared_ptr<AbstractForce<ELEMENT_DIM,
184     >>& rGetForceCollection() const;
185
186 // Serialization for Boost ≥ 1.36
187 #include "SerializationExportWrapper.hpp"
188 EXPORT_TEMPLATE_CLASS_ALL_DIMS(OffLatticeSimulationPropertyStop)
189
190 namespace boost
191 {
192 namespace serialization
193 {
194 /**
195  * Serialize information required to construct an OffLatticeSimulation.
196 */
197 template<class Archive, unsigned ELEMENT_DIM, unsigned SPACE_DIM>
198 inline void save_construct_data(Archive & ar, const
199   OffLatticeSimulationPropertyStop<ELEMENT_DIM, SPACE_DIM> * t,
200                           const unsigned int file_version)
201 {
202     // Save data required to construct instance
203     const AbstractCellPopulation<ELEMENT_DIM, SPACE_DIM>*
204       p_cell_population = &(t->rGetCellPopulation());
205     ar & p_cell_population;
206 }
207
208 /**
209  * De-serialize constructor parameters and initialise an
210  * OffLatticeSimulation.
211 */
212 template<class Archive, unsigned ELEMENT_DIM, unsigned SPACE_DIM>
213 inline void load_construct_data(Archive & ar,
214   OffLatticeSimulationPropertyStop<ELEMENT_DIM, SPACE_DIM> * t,

```

```

209                         const unsigned int file_version)
210 {
211     // Retrieve data from archive required to construct new instance
212     AbstractCellPopulation<ELEMENT_DIM, SPACE_DIM>* p_cell_population;
213     ar >> p_cell_population;
214
215     // Invoke inplace constructor to initialise instance, middle two
216     // variables set extra
217     // member variables to be deleted as they are loaded from archive and
218     // to not initialise cells.
219     ::new (t) OffLatticeSimulationPropertyStop<ELEMENT_DIM,
220           SPACE_DIM>(*p_cell_population, true, false);
221 }
222 }
223 } // namespace
224
225 #endif /*OFFLATTICESIMULATIONPROPERTYSTOP_HPP_*/

```

---

### 16.2.32 /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

```

```

13         rModel.mPopulation), mOutput(rModel.mOutput),
14         ↳ mEventStartTime(rModel.mEventStartTime), mDebug(
15             rModel.mDebug), mTimeID(rModel.mTimeID),
16             ↳ mVarIDs(rModel.mVarIDs),
17             ↳ mDebugWriter(rModel.mDebugWriter), mBasePopulation(
18                 rModel.mBasePopulation), mGammaShift(rModel.mGammaShift),
19                 ↳ mGammaShape(rModel.mGammaShape), mGammaScale(
20                     rModel.mGammaScale), mMitoticMode(rModel.mMitoticMode),
21                     ↳ mSeed(rModel.mSeed), mTimeDependentCycleDuration(
22                         rModel.mTimeDependentCycleDuration),
23                         ↳ mPeakRateTime(rModel.mPeakRateTime),
24                         ↳ mIncreasingRateSlope(
25                             rModel.mIncreasingRateSlope),
26                             ↳ mDecreasingRateSlope(rModel.mDecreasingRateSlope),
27                             ↳ mBaseGammaScale(
28                                 rModel.mBaseGammaScale),
29                                 ↳ mHeParamVector(rModel.mHeParamVector)
30     {
31 }
32
33 AbstractCellCycleModel* WanStemCellCycleModel::CreateCellCycleModel()
34 {
35     return new WanStemCellCycleModel(*this);
36 }
37
38 void WanStemCellCycleModel::SetCellCycleDuration()
39 {
40     RandomNumberGenerator* p_random_number_generator =
41         ↳ RandomNumberGenerator::Instance();
42
43     mCellCycleDuration = mGammaShift +
44         ↳ p_random_number_generator->GammaRandomDeviate(mGammaShape,
45             ↳ mGammaScale);
46
47     /*****
48     * CELL CYCLE DURATION RANDOM VARIABLE
49     *****/
50
51 //Wan stem cell cycle length determined by the same formula as He
52     ↳ RPCs
53 /*
54     if (!mTimeDependentCycleDuration) //Normal operation, cell cycle
55         ↳ length stays constant
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
85
86
87
88
89
90

```

```

41     //He cell cycle length determined by shifted gamma distribution
42     → reflecting 4 hr refractory period followed by gamma pdf
43     mCellCycleDuration = mGammaShift +
44     → p_random_number_generator→GammaRandomDeviate(mGammaShape,
45     → mGammaScale);
46     }*/
47
48 /**
49 * Variable cycle length
50 * Give -ve mIncreasingRateSlope and +ve mDecreasingRateSlope,
51 * cell cycle length linearly declines (increasing rate), then
52 * increases, switching at mPeakRateTime
53 */
54
55 /*
56 else
57 {
58 double currTime = SimulationTime::Instance()→GetTime();
59 if (currTime ≤ mPeakRateTime)
60 {
61     mGammaScale = std::max((mBaseGammaScale - currTime *
62     → mIncreasingRateSlope), .000000000001);
63     if (currTime > mPeakRateTime)
64     {
65         mGammaScale = std::max(((mBaseGammaScale - mPeakRateTime *
66     → mIncreasingRateSlope)
67     + (mBaseGammaScale + (currTime - mPeakRateTime) *
68     → mDecreasingRateSlope)),.000000000001);
69     }
70     Timer::Print("mGammaScale: " + std::to_string(mGammaScale));
71     mCellCycleDuration = mGammaShift +
72     → p_random_number_generator→GammaRandomDeviate(mGammaShape,
73     → mGammaScale);
74     */
75 }
76 */
77 }

78 void WanStemCellCycleModel::ResetForDivision()
79 {
80
81     mMitoticMode = 1; //by default, asymmetric division giving rise to He
82     → cell (mode 1)
83

```

```

74     if (mExpandingStemPopulation)
75     {
76         double currRetinaAge = SimulationTime::Instance()→GetTime() +
77             → mEventStartTime;
78         double lensGrowthFactor = .09256 * pow(currRetinaAge, .52728); // //
79         unsigned currentPopulationTarget = int(std::round(mBasePopulation
80             → * lensGrowthFactor));
81
82         unsigned currentStemPopulation =
83             (mPopulation→GetCellProliferativeTypeCount())[0];
84         if (currentStemPopulation < currentPopulationTarget)
85         {
86             mMitoticMode = 0; //if the current population is < target,
87             → symmetrical stem-stem division occurs (mode 0)
88         }
89     }
90
91     /*****
92     * Write mitotic event to relevant files
93     * *****/
94     if (mDebug)
95     {
96         WriteDebugData();
97     }
98
99     if (mOutput)
100    {
101        WriteModeEventOutput();
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     SetCellCycleDuration();
112 }
113
114 void WanStemCellCycleModel::InitialiseDaughterCell()
115 {
116
117     if (mMitoticMode == 1)
118     {
119         /*****  

120          * RPC-fated cells are given HeCellCycleModel  

121          *****/  

122
123     double tiLOffset = -(SimulationTime::Instance()→GetTime());
124     //Initialise a HeCellCycleModel and set it up with appropriate
125     // TiL value & parameters
126     HeCellCycleModel* p_cycle_model = new HeCellCycleModel;
127     p_cycle_model→SetModelParameters(tiLOffset, mHeParamVector[0],
128                                     mHeParamVector[1], mHeParamVector[2],
129                                     mHeParamVector[3],
130                                     mHeParamVector[4],
131                                     mHeParamVector[5],
132                                     mHeParamVector[6],
133                                     mHeParamVector[7],
134                                     mHeParamVector[8],
135                                     mHeParamVector[9],
136                                     mHeParamVector[10],
137                                     mHeParamVector[11]);
138     p_cycle_model→EnableKillSpecified();
139
140     //if debug output is enabled for the stem cell, enable it for its
141     // progenitor offspring
142     if (mDebug)
143     {
144         p_cycle_model→PassDebugWriter(mDebugWriter, mTimeID,
145                                     mVarIDs);
146     }
147
148     mpCell→SetCellCycleModel(p_cycle_model);
149     p_cycle_model→Initialise();
150 }
151
152 else
153 {

```

```

144         SetCellCycleDuration();
145     }
146 }
147
148 void WanStemCellCycleModel::SetModelParameters(double gammaShift, double
149     ↪ gammaShape, double gammaScale,
150                     std::vector<double>
151     ↪ heParamVector)
152 {
153     mGammaShift = gammaShift;
154     mGammaShape = gammaShape;
155     mGammaScale = gammaScale;
156     mHeParamVector = heParamVector;
157 }
158
159 void WanStemCellCycleModel::EnableExpandingStemPopulation(int
160     ↪ basePopulation,
161                     boost::shared_ptr<AbstractPopulation>
162     ↪ p_population)
163 {
164     mExpandingStemPopulation = true;
165     mBasePopulation = basePopulation;
166     mPopulation = p_population;
167 }
168
169 void WanStemCellCycleModel::SetTimeDependentCycleDuration(double
170     ↪ peakRateTime, double increasingSlope,
171                     double
172     ↪ decreasingSlope)
173 {
174     mTimeDependentCycleDuration = true;
175     mPeakRateTime = peakRateTime;
176     mIncreasingRateSlope = increasingSlope;
177     mDecreasingRateSlope = decreasingSlope;
178     mBaseGammaScale = mGammaScale;
179 }
180
181 void WanStemCellCycleModel::EnableModeEventOutput(double eventStart,
182     ↪ unsigned seed)
183 {
184     mOutput = true;
185     mEventStartTime = eventStart;

```

```

179     mSeed = seed;
180 }
181
182 void WanStemCellCycleModel::WriteModeEventOutput()
183 {
184     double currentTime = SimulationTime::Instance()→GetTime() +
185         → mEventStartTime;
186     CellPtr currentCell = GetCell();
187     double currentCellID = (double) currentCell→GetCellId();
188     (*LogFile::Instance()) << currentTime << "\t" << mSeed << "\t" <<
189         → currentCellID << "\t" << mMitoticMode << "\n"; //
190 }
191
192 void
193     → WanStemCellCycleModel::EnableModelDebugOutput(boost :: shared_ptr<ColumnDataWriter>
194         → debugWriter)
195 {
196     mDebug = true;
197     mDebugWriter = debugWriter;
198
199     mTimeID = mDebugWriter→DefineUnlimitedDimension("Time", "h");
200     mVarIDs.push_back(mDebugWriter→DefineVariable("CellID", "No"));
201     mVarIDs.push_back(mDebugWriter→DefineVariable("TIL", "h"));
202     mVarIDs.push_back(mDebugWriter→DefineVariable("CycleDuration",
203         → "h"));
204     mVarIDs.push_back(mDebugWriter→DefineVariable("Phase2Boundary",
205         → "h"));
206     mVarIDs.push_back(mDebugWriter→DefineVariable("Phase3Boundary",
207         → "h"));
208     mVarIDs.push_back(mDebugWriter→DefineVariable("Phase", "No"));
209     mVarIDs.push_back(mDebugWriter→DefineVariable("MitoticModeRV",
210         → "Percentile"));
211     mVarIDs.push_back(mDebugWriter→DefineVariable("MitoticMode",
212         → "Mode"));
213     mVarIDs.push_back(mDebugWriter→DefineVariable("Label", "binary"));
214
215     mDebugWriter→EndDefineMode();
216 }
217
218
219 void WanStemCellCycleModel::WriteDebugData()
220 {
221     double currentTime = SimulationTime::Instance()→GetTime();
222     double currentCellID = mpCell→GetCellId();

```

```
213     mDebugWriter→PutVariable(mTimeID, currentTime);
214     mDebugWriter→PutVariable(mVarIDs[0], currentCellID);
215     mDebugWriter→PutVariable(mVarIDs[1], 0);
216     mDebugWriter→PutVariable(mVarIDs[2], mCellCycleDuration);
217     mDebugWriter→PutVariable(mVarIDs[3], 0);
218     mDebugWriter→PutVariable(mVarIDs[4], 0);
219     mDebugWriter→PutVariable(mVarIDs[5], 0);
220     mDebugWriter→PutVariable(mVarIDs[7], mMitoticMode);
221     mDebugWriter→AdvanceAlongUnlimitedDimension();
222 }
223
224 /**
225  * UNUSED FUNCTIONS (required for class nonvirtuality, do not remove)
226  */
227 ****
228
229 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)
```

### 16.2.33 /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  *
25  * Change default model parameters with SetModelParameters(<params>);
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  * 1 mitotic-event-sequence sampler (only samples one "path" through the
36  *      ↳ lineage):
37  * EnableSequenceSampler() - one "sequence" of progenitors writes mitotic
38  *      ↳ event type to a string in the singleton log file
39  *
40 *****/

```

```
37 class WanStemCellCycleModel : public AbstractSimpleCellCycleModel
38 {
39     friend class TestSimpleCellCycleModels;
40
41 private:
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     {
55         archive &
56             → boost::serialization::base_object<AbstractSimpleCellCycleModel>(*this);
57
58         SerializableSingleton<RandomNumberGenerator>* p_wrapper =
59
60             → RandomNumberGenerator::Instance()→GetSerializationWrapper();
61
62         archive & p_wrapper;
63         archive & mCellCycleDuration;
64     }
65
66     //Private write functions for models
67     void WriteModeEventOutput();
68     void WriteDebugData();
69
70 protected:
71     //mode/output variables
72     bool mExpandingStemPopulation;
73     boost::shared_ptr<AbstractCellPopulation<2>> mPopulation;
74     bool mOutput;
75     double mEventStartTime;
76     //debug writer stuff
77     bool mDebug;
78     int mTimeID;
79     std::vector<int> mVarIDs;
80     boost::shared_ptr<ColumnDataWriter> mDebugWriter;
```

```

77     //model parameters and state memory vars
78     int mBasePopulation;
79     double mGammaShift;
80     double mGammaShape;
81     double mGammaScale;
82     unsigned mMitoticMode;
83     unsigned mSeed;
84     bool mTimeDependentCycleDuration;
85     double mPeakRateTime;
86     double mIncreasingRateSlope;
87     double mDecreasingRateSlope;
88     double mBaseGammaScale;
89     std::vector<double> mHeParamVector;

90
91 /**
92  * Protected copy-constructor for use by CreateCellCycleModel().
93  *
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  */

```

```
112     WanStemCellCycleModel();  
113  
114     /**  
115      * SetCellCycleDuration() method to set length of cell cycle  
116      */  
117     void SetCellCycleDuration();  
118  
119     /**pro  
120      * Overridden builder method to create new copies of  
121      * this cell-cycle model.  
122      *  
123      * @return new cell-cycle model  
124      */  
125     AbstractCellCycleModel* CreateCellCycleModel();  
126  
127     /**  
128      * 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      */
```

```
151 void SetModelParameters(double gammaShift = 4, double gammaShape = 2,
152     ↵ double gammaScale = 1, std::vector<double> heParamVector = { 8,
153     ↵ 15, 1, 0, .2, .4, .2, 0, 4, 2, 1, 1 });
154 void EnableExpandingStemPopulation(int basePopulation,
155     ↵ boost::shared_ptr<AbstractCellPopulation<2>> p_population);
156 void SetTimeDependentCycleDuration(double peakRateTime, double
157     ↵ increasingSlope, double decreasingSlope);
158
159 //Functions to enable per-cell mitotic mode logging for mode rate &
160     ↵ sequence sampling fixtures
161 //Uses singleton logfile
162 void EnableModeEventOutput(double eventStart, unsigned seed);
163
164 //More detailed debug output. Needs a ColumnDataWriter passed to it
165 //Only declare ColumnDataWriter directory, filename, etc; do not set
166     ↵ up otherwise
167 void EnableModelDebugOutput(boost::shared_ptr<ColumnDataWriter>
168     ↵ debugWriter);
169
170 //Not used, but must be overwritten lest WanStemCellCycleModels be
171     ↵ abstract
172 double GetAverageTransitCellCycleTime();
173 double GetAverageStemCellCycleTime();
174
175 /**
176 * Overridden OutputCellCycleModelParameters() method.
177 *
178 * @param rParamsFile the file stream to which the parameters are
179     ↵ output
180     */
181 virtual void OutputCellCycleModelParameters(out_stream& rParamsFile);
182 };
183
184 #include "SerializationExportWrapper.hpp"
185 // Declare identifier for the serializer
186 CHASTE_CLASS_EXPORT(WanStemCellCycleModel)
187
188 #endif /*WANSTEMCELLCYCLEMODEL_HPP_*/
```

## 16.3 NGRefTools

### 16.3.1 /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

```

---

### 16.3.2 /src/MarginalTDist.jl

---

```

1 """
2     MarginalTDist(v,μ,σ)
3
4     Return a shifted, scaled T Distribution with degrees of freedom `v`, mean
5     ↪ `μ`, and standard deviation `σ`.
6
7     This is the marginal distribution of the posterior mean for an
8     ↪ uninformative (reference) Normal Gamma prior distribution. It is also
9     ↪ the distribution of the posterior predictive for m=1 new
10    ↪ observations.
11
12    MarginalTDist(v)           #Standard TDist with v dof
13    MarginalTDist(v,μ,σ)       #TDist with v dof shifted by μ, scaled by σ
14
15    params(d)                 #Return d.v, d.μ, d.σ
16    mean(d)                   #Return d.μ (median, mode are identical)
17    std(d)                    #Return d.σ
18    quantile(d,p)            #Return quantile at probability p
19
20
21    Reference: Kevin P. Murphy, Conjugate Bayesian Analysis of the Gaussian
22    ↪ Distribution. 2007.
23    ↪ https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf
24 """
25
26 MarginalTDist=LocationScale{Float64,TDist{Float64}}

```

```

19
20 Distributions.dof(d::MarginalTDist) = d.ρ.ν
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
28     ↳ marginal posterior distribution of the mean of a Normal model of
29     ↳ observations `x`.
30
31 Equivalent to the frequentist sampling distribution of the MLE mean. If
32     ↳ `PP=true`, return the posterior predictive distribution for m=1
33     ↳ additional observations sampled from the marginal posterior mean
34     ↳ instead.
35
36 Example:
37
38 julia> PMM_MarginalTDist(rand(10))
39 MarginalTDist{Float64}(
40     t: TDist{Float64}(v=9.0)
41     μ: 0.4789484996068401
42     σ: 0.08491142725619677
43     )
44
45 Reference: Kevin P. Murphy, Conjugate Bayesian Analysis of the Gaussian
46     ↳ Distribution. 2007.
47     ↳ https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf
48 """
49
50 function Distributions.fit(::Type{MarginalTDist},
51     ↳ x::AbstractVector{<:Real}; PP=false)
52     PP ? PPM_MTDist(x) : PMM_MTDist(x)
53 end
54
55 #Posterior marginal mean distribution
56 function PMM_MTDist(x)
57     n=length(x)
58     μ=mean(x)
59     ssr=sum((x.-μ).^2)
60     σ=sqrt(ssr/(n*(n-1)))
61     return MarginalTDist(μ,σ,TDist(n-1))
62 end
63

```

```

54
55 #Posterior predictive mean distribution
56 function PPM_MTDist(x)
57     n=length(x)
58     μ=mean(x)
59     ssr=sum((x.-μ).^2)
60     αn=(n-1)/2
61     βn=.5*ssr
62     return MarginalTDist(μ,sqrt((βn*(n+1))/(αn*n)),TDist(2*αn))
63 end
64
65 """
66 MTDist_MC_func(func, xs ... ; lower=.025, upper=.975, mc_its=1e6,
67   → summary=false)
68 Monte Carlo execute `func` with a random sample from a MarginalTDist
69   → fitted to each x in xs, over `mc_its` iterates, returning the
70   → calculated results (if `summary` is `true`) or the mean and quantiles
71   → specified by `lower` and `upper`.
72 Example:
73 julia> NGRefTools.MTDist_MC_func(*,[rand(10),rand(10)],summary=true)
74 (0.09491370397229557, 0.20712818357976473, 0.3385677491994645)
75 See also: [`fit(MarginalTDist,x)`](@ref)
76 """
77 function MTDist_MC_func(func::Function, xs; lower=.025, upper=.975,
78   → mc_its=1e7, summary=false)
79     dists=[fit(MarginalTDist,x) for x in xs]
80     results=Vector{Float64}()
81     for it in 1:mc_its
82       push!(results, func(rand.(dists)...))
83     end
84     summary ? (return quantile(results, lower), mean(results),
85       → quantile(results,upper)) : (return results)
86 end
87
88 function plot_logn_MTDist(xs, colors, markers, labels, xlabel, ylabel;
89   → args...)
90   plt=plot(;args...)
91   for (x,color) in zip(xs,colors)
92     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
95             → plot!(plt,X,y,ribbon=(y,zeros(length(y))),label=:none,color=color,xlabel=
96         end
97         for (x,color,marker,label) in zip(xs,colors, markers,labels)
98             pmdist=fit(MarginalTDist, log.(x))
99             min=floor(minimum(x)-.15*mean(x))
100            max=ceil(maximum(x)+.20*mean(x))
101            X=collect(min:max)
102            y=pdf(pmdist,log.(X))
103            scaty=[mean(y) for x in 1:length(x)]
104            plt=scatter!(plt,x,scaty,color=color, marker=marker, label=label)
105        end
106
107    return plt
108 end
109
110 function plot_n_MTDist(xs, colors, markers, labels, xlabel, ylabel;
111     ← args ... )
112     plt=plot(;args ... )
113     for (x,color) in zip(xs,colors)
114         min=(minimum(x)-.01*mean(x))
115         max=(maximum(x)+.01*mean(x))
116         X=collect(min:(max-min)/1000:max)
117         pmdist=fit(MarginalTDist, x)
118         y=pdf(pmdist,X)
119
120             → plot!(plt,X,y,ribbon=(y,zeros(length(y))),color=color,label=:none,xlabel=
121         end
122         for (x,color,marker,label) in zip(xs,colors, markers,labels)
123             pmdist=fit(MarginalTDist, x)
124             min=(minimum(x)-.01*mean(x))
125             max=(maximum(x)+.01*mean(x))
126             X=collect(min:max)
127             y=pdf(pmdist,X)
128             scaty=[mean(y) for x in 1:length(x)]
129             plt=scatter!(plt,x,scaty,color=color, marker=marker,label=label)
130         end
131
132     return plt

```

```

130 end

131
132 function mean_mass_comparator(x,y;labels=[ "x" , "y" ])
133     xdist=fit(MarginalTDist,x)
134     ydist=fit(MarginalTDist,y)
135     if mean(xdist)>mean(ydist)
136         frac=ccdf(xdist,mean(ydist))
137         println("$(round(frac*100,digits=1))% of the marginal posterior
138             ↪ mean density of $(labels[1]) is above the mean of
139             ↪ $(labels[2]).")
140     else
141         frac=cdf(xdist,mean(ydist))
142         println("$(round(frac*100,digits=1))% of the marginal posterior
143             ↪ mean density of $(labels[1]) is below the mean of
144             ↪ $(labels[2]).")
145     end
146 end

147
148 function log_mean_mass_comparator(x,y;labels=[ "x" , "y" ])
149     xdist=fit(MarginalTDist,log.(filter(n→!iszero(n),x)))
150     ydist=fit(MarginalTDist,log.(filter(n→!iszero(n),y)))
151     if mean(xdist)>mean(ydist)
152         frac=ccdf(xdist,mean(ydist))
153         println("$(round(frac*100,digits=1))% of the marginal posterior
154             ↪ mean density of $(labels[1]) is above the mean of
155             ↪ $(labels[2]).")
156     else
157         frac=cdf(xdist,mean(ydist))
158         println("$(round(frac*100,digits=1))% of the marginal posterior
159             ↪ mean density of $(labels[1]) is below the mean of
160             ↪ $(labels[2]).")
161     end
162 end

```

---

### 16.3.3 /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,
10    ↪ rate=0.5057485400844524)
11 Reference: Kevin P. Murphy, Conjugate Bayesian Analysis of the Gaussian
12    ↪ Distribution. 2007.
13    ↪ https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf
14    """
15
16        function Distributions.fit(::Type{NormalGamma},x::AbstractVector)
17            n=length(x)
18            μ=mean(x)
19            ssr=sum((x.-μ).^2)
20            α=(n-1)/2
21            β=ssr/2
22            return NormalGamma(μ,n,α,β)
23        end
24
25
26        function Distributions.params(d::NormalGamma)
27            return d.mu, d.nu, d.shape, d.rate
28        end
29
30        function marginals(d::NormalGamma)
31            ↪ m_T=MarginalTDist(d.mu,sqrt(d.rate/(d.shape*d.nu)),TDist(2*d.shape))
32            m_Ga=Gamma(d.shape,1/d.rate)
33            return m_T,m_Ga
34        end

```

---

### 16.3.4 /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
11   include("LogNormalUtils.jl")
12   export get_lognormal_params, get_lognormal_desc
13 end # module

```

---

### 16.3.5 /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))
11 NormalInverseGamma{Float64}(mu=0.5052725306750604, nu=10.0,
12   ↪ shape=4.5, rate=0.5057485400844524)
13
14 Reference: Kevin P. Murphy, Conjugate Bayesian Analysis of the Gaussian
15   ↪ Distribution. 2007.
16   ↪ https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf
17 """
18
19 function
20   ↪ Distributions.fit(::Type{NormalInverseGamma}, x::AbstractVector)
21   n=length(x)
22   Vn = 1/n
23   μ=mean(x)
24   an = -.5 + (n/2)
25   bn = .5 * (sum(x.^2)-(μ^2/Vn))
26   return NormalInverseGamma(μ,Vn,an,bn)
27 end
28
29 function Distributions.params(d::NormalInverseGamma)
30     return d.mu, d.v0, d.shape, d.scale
31 end
32
33 function marginals(d::NormalInverseGamma)
34   ↪ m_T=MarginalTDist(d.mu,sqrt(d.shape*d.v0/d.scale),TDist(2*d.shape))

```

```

28     m_Ga=InverseGamma(d.shape,d.scale)
29     return m_T,m_Ga
30 end
31
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 #
43 #   return surface(xs,ys,zs)
# end

```

---

## 16.4 GMC\_NS

### 16.4.1 /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
12    import Base: show
13    import Measurements: measurement
14    import ConjugatePriors: NormalGamma, NormalInverseGamma
15    import NGRefTools: MarginalTDist
16    import KernelDensityEstimate: kde!

```

```

17  #[GMC_Nmin::Int64, GMC_τ_death::Float64, GMC_init_τ::Float64,
18  ↵  GMC_tune_μ::Int64, GMC_tune_α::Float64,
19  ↵  GMC_tune_PID::NTuple{3,Float64}, GMC_timestep_η::Float64,
20  ↵  GMC_reflect_η::Float64, GMC_exhaust_σ::Float64]
21
22  GMC_DEFAULTS=Vector{Any}([50,1e-6,.5, 50,.8,(.175,1e-5,.1), .001,
23  ↵  .01, 10.])
24  export GMC_DEFAULTS
25
26  include("ensemble/GMC_NS_Engine.jl")
27  export GMC_NS_Engine
28  include("GMC_NS_Model.jl")
29  export GMC_NS_Model, GMC_NS_Model_Record
30  include("GMC/galilean_trajectory.jl")
31  include("utilities/t_Tuner.jl")
32  include("utilities/ellipsoid.jl")
33  include("nested_sampler/search_patterns.jl")
34  include("nested_sampler/nested_step.jl")
35  include("nested_sampler/converge_ensemble.jl")
36  export converge_ensemble!
37  include("utilities/coordinate_utils.jl")
38  export box_bound!, to_prior, to_unit_ball, box_reflect!
39  include("utilities/ensemble_utilities.jl")
40  export ensemble_history, clean_ensemble_dir, measure_evidence,
41  ↵  reset_ensemble!, move_ensemble!, copy_ensemble!, rewind_ensemble,
42  ↵  show_models, show_models_e, get_model, rectify_ensemble!,
43  ↵  posterior_kde
44  include("utilities/ns_progressmeter.jl")
45  include("utilities/progress_displays.jl")
46  export
47    ↵  tuning_display,evidence_display,convergence_display,info_display,lh_display,l
48  include("utilities/stats.jl")
49  include("normal/Normal_Model.jl")
50  include("normal/Normal_Engine.jl")
51  export Normal_Engine
52  include("normal/LogNormal_Model.jl")
53  include("normal/LogNormal_Engine.jl")
54  export LogNormal_Engine
55
56 end # module

```

---

## 16.4.2 /src/GMC\_NS\_Model.jl

```

1      """
2      GMC_NS_Model_Record
3
4  Abstract supertype for records of serialised 'GMC_NS_Model's. Subtyped
5  record structs must implement the following fields as their interface
6  with 'GMC_NS':
7
8      struct Minimal_Record
9          trajectory::Integer #id assigned at construction, identifies
10         trajectory
11         i::Integer #id integer assigned at construction, identifies
12         position on trajectory
13         pos::Vector{<:AbstractFloat} #hypersphere position of
14         model-particle
15         path::String #model is serialized to ensembledir/trajectory.i
16         log_Li::AbstractFloat #log likelihood calculated by
17         'GMC_NS_Model' constructor
18         end
19
20      """
21
22      abstract type GMC_NS_Model_Record end
23
24
25      """
26      GMC_NS_Model_Record
27
28      Abstract supertype for models in 'GMC_NS_Ellement's. Subtyped model
29      structs must implement the following fields as their interface with
30      'GMC_NS':
31
32      mutable struct Minimal_Model
33          trajectory::Integer #id assigned at construction, identifies
34          trajectory
35          i::Integer #id integer assigned at construction, identifies
36          position on trajectory
37
38          θ::Vector{<:AbstractFloat} #model's parameter Vector
39          log_Li<:AbstractFloat #model's likelihood, calculated by
40          constructor
41
42          pos :: Vector{<:AbstractFloat}
43          v :: Vector{<:AbstractFloat} #model's velocity in hypersphere
44          coords

```

```
30      end
31      """
32 abstract type GMC_NS_Model end
```

#### 16.4.3 /src/GMC/galilean\_trajectory.jl

```

1      """
2      galilean_trajectory_sample(e, m, tau)
3
4  Given 'm <: GMC_NS_Model', 'e <: GMC_NS_Ensemble', and time step  $\tau$ ,
5  attempt to find new model obeying 'new_m.log_Li > e.contour', first
6  by permuting ' $m.\theta$ ' by the distance given by velocity ' $m.v$ ' and time
7  step ' $\tau$ ', with ' $m$ ' retaining particle velocity  $v$ :
8
9  "[ $\theta, v$ ] → [ $\theta = \theta + \tau v$ ,  $v$ ]"
10 If 'e.GMC_timestep_eta' is a positive value,  $\tau$  has a random error term of
11    $\text{Normal}(0, \tau * \eta)$  applied.
12
13 If this fails, the rejected particle "reflects" off the contour boundary
14   as represented by the gradient normal ' $n$ ':
15
16 "[ $\theta, v$ ] → [ $\theta = \theta + \tau v + \tau v \cdot n$ ,  $v = v - 2n(n \cdot v)$ ]"
17
18 If 'e.GMC_reflect_eta' is a positive value, this value ( $\eta$  below) is used
19   with a newly sampled isotropic vector to calculate a perturbed
20   reflection vector ' $v_{\perp}$ ':
21
22 """
23 function galilean_trajectory_sample!(m, e, tuner)
24     t=m.trajectory;i=m.i+1
25

```

```

26   e.GMC_timestep_η > 0. ?
27   ↵ (τ=max(tuner.τ+rand(Normal(0,tuner.τ*e.GMC_timestep_η)),eps())) :
28   ↵ (τ=tuner.τ) #perturb τ if specified
29 d=τ*m.v #distance vector for given timestep
30 box_rflct=false
31 fwd_pos,adj_d=box_move(m.pos,d,e.box)
32 isapprox(fwd_pos,m.pos)&&(box_rflct=true)

33
34 !box_rflct ? (new_m=e.model_initλ(t, i, to_prior.(fwd_pos,e.priors),
35   ↵ fwd_pos, m.v, e.obs, e.constants ... )) : (new_m=m) #try to proceed
36   ↵ along distance vector

37
38 if box_rflct || (new_m.log_Li < m.log_Li) #if that didnt work
39   process_report!(tuner, false)

40 lhδ=lps(m.log_Li,-new_m.log_Li) #get the lhδ between the two
41   ↵ points
42 n=boundary_norm(adj_d,lhδ) #get the gradient normal for the
43   ↵ distance and lhδ
44 box_rflct ? (v[] = box_reflect(m.pos,e.box,m.v)) :
45   ↵ v[] = reflect(m.v,n,e.GMC_reflect_η) #get the reflected velocity
46   ↵ vector off the boundary "east"
47 rd=τ*v[] #reflected distance given the timestep
48 east_pos,_=box_move(m.pos,rd,e.box)
49 new_m=e.model_initλ(t, i, to_prior.(east_pos,e.priors), east_pos,
50   ↵ v[], e.obs, e.constants ... ) #try going east
51 if new_m.log_Li < m.log_Li && !box_rflct
52   process_report!(tuner, false)

53 west_pos,_=box_move(m.pos,-rd,e.box)
54 new_m=e.model_initλ(t, i, to_prior.(west_pos,e.priors),
55   ↵ west_pos, -v[], e.obs, e.constants ... ) #if east
56   ↵ reflection fails, try west
57 end
58 if new_m.log_Li < m.log_Li
59   process_report!(tuner, false)
60   south_pos,_=box_move(m.pos,-d,e.box)
61   new_m=e.model_initλ(t, i,
62     ↵ to_prior.(south_pos,e.priors),south_pos,-m.v, e.obs,
63     ↵ e.constants ... ) #if west reflection fails, try south
64 end

```

```

54     new_m.log_Li < m.log_Li && (process_report!(tuner, false);
55     ↵   m.v=-m.v) #if that fails reverse the particle velocity and
56     ↵   wait for smaller τ
57 end
58
59     function box_move(pos,d,box)
60         if !(all(box[:,1].<pos+d.<box[:,2]))
61             b=isapprox.(pos,box)
62             pos[b[:,1]]+=box[b[:,1],1]
63             pos[b[:,2]]+=box[b[:,2],2]
64             boundary_d=box.-pos
65             boundary_t=boundary_d./d
66
67             ↵   first_b_idx=findfirst(isequal(minimum(boundary_t[findall(
68                 first_b_d=boundary_d[first_b_idx]
69                 d=d.*((first_b_d/d)[first_b_idx[1]]))
70             end
71
72             return pos+d, d
73         end
74
75         function box_reflect(pos,box,v)
76             if !(all(box[:,1].<pos.<box[:,2]))
77                 b=isapprox.(pos,box)
78                 low_idxs=[v[i]<0&&b[i,1] for i in 1:length(v)]
79                 hi_idxs=[v[i]>0&&b[i,2] for i in 1:length(v)]
80                 v[low_idxs]=-v[low_idxs]
81                 v[hi_idxs]=-v[hi_idxs]
82             end
83             return v
84         end
85
86         function boundary_norm(v, lhδ)
87             b=[lhδ/vi for vi in v]
88             b[b.==Inf]=prevfloat(Inf) #rectify invalid vals
89             b[b.==0.]=rand([nextfloat(0.),prevfloat(0.)])
90             b[b.==-Inf]=nextfloat(-Inf)
91             return normalize(b)
92         end

```

```

93         function reflect(v, n, η)
94             v[] = v - (2 * n * dot(n, v))
95             η > 0. ? (return r_perturb(v[], η)) : (return v[])
96         end
97
98         function r_perturb(v[], η)
99             return v[] * cos(η) +
100                rand(MvNormal(length(v[]), 1.)) * sin(η)
101         end

```

#### 16.4.4 /src/ensemble/GMC\_NS\_Engsemble.jl



```

50 Display the ensemble path, likelihood histogram, contour, maximum log
51   ↵ likelihood, and log evidence (marginal likelihood).
52 """
53 function Base.show(io::IO, e::GMC_NS_Ensemble; 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     try
61         hist=UnicodePlots.histogram(livec, title="Ensemble Likelihood
62           ↵ Distribution")
63         show(io, hist)
64         println()
65         progress && return(nrows(hist.graphics)+6)
66     catch
67         printstyled("HISTOGRAM UNAVAILABLE", color=:green)
68         println()
69         progress && return 4
70     end
71 end

```

#### 16.4.5 /src/nested\_sampler/converge\_ensemble.jl

```

1 function converge_ensemble!(e::GMC_NS_Ellement;
2     max_iterates=typemax(Int64), backup::Tuple{Bool, Integer}=(false, 0),
3     clean::Tuple{Bool, Integer, Integer}=(false, 0, 0),
4     converge_criterion::String="standard",
5     converge_factor::AbstractFloat=.01, mc_noise::AbstractFloat=0.,
6     progargs ... )
7     N = length(e.models); curr_it=length(e.log_Li)
8
9     if curr_it==1 || !isfile(e.path*"/tuner")
10        tuner_dict=get_tuner_dict(e)
11    else
12        tuner_dict=deserialize(e.path*"/tuner") #restore tuner from saved
13        → if any
14    end
15
16    meter = GMC_NS_Progress(e, 0.; start_it=curr_it, progargs ... )

```

```

12   converge_check = get_convfunc(converge_criterion)
13   while !converge_check(e, converge_factor, mc_noise) && (curr_it ≤
14     → max_iterates)
15     warn = nested_step!(e, tuner_dict)
16     warn == 1 && (@error "Failed to find new models, aborting at
17       → current iterate."; return e)
18     curr_it += 1
19
20     backup[1] && curr_it%backup[2] == 0 && e_backup(e,tuner_dict)
21       → #every backup interval, serialise the ensemble and tuner
22     clean[1] && !e.sample_posterior && curr_it%clean[2] == 0 &&
23       → clean_ensemble_dir(e,clean[3]) #every clean interval, remove
24       → old discarded models
25
26     update!(meter, converge_check(e,converge_factor,vals=true) ... )
27   end
28
29   if converge_check(e,converge_factor, mc_noise)
30     ms=measure_evidence(e)
31     @info "Job done, sampled to convergence. Final logZ $(ms.val) ±
32       → $(ms.err)"
33
34     e_backup(e,tuner_dict)
35     clean[1] && !e.sample_posterior && clean_ensemble_dir(e,0) #final
36       → clean
37     return ms
38   elseif curr_it==max_iterates
39     @info "Job done, sampled to maximum iterate $max_iterates.
40       → Convergence criterion not obtained."
41
42     e_backup(e,tuner_dict)
43     clean[1] && !e.sample_posterior && clean_ensemble_dir(e,0) #final
44       → clean
45     return e.log_Zi[end]
46   end
47 end
48
49 function evidence_converge(e, evidence_fraction,
50   → mc_noise=0.; vals=false)
51   mc_noise > 0. && noise_check(e,mc_noise) && return
52     → true
53
54   val=lps(findmax([model.log_Li for model in
55     → e.models])[1], e.log_Xi[end])

```

```

43             thresh=lps(log(evidence_fraction),e.log_Zi[end])
44
45             isinf(val)||isinf(thresh) && return false
46
47             vals ? (return val, thresh) : (return val


---



```

#### 16.4.6 /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     N = length(e.models) #number of sample models/particles on the
10    ↵ posterior surface
11    Np1=N #decrement this if a model is killed for trapezoidal approx
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    model_selected=false
24    samples=1;
25    ↵ sample_limit=Int64(floor(length(e.models)*e.GMC_exhaust_σ))
26
27    while !model_selected && samples≤sample_limit
28        if t.τ > e.GMC_τ_death
29
30            ↵ model_selected=galilean_search!(m,e,t,least_likely_idx,samples,sample_
31            elseif (N-1)<e.GMC_Nmin #if removing the particle would put the
32            ↵ sample under the minimum size, resample
33
34            ↵ model_selected=diffusion_search(e,tuners,least_likely_idx,N,e.GMC_τ_d_
35            samples+=1
36
37        else #if not, push to the posterior samples and let it die
38            Np1-=1
39            model selected=true
40
41

```

```

32         e.sample_posterior && push!(e.posterior_samples,
33             → e.models[least_likely_idx])#if sampling posterior, push
34             → the model record to the ensemble's posterior samples
35             → vector
36             deleteat!(e.models, least_likely_idx) #remove least likely
37             → model record
38     end
39 end
40
41 samples=sample_limit && (return 1)
42
43 #UPDATE ENSEMBLE QUANTITIES
44 push!(e.log_Li, minimum([model.log_Li for model in e.models])) #log
45             → likelihood of the least likely model - the current ensemble ll
46             → contour at Xi
47 push!(e.log_Xi, -i/N) #log Xi - crude estimate of the iterate's
48             → enclosed prior mass
49 push!(e.log_wi, logaddexp(e.log_Xi[i], - ((j+1)/Np1)) - log(2)) #log
50             → width of prior mass spanned by the last step-trapezoidal approx
51 push!(e.log_Liwi, lps(e.log_Li[j],e.log_wi[j])) #log likelihood + log
52             → width = increment of evidence spanned by iterate
53 push!(e.log_Zi, logaddexp(e.log_Zi[i],e.log_Liwi[j]))      #log
54             → evidence
55 #information- dimensionless quantity
56 Hj=lps(
57     (exp(lps(e.log_Liwi[j],-e.log_Zi[j])) * e.log_Li[j]),
58     (exp(lps(e.log_Zi[i],-e.log_Zi[j])) * lps(e.Hi[i],e.log_Zi[i])),
59     -e.log_Zi[j])
60 Hj ≡ -Inf ? push!(e.Hi,0.) : push!(e.Hi, Hj)
61
62 return 0
63 end

```

---

#### 16.4.7 /src/nested\_sampler/search\_patterns.jl

---

```

1 function ellipsoid_resample!(e,tdict,llidx,samples,s_limit)
2     posmat=zeros(length(e.models[1].pos),length(e.models))
3     for (n,rec) in enumerate(e.models)
4         posmat[:,n]=rec.pos
5     end
6     e_ellip=bounding_ellipsoid(posmat)
7

```

```

8 @info "Ellipsoid resample $(samples)"
9
10 new_pos=sample_ellipsoid(e_ellip)
11 box_bound!(new_pos,e.box)
12 new_m=e.model_initλ(e.t_counter, 1, to_prior.(new_pos,e.priors),
13   ↳ new_pos, [0.], e.obs, e.box, e.constants..., v_init=true)
14 samples+=1
15
16 while new_m.log_Li < e.contour && samples < s_limit
17   move_cursor_up_while_clearing_lines(stdout,1)
18   @info "Ellipsoid resample $(samples)"
19   new_pos=sample_ellipsoid(e_ellip)
20   box_bound!(new_pos,e.box)
21   new_m=e.model_initλ(e.t_counter, 1, to_prior.(new_pos,e.priors),
22     ↳ new_pos, [0.], e.obs, e.box, e.constants..., v_init=true)
23   samples+=1
24 end
25
26 if new_m.log_Li > e.contour
27   move_cursor_up_while_clearing_lines(stdout,1)
28   @info "Found"
29   e.sample_posterior && push!(e.posterior_samples,
30     ↳ e.models[llidx])#if sampling posterior, push the model record
31   ↳ to the ensemble's posterior samples vector
32   deleteat!(e.models, llidx) #remove least likely model record
33
34   new_model_record = typeof(e.models[1])(new_m.trajectory,new_m.i,
35     ↳ new_m.pos, string(e.path,'/',new_m.trajectory,'.',new_m.i),
36     ↳ new_m.log_Li)
37   push!(e.models, new_model_record) #insert new model record
38   serialize(new_model_record.path, new_m)
39   tdict[e.t_counter]=τ_PID(e)
40   e.t_counter +=1
41   move_cursor_up_while_clearing_lines(stdout,1)
42   return true
43 else
44   move_cursor_up_while_clearing_lines(stdout,1)
45   @info "Ellipsoid resample failure!"
46   move_cursor_up_while_clearing_lines(stdout,1)
47   return false
48 end
49 end

```

```

45 function galilean_search!(m,e,t,llidx,samples,s_limit)
46     candidate=galilean_trajectory_sample!(m,e,t)
47
48     @info "Galilean search $(samples), trajectory $(m.trajectory), i
        ↳ $(m.i), τ $(t.τ) + $(e.GMC_τ_death)"
49
50     while candidate.i==m.i && samples≤s_limit && t.τ >
        ↳ e.GMC_τ_death#failure to find any new step, return tau and try
        ↳ again until sample limit reached
51         move_cursor_up_while_clearing_lines(stdout,1)
52         @info "Galilean search $(samples), trajectory $(m.trajectory), i
            ↳ $(m.i), τ $(t.τ) + $(e.GMC_τ_death)"
53
54         candidate=galilean_trajectory_sample!(m,e,t)
55         samples+=1
56         samples%4==0 && (m.v=rand(MvNormal(length(m.θ),1.)))
57     end
58
59     if candidate.i!=m.i
60         move_cursor_up_while_clearing_lines(stdout,1)
61         @info "Found"
62
63         e.sample_posterior && push!(e.posterior_samples,
        ↳ e.models[llidx])#if sampling posterior, push the model record
        ↳ to the ensemble's posterior samples vector
64         deleteat!(e.models, llidx) #remove least likely model record
65
66         new_model_record =
        ↳ typeof(e.models[1])(candidate.trajectory,candidate.i,
        ↳ candidate.pos,
        ↳ string(e.path,'/',candidate.trajectory,'.',candidate.i),
        ↳ candidate.log_Li)
67         push!(e.models, new_model_record) #insert new model record
68         serialize(new_model_record.path, candidate)
69         move_cursor_up_while_clearing_lines(stdout,1)
70
71         return true
72     else
73         move_cursor_up_while_clearing_lines(stdout,1)
74         @info "+"
75         move_cursor_up_while_clearing_lines(stdout,1)
76         return false
77     end

```

```

78 end

79
80 function diffusion_search(e, tuners, llidx, sample_limit, τ_limit)
81     sample=0; model_selected=false; new_m=0; src_traj=0
82     @info "Diffusion resample"
83     while !(sample > sample_limit) && !model_selected
84         move_cursor_up_while_clearing_lines(stdout,1)

85
86     sample +=1
87     mrec=rand(e.models)
88     model=deserialize(mrec.path)
89     randdir=rand(MvNormal(length(model.θ),1.))
90     τ=tuners[model.trajectory].τ
91     @info "Diffusion resample $(sample) from $(model.trajectory) with
92         → τ $τ"
93     new_pos=model.pos+randdir*τ
94     box_bound!(new_pos,e.box)
95     new_m=e.model_initλ(e.t_counter, 1, to_prior.(new_pos,e.priors),
96         → new_pos, randdir, e.obs, e.constants ... )
97
98     while τ*.7>τ_limit && new_m.log_Li<e.contour
99         τ*=.7
100        move_cursor_up_while_clearing_lines(stdout,1)
101        @info "Diffusion resample $(sample) from $(model.trajectory)
102            → with τ $τ"
103        randdir=rand(MvNormal(length(model.θ),1.))
104        new_pos=model.pos+randdir*τ
105        box_bound!(new_pos,e.box)
106        new_m=e.model_initλ(e.t_counter, 1,
107            → to_prior.(new_pos,e.priors), new_pos, randdir, e.obs,
108            → e.constants ... )
109    end
110
111    new_m.log_Li > e.contour && (model_selected=true;
112        → src_traj=model.trajectory)
113
114 if new_m.log_Li > e.contour
115     move_cursor_up_while_clearing_lines(stdout,1)
116     @info "Found"
117     e.sample_posterior && push!(e.posterior_samples,
118         → e.models[llidx])#if sampling posterior, push the model record
119         → to the ensemble's posterior samples vector

```

```

113     deleteat!(e.models, llidx) #remove least likely model record
114
115     new_model_record = typeof(e.models[1])(new_m.trajectory,new_m.i,
116         ↳ new_m.pos, string(e.path,'/'),new_m.trajectory,'.',new_m.i),
117         ↳ new_m.log_Li)
118     push!(e.models, new_model_record) #insert new model record
119     serialize(new_model_record.path, new_m)
120     tuners[e.t_counter]=τ_PID(e)
121     tuners[e.t_counter].τ=tuners[src_traj].τ
122     e.t_counter +=1
123     move_cursor_up_while_clearing_lines(stdout,1)
124     return true
125   else
126     move_cursor_up_while_clearing_lines(stdout,1)
127     @info "Diffusion resample failure!"
128     move_cursor_up_while_clearing_lines(stdout,1)
129     return false
130   end
131 end

```

---

#### 16.4.8 /src/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
35 t_counter::Int64
36 end
37
38 LogNormal_Engsemble(path::String, no_models::Integer,
39   ← obs::AbstractVector{<:AbstractFloat}, prior, box, GMC_settings ... ;
40   ← sample_posterior::Bool=true) =
41 LogNormal_Engsemble(
42   path,
43   construct_lognormal_model,
44   assemble_lNMs(path, no_models, obs, prior, box, Vector{Real}()) ... ,
45   [-Inf], #L0 = 0
46     [0.], #ie exp(0) = all of the prior is covered
47     [-Inf], #w0 = 0
48     [-Inf], #Liwi0 = 0
49     [-1e300], #Z0 = 0
50     [0.], #H0 = 0,
51   obs,
52   length(prior)=1 ? ([GMC_NS.marginals(prior) ... ]) : (prior),
53   Vector{Real}(),
54   length(prior)=1 ? (to_unit_ball.(box,[GMC_NS.marginals(prior) ... ]))
55   ← : (to_unit_ball.(box,prior)),
56   sample_posterior,
57   Vector{Normal_Record}(),
58   GMC_settings ... ,
59   no_models+1)
60
61 function Base.show(io::IO, m::LogNormal_Model, e::LogNormal_Engsemble;
62   ← xsteps=100, progress=true)
63   μ, λ=m.θ

```

```

60   lμ=log(μ^2/sqrt(μ^2 + inv(λ)))
61   lσ=sqrt(log(1+(inv(λ)/μ^2)))
62   n=LogNormal(lμ,lσ)
63
64   ↳ X=[quantile(n,.025):(quantile(n,.975)-quantile(n,.025))/xsteps:quantile(n,.975)]
65
66   scattermin=maximum(y)*.25
67   scattermax=maximum(y)*.75
68
69   ↳ scatterrange=[scattermin:(scattermax-scattermin)/length(e.obs):scattermax ...]
70   scattery=[rand(scatterrange) for i in 1:length(e.obs)]
71
72   xlims=(min(minimum(e.obs),minimum(X)),max(maximum(e.obs),maximum(X)))
73
74   plt=lineplot(X,y,xlabel="X",ylabel="p",title="LogNormal Model"
75   ↳ $(m.trajectory).$(m.i), log_Li $(m.log_Li)", name="Model",
76   ↳ xlim=xlims)
77   scatterplot!(plt, e.obs, scattery, name="Obs")
78
79   show(io, plt)
80   println()
81   println("θ: $(m.θ)")
82   println("v: $(m.v)")
83
84   progress && return nrows(plt.graphics)+8
85 end
86
87
88 function assemble_lNMs(path::String, no_trajectories::Integer, obs,
89   ↳ prior, box, constants)
90   ensemble_records = Vector{Normal_Record}()
91   !isdir(path) && mkpath(path)
92   length(prior)==1 ? (marginals=[GMC_NS.marginals(prior)...]) :
93   ↳ (marginals=prior)
94   box=to_unit_ball.(box,marginals)
95
96   @showprogress 1 "Assembling log-Normal Model ensemble..." for
97   ↳ trajectory_no in 1:no_trajectories
98     model_path = string(path,'/','trajectory_no','.',1)
99     if !isfile(model_path)
100       proposal=[rand.(prior)...]
101       pos=to_unit_ball.(proposal,marginals)
102       box_bound!(pos,box)
103     end
104   end
105 end

```

```

96         θvec=to_prior.(pos,marginals)
97
98         model = LogNormal_Model(trajectory_no, 1, θvec, pos, [0.],
99             ← 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

```

---

### 16.4.9 /src/normal/LogNormal\_Model.jl

```

1 mutable struct LogNormal_Model <: GMC_NS_Model
2     trajectory::Integer #id integer assigned at construction
3     i::Integer #id of parent model, if any
4
5     θ::Vector{Float64} #model's parameter Vector
6     log_Li::Float64 #model's likelihood, calculated by constructor
7
8     pos::Vector{Float64}
9     v::Vector{Float64} #model's velocity in parameter space
10
11    function LogNormal_Model(trajectory::Integer, i::Integer,
12        ← θ::Vector{Float64}, pos::Vector{Float64}, v::Vector{Float64},
13        ← obs::Vector{Float64}; v_init=false)
14        μ, λ=θ
15        lμ=log(μ^2/sqrt(μ^2 + inv(λ)))
16        lσ=sqrt(log(1+inv(λ)/μ^2))
17        mod_lnormal=LogNormal(lμ,lσ)
18        log_lh=lps(logpdf(mod_lnormal, obs))
19        v_init && (v=rand(MvNormal(length(θ),1.)))

```

```

19         new(trajectory, i, θ, log_lh, pos, v)
20     end
21 end
22
23 function construct_lognormal_model(args ... ; kwargs ... )
24     return LogNormal_Model(args ... ; kwargs ... )
25 end
26
27 function Base.show(io::IO, m::LogNormal_Model; xsteps=100)
28     μ, λ=m.θ
29     lμ=log(μ^2/sqrt(μ^2 + inv(λ)))
30     lσ=sqrt(log(1+(inv(λ)/μ^2)))
31     n=LogNormal(lμ,lσ)
32
33     → X=[quantile(n,.025):(quantile(n,.975)-quantile(n,.025))/xsteps:quantile(n,.975)]
34     y=pdf.(n,X)
35
36     show(io, lineplot(X,y,xlabel="X",ylabel="p",title=title="LogNormal
37     → Model $(m.trajectory).$(m.i), log_Li $(m.log_Li)"))
38     println()
39     println("θ: $(m.θ)")
40     println("v: $(m.v)")
41 end

```

---

#### 16.4.10 /src/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
35   t_counter::Int64
36 end
37
38 Normal_Engsemble(path::String, no_models::Integer,
39   ↳ obs::AbstractVector{<:AbstractFloat}, prior, box, GMC_settings ... ;
40   ↳ sample_posterior::Bool=true) =
41
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.], #ie exp(0) = all of the prior is covered
48     [-Inf], #w₀ = 0
49     [-Inf], #Liwi₀ = 0
50     [-1e300], #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)

```

```

57
58 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   progress && return nrows(plt.graphics)+8
81 end
82
83 function assemble_NM(path::String, no_trajectories::Integer, obs, prior,
84   ↪ box, constants)
85   ensemble_records = Vector{Normal_Record}()
86   !isdir(path) && mkpath(path)
87   length(prior)==1 ? (marginals=[GMC_NS.marginals(prior) ... ]) :
88   ↪ (marginals=prior)
89   box=to_unit_ball.(box,marginals)
90
91   @showprogress 1 "Assembling Normal Model ensemble..." for
92     ↪ trajectory_no in 1:no_trajectories
93       model_path = string(path,'/',trajectory_no,'.',1)
94       if !isfile(model_path)
95         proposal=[rand.(prior) ... ]

```

```

93         pos=to_unit_ball.(proposal,marginals)
94         box_bound!(pos,box)
95         θvec=to_prior.(pos,marginals)
96
97         model = Normal_Model(trajectory_no, 1, θvec, pos, [0.], obs,
98             → constants ... ; v_init=true)
99
100            serialize(model_path, model) #save the model to
101            → the ensemble directory
102            push!(ensemble_records,
103                → Normal_Record(trajectory_no,1,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,
107                    → Normal_Record(trajectory_no,1,model.pos,model_path,model.log_Li))
108        end
109    end
110
111    return ensemble_records, minimum([record.log_Li for record in
112        → ensemble_records])
113 end

```

---

#### 16.4.11 /src/normal/Normal\_Model.jl

```

1 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 mutable 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

```

```

19   function Normal_Model(trajectory :: Integer, i :: Integer,
20     ↳ θ :: Vector{Float64}, pos :: Vector{Float64}, v :: Vector{Float64},
21     ↳ obs :: Vector{Float64}; v_init=false)
22     μ, λ = θ
23     mod_normal = Normal(μ, sqrt(1/λ))
24     log_lh = lps(logpdf(mod_normal, obs))
25     v_init && (v = rand(MvNormal(length(θ), 1.)))
26
27     new(trajectory, i, θ, log_lh, pos, v)
28   end
29 end
30
31 function construct_normal_model(args ... ; kwargs ... )
32   return Normal_Model(args ... ; kwargs ... )
33 end
34
35 function Base.show(io::IO, m::Normal_Model; xsteps=100)
36   μ, λ = m.θ
37   σ = sqrt(1/λ)
38   n = Normal(μ, σ)
39   X = [μ - 4σ : ((μ + 4σ) - (μ - 4σ)) / xsteps : μ + 4σ ... ]
40   y = pdf.(n, X)
41
42   show(io, lineplot(X, y, xlabel="X", ylabel="p", title="Normal Model
43     ↳ $(m.trajectory).$(m.i), log_Li $(m.log_Li)"))
44   println()
45   println("θ: $(m.θ)")
46   println("v: $(m.v)")
47 end

```

---

### 16.4.12 /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

```

---

### 16.4.13 /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 nbball_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

```

---

#### 16.4.14 /src/utilities/ensemble\_utilities.jl

```

1 function ensemble_history(e::GMC_NS_Ensemble, 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_Ensemble, tuner::Dict{Int64,<:GMC_Tuner})
10    serialize(string(e.path,'/',"ens"), e)
11    serialize(string(e.path,'/',"tuner"), tuner)
12 end
13
14 function clean_ensemble_dir(e::GMC_NS_Ensemble, model_pad::Integer;
15     # → ignore_warn=false)
16     !ignore_warn && e.sample_posterior && throw(ArgumentError("Ensemble
17     # → is set to retain posterior samples and its directory should not
18     # → be cleaned!"))
19     for file in readdir(e.path)
20         !(file in vcat([basename(model.path) for model in
21             e.models], "ens", [string(number) for number in
22                 e.model_counter-length(e.models)-model_pad:e.model_counter-1])) #
23             && rm(e.path*'*file)

```

```

16     end
17 end
18
19 function complete_evidence(e::GMC_NS_Ensemble)
20     return final_logZ = logaddexp(e.log_Zi[end], (logsumexp([model.log_Li
21         → for model in e.models] .+ (e.log_Xi[length(e.log_Li)] -
22         → log(length(e.models))))))
23 end
24
25 function measure_evidence(e::GMC_NS_Ensemble)
26     return
27         → ms=measurement(complete_evidence(e),sqrt(abs(e.Hi[end])/length(e.models)))
28 end
29
30
31 function reset_ensemble!(e::GMC_NS_Ensemble)
32     new_e=deepcopy(e)
33     for i in 1:length(e.models)
34         if string(i) in [basename(record.path) for record in e.models]
35             new_e.models[i]=e.models[findfirst(isequal(string(i)),
36                 → [basename(record.path) for record in e.models])]
37         else
38             new_e.models[i]=e.posterior_samples[findfirst(isequal(string(i)),
39                 → [basename(record.path) for record in
40                     → e.posterior_samples])]
41         end
42     end
43     new_e.contour=minimum([record.log_Li for record in new_e.models])
44
45     new_e.log_Li=[new_e.log_Li[1]]
46     new_e.log_Xi=[new_e.log_Xi[1]]
47     new_e.log_wi=[new_e.log_wi[1]]
48     new_e.log_Liwi=[new_e.log_Liwi[1]]
49     new_e.log_Zi=[new_e.log_Zi[1]]
50     new_e.Hi=[new_e.Hi[1]]
51
52     new_e.posterior_samples=Vector{typeof(e.models[1])}()
53
54     new_e.model_counter=length(new_e.models)+1
55
56     clean_ensemble_dir(new_e, 0; ignore_warn=true)

```

```
52     isfile(e.path*/'tuner') && rm(e.path*/'tuner')
53     serialize(e.path*/'ens', new_e)
54
55     return new_e
56 end
57
58 function move_ensemble!(e::GMC_NS_Ensemble, path::String)
59     !isdir(path) && mkdir(path)
60     for file in readdir(e.path)
61         mv(e.path*/'*file, path*/'*file)
62     end
63
64     for (n,model) in enumerate(e.models)
65         e.models[n]=typeof(e.models[1])(path*/'*basename(model.path),
66             → model.log_Li)
67     end
68     if e.sample_posterior
69         for (n,model) in enumerate(e.posterior_samples)
70             → e.posterior_samples[n]=typeof(e.models[1])(path*/'*basename(model.pa
71             → model.log_Li)
72         end
73     end
74     rm(e.path)
75     e.path=path
76     serialize(e.path*/'ens',e)
77     return e
78 end
79 function copy_ensemble!(e::GMC_NS_Ensemble, path::String)
80     new_e=deepcopy(e)
81     !isdir(path) && mkdir(path)
82     for file in readdir(e.path)
83         cp(e.path*/'*file, path*/'*file, force=true)
84     end
85
86     for (n,model) in enumerate(e.models)
87         new_e.models[n]=Model_Record(path*/'*basename(model.path),
88             → model.log_Li)
89     end
90     if e.sample_posterior
91         for (n,model) in enumerate(e.posterior_samples)
```

```

91         ↪ new_e.posterior_samples[n]=Model_Record(path*'/'*basename(model.path)
92             ↪ model.log_Li)
93     end
94
95     new_e.path=path
96     serialize(new_e.path*"/ens",e)
97     return new_e
98 end
99
100 function rewind_ensemble(e::GMC_NS_Ensemble,rewind_idx)
101     !e.sample_posterior && throw(ArgumentError("An ensemble not retaining
102         ↪ posterior samples cannot be rewound!"))
103     rewind_idx ≥ length(e.log_Li) && throw(ArgumentError("rewind_idx
104         ↪ must be less than the current iterate!"))
105
106     n=length(e.models)
107     max_model_no=length(e.log_Li)+length(e.models)-1
108     rewind_model_no=rewind_idx+length(e.models)-1
109     new_e = deepcopy(e)
110
111     rm_models=[string(name) for name in rewind_model_no+1:max_model_no]
112
113     filter!(model→!(basename(model.path) in rm_models),new_e.models)
114     filter!(model→!(basename(model.path) in
115         ↪ rm_models),new_e.posterior_samples)
116
117     while length(new_e.models) < n
118         push!(new_e.models,pop!(new_e.posterior_samples))
119     end
120     new_e.contour=new_e.log_Li[rewind_idx]
121     new_e.log_Li=new_e.log_Li[1:rewind_idx]
122     new_e.log_Xi=new_e.log_Xi[1:rewind_idx]
123     new_e.log_wi=new_e.log_wi[1:rewind_idx]
124     new_e.log_Liwi=new_e.log_Liwi[1:rewind_idx]
125     new_e.log_Zi=new_e.log_Zi[1:rewind_idx]
126     new_e.Hi=new_e.Hi[1:rewind_idx]
127
128     new_e.model_counter=length(new_e.models)+rewind_idx
129
130     return new_e
131 end

```

```

129
130 function show_models(e::GMC_NS_Ensemble,idxs)
131     liperm=sortperm([model.log_Li for model in e.models],rev=true)
132     for idx in idxs
133         m=deserialize(e.models[liperm[idx]].path)
134         show(m)
135     end
136 end
137
138 function show_models_e(e::GMC_NS_Ensemble,idxs)
139     liperm=sortperm([model.log_Li for model in e.models],rev=true)
140     for idx in idxs
141         m=deserialize(e.models[liperm[idx]].path)
142         show(stdout,m,e)
143     end
144 end
145
146 function get_model(e::GMC_NS_Ensemble,no)
147     return deserialize(e.path*"/"*string(no))
148 end
149
150 #recalculate the ensemble's history from the succession of log_Li and
151 #      ↵ log_Xi values
152 function rectify_ensemble!(e::GMC_NS_Ensemble, wi_mode="trapezoidal")
153     Ni=round.(collect(-1:-1:-length(e.log_Li)+1)./e.log_Xi[2:end])
154     push!(Ni,Ni[end])
155
156     for i in 1:length(e.log_Li)-1
157         j=i+1
158         if wi_mode=="trapezoidal"
159             e.log_wi[j]= logaddexp(e.log_Xi[i], - ((j+1)/Ni[j])) - log(2)
160             #log width of prior mass spanned by the last
161             #step-trapezoidal approx
162         elseif wi_mode=="simple"
163             e.log_wi[j]= logaddexp(e.log_Xi[i], - e.log_Xi[j]/Ni[i]) #log
164             #width of prior mass spanned by the last step-simple
165             #approx
166         else
167             throw(ArgumentError("Unsupported wi_mode!"))
168         end
169         e.log_Liwi[j]=lps(e.log_Li[j],e.log_wi[j]) #log likelihood + log
170             #width = increment of evidence spanned by iterate
171         e.log_Zi[j]=logaddexp(e.log_Zi[i],e.log_Liwi[j])    #log evidence

```

```

166     #information- dimensionless quantity
167     Hj=lps(
168         (exp(lps(e.log_Liwi[j],-e.log_Zi[j])) * e.log_Li[j]),
169         (exp(lps(e.log_Zi[i],-e.log_Zi[j])) *
170             → lps(e.Hi[i],e.log_Zi[i])),
171         -e.log_Zi[j])
172     Hj == -Inf ? (e.Hi[j]=0.) : (e.Hi[j]=Hj)
173 end
174
175 #get a posterior kernel density estimate from the ensemble
176 function posterior_kde(e::GMC_NS_Engine, scale=400., bw=1.)
177     !e.sample_posterior && throw(ArgumentError("This ensemble is not set
178         → to collect posterior samples!"))
179
180     → θ_mat=zeros(length(e.models[1].pos),length(e.models)+length(e.posterior_samples))
181     weight_vec=zeros(length(e.models)+length(e.posterior_samples))
182
183     for (n,mrec) in enumerate(e.posterior_samples)
184         m=deserialize(mrec.path)
185         θ_mat[:,n]=m.θ
186         weight_vec[n]=e.log_Liwi[n+1]
187     end
188
189     for (n,mrec) in enumerate(e.models)
190         m=deserialize(mrec.path)
191         p=length(e.posterior_samples)
192         θ_mat[:,n+p]=m.θ
193         weight_vec[n+p]=m.log_Li + lps(e.log_Xi[end], -length(e.models))
194     end
195
196     weight_vec-=maximum(weight_vec)-scale
197
198     return kde!(θ_mat,[bw],exp.(weight_vec))
199 end

```

---

### 16.4.15 /src/utilities/ns\_progressmeter.jl

---

```

1 const CONVERGENCE_MEMORY=500
2
3 mutable struct GMC_NS_Progress{T<:Real} <: AbstractProgress

```



```

43                         start_it::Int=1,
44                         upper_displays=[[evidence_display]],
45                         lower_displays=[[ensemble_display]],
46                         disp_rot_its=0,
47                         ) where T
48
49     tfirst = tlast = time()
50
51     printed = false
52
53     new{T}(interval,
54           dt,
55           start_it,
56           start_it,
57           false,
58           tfirst,
59           tlast,
60           0.,
61           printed,
62           desc,
63           color,
64           output,
65           0,
66           offset,
67           e,
68           #tuner,
69           top_m,
70           0.,
71           upper_displays,
72           lower_displays,
73           disp_rot_its,
74           1,
75           1,
76           zeros(CONVERGENCE_MEMORY),
77           zeros(CONVERGENCE_MEMORY))
78
79     end
80   end
81
82   function GMC_NS_Progress(e::GMC_NS_E ensemble,
83     #tuner::GMC_Tuner,
84     interval::Real; dt::Real=0.1, desc::AbstractString="GMC-NS::",
85     → color::Symbol=:green, output::IO=stderr, offset::Integer=0,
86     → start_it::Integer=1,
87     → upper_displays::AbstractVector{<:AbstractVector{Function}},
88     → lower_displays::AbstractVector{<:AbstractVector{Function}},
89     → disp_rot_its::Integer)

```

```

81     top_m = deserialize(e.models[findmax([model.log_Li for model in
82         → e.models])[2]].path)
83
84     return GMC_NS_Progress{typeof(interval)}(e, top_m,
85         # tuner,
86         interval, dt=dt, desc=desc, color=color, output=output,
87         → offset=offset, start_it=start_it, upper_displays=upper_displays,
88         → lower_displays=lower_displays, disp_rot_its=disp_rot_its)
89 end
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114

```

`top_m = deserialize(e.models[findmax([model.log_Li for model in
→ e.models])[2]].path)

return GMC_NS_Progress{typeof(interval)}(e, top_m,
# tuner,
interval, dt=dt, desc=desc, color=color, output=output,
→ offset=offset, start_it=start_it, upper_displays=upper_displays,
→ lower_displays=lower_displays, disp_rot_its=disp_rot_its)

end

function update!(p::GMC_NS_Progress, val, thresh; options ... )
 p.counter += 1

 p.tstp=time()-p.tlast
 popfirst!(p.time_history)
 push!(p.time_history, p.tstp)

 p.interval = val - thresh
 popfirst!(p.convergence_history)
 push!(p.convergence_history, p.interval)

 string(p.top_m.trajectory,'.',p.top_m.i) ≠
 → basename(p.e.models[findmax([model.log_Li for model in
 → p.e.models])[2]].path) &&
 → (p.top_m=deserialize(p.e.models[findmax([model.log_Li for model
 → in p.e.models])[2]].path))

 updateProgress!(p; options ... )

end

function updateProgress!(p::GMC_NS_Progress; offset::Integer = p.offset,
 → keep = (offset == 0))
 p.offset = offset
 t = time()
 p.display_rotate_iterates > 0 && p.counter%p.display_rotate_iterates
 → = 0 && rotate_displays(p)

 if p.interval ≤ 0 && !p.triggered
 p.triggered = true
 if p.printed
 p.triggered = true
 dur = durationstring(t-p.tfist)
 end
 end
end`

```

115     msg = @sprintf "%s Converged. Time: %s (%d iterations). logZ:
116         → %s\n" p.desc dur p.counter p.e.log_Zi[end]
117
118     print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
119     move_cursor_up_while_clearing_lines(p.output,
120         → p.numprintedvalues)
121     length(p.upper_displays)>0 ? (upper_lines=display_dash(p,
122         → p.upper_displays[p.upperidx])) : (upper_lines=0)
123     printover(p.output, msg, :magenta)
124     length(p.lower_displays)>0 ? (lower_lines=display_dash(p,
125         → p.lower_displays[p.loweridx])) : (lower_lines=0)
126
127     p.numprintedvalues=upper_lines + lower_lines + 1
128
129     if keep
130         println(p.output)
131     else
132         print(p.output, "\r\u1b[A" ^ (p.offset +
133             → p.numprintedvalues))
134     end
135   end
136   return
137 end
138
139 if t > p.tlast+p.dt && !p.triggered
140   p.counter < CONVERGENCE_MEMORY ?
141       mean_step_time=mean(p.time_history[end-(p.counter-1):end]) :
142           → mean_step_time=mean(p.time_history)
143   msg = @sprintf "%s Iterate: %s Recent step time μ: %s Convergence
144       → Interval: %g\n" p.desc p.counter hmss(mean_step_time)
145       → p.interval
146
147   print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
148   move_cursor_up_while_clearing_lines(p.output, p.numprintedvalues)
149   length(p.upper_displays)>0 ? (upper_lines=display_dash(p,
150         → p.upper_displays[p.upperidx])) : (upper_lines=0)
151   printover(p.output, msg, p.color)
152   length(p.lower_displays)>0 ? (lower_lines=display_dash(p,
153         → p.lower_displays[p.loweridx])) : (lower_lines=0)
154
155   p.numprintedvalues=upper_lines + lower_lines + 1
156   print(p.output, "\r\u1b[A" ^ (p.offset + p.numprintedvalues))

```

```

147
148      # Compensate for any overhead of printing. This can be
149      # especially important if you're running over a slow network
150      # connection.
151      p.tlast = t + 2*(time()-t)
152      p.printed = true
153  end
154 end
155
156     function rotate_displays(p)
157         p.upperidx < length(p.upper_displays) ? p.upperidx+=1
158         ↪   : p.upperidx=1
159         p.loweridx < length(p.lower_displays) ? p.loweridx+=1
160         ↪   : p.loweridx=1
161     end
162
163     function hmss(dt)
164         dt<0 ? (dt=-dt; prfx="-") : (prfx="")
165         isnan(dt) && return "NaN"
166         (h,r) = divrem(dt,60*60)
167         (m,r) = divrem(r, 60)
168         (isnan(h)||isnan(m)||isnan(r)) && return "NaN"
169         string(prfx,Int(h),":",Int(m),":",Int(ceil(r)))
170     end
171
172     function display_dash(p::GMC_NS_Progress,
173         ↪   funcvec::Vector{Function})
174         lines=0
175         for func in funcvec
176             lines+=func(p)
177         end
178         return lines
179     end

```

---

#### 16.4.16 /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    show(p.output, ciplot); println()
15    return lines
16  end
17 end
18
19 function evidence_display(p)
20   try
21     evplot=lineplot(p.e.log_Zi[2:end], title="Evidence History",
22     ↳ xlabel="Iterate", color=:red, name="Ensemble logZ")
23     lines=nrows(evplot.graphics)+5
24     show(p.output, evplot); println()
25     return lines
26   catch
27     printstyled(p.output, "EVIDENCE PLOT UNAVAILABLE. STANDBY\n",
28     ↳ bold=true, color=:red); println()
29     return 2
30   end
31 end
32
33 function info_display(p)
34   try
35     infoplot=lineplot(p.e.Hi[2:end], title="Information History",
36     ↳ xlabel="Iterate", color=:green, name="Ensemble H")
37     lines=nrows(infoplot.graphics)+5
38     show(p.output, infoplot); println()
39     return lines
40   catch
41     printstyled(p.output, "INFORMATION PLOT UNAVAILABLE. STANDBY\n",
42     ↳ bold=true, color=:green); println()
43     return 2
44   end
45 end
46
47 end

```

```

43 function lh_display(p)
44   try
45     lhplot=lineplot(p.e.log_Li[2:end], title="Contour History",
46     ↪ xlabel="Iterate", color=:magenta, name="Ensemble logLH")
47     lines=nrows(lhplot.graphics)+5
48     show(p.output, lhplot); println()
49     return lines
50   catch
51     printstyled(p.output, "CONTOUR HISTORY UNAVAILABLE. STANDBY\n",
52     ↪ bold=true, color=:magenta); println()
53     return 2
54   end
55 end

56 function liwi_display(p)
57   try
58     ↪ liwiplot=lineplot([max(2,p.counter-(CONVERGENCE_MEMORY-1)):p.counter ... ],
59     ↪ title="Recent iterate evidentiary weight", xlabel="Iterate",
60     ↪ name="Ensemble log Liwi", color=:cyan)
61     lines=nrows(liwiplot.graphics)+5
62     show(p.output, liwiplot); println()
63     return lines
64   catch
65     printstyled(p.output, "EVIDENTIARY HISTORY UNAVAILABLE.
66     ↪ STANDBY\n", bold=true, color=:cyan); println()
67     return 2
68   end
69 end

70 function ensemble_display(p)
71   return lines=show(p.output, p.e, progress=true)
72 end

73 function model_display(p)
74   try
75     println("Current MAP model:")
76     return lines=show(p.output, p.top_m, progress=true)
77   catch
78     printstyled(p.output, "MODEL DISPLAY UNAVAILABLE\n", bold=true,
79     ↪ color=:blue); println()
80     return 3
81   end

```

```

79 end
80
81 function model_obs_display(p)
82     try
83         println("Current MAP model")
84         return lines=show(p.output, p.top_m, p.e, progress=true)
85     catch
86         printstyled(p.output, "MODEL DISPLAY UNAVAILABLE\n", bold=true,
87                     ↪ color=:blue); println()
88         return 3
89     end
90 end

```

---

#### 16.4.17 /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
11     ↪ m_T=MarginalTDist(d.mu,sqrt((d.shape*inv(d.v0))/d.scale),TDist(2*d.shape))
12     m_Ga=InverseGamma(d.shape,d.scale)
13     return m_T,m_Ga
14 end
15
16 function marginals(d::NormalGamma)
17     m_T=MarginalTDist(d.mu,sqrt(d.rate/(d.shape*d.nu)),TDist(2*d.shape))
18     m_Ga=Gamma(d.shape,1/d.rate)
19     return m_T,m_Ga
20 end

```

---

#### 16.4.18 /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(0),
14            e.GMC_tune_α,
15            e.GMC_tune_β,
16            falses(0),
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     iterm::Float64
38     dterm::Float64
39
40    function τ_PID(e::GMC_NS_Ensemble)
41        new(e.GMC_init_τ,
42            zeros(0),
43            e.GMC_tune_α,
44            1.,
45            falses(0),
46            falses(0),
```

```

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, update::Bool=true)
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   update && 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.[] [end] - t.α
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.[] [end-Int64(round(.1*t.memory))] - t.[] [end]) : #d is positive
71   ↳ if acceptance is declining- in this case want smaller beta and
72   ↳ tau
73   ↳ d=0.
74   t.dterm=t.Kd * d
75
76   t.β=max(eps(), 1. + t.pterm + t.iterm - t.dterm)
77   t.τ=max(eps(), t.τ * t.β)
78 end
79
80
81 function init_tune(e::GMC_NS_Ensemble)
82   @info "Performing initial GMC timestep tuning on ensemble ... "
83   τ=1.; tuned=false; τd=1.1; accept=1.; it=1
84   while !tuned && it<e.GMC_tune_l
85     test_report=false(0)
86     println(stdout, "τ: $τ, It: $it, Accept: $accept")
87
88   while
89     ↳ length(test_report)<Int64(floor(length(e.models)*e.GMC_tune_p))

```

```

85         m=deserialize(rand([m.path for m in e.models]))
86         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
115 function tune_τ!(t::τ_Tuner, step_report::BitVector)
116     push!(t.τ_history,t.τ)
117     t.a=vcat(t.a,step_report)
118     ls=length(t.a)
119     ls>t.memory && (t.a=t.a[ls-t.memory:end])
120
121     accept=sum(t.a)/ls
122     if accept==1.
123         t.τ=t.τ*(1+t.β)
124     elseif accept<t.α
125         t.τ=t.τ*(1-t.β)

```

```

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
158     catch
159         printstyled("τ_PID NOT AVAILABLE. STANDBY", bold=true)
160         progress && return 1
161     end
162 end
163
164 function get_tuner_dict(e)
165     d=Dict{Int64,τ_PID}()
166     for rec in e.models
167         d[rec.trajectory]=τ_PID(e)
168     end
169     return d
170 end
171
172 end

```

---

### 16.4.19 /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)

```

---

## 16.5 CMZNicheSims

### 16.5.1 /src/BioSimpleStochastic.jl

---

```

1 module BioSimpleStochastic
2     using Distributions, GMC_NS, UnicodePlots
3     import ProgressMeter:
4         → @showprogress,move_cursor_up_while_clearing_lines
5     import BioBackgroundModels:lps
6     import StatsFuns:logsumexp
7     import Serialization:serialize,deserialize
8
9     include("thymidine_sim/thymidine_cell.jl")
10    include("thymidine_sim/thymidine_model.jl")
11    include("thymidine_sim/thymidine_ensemble.jl")

```

```

11   export Thymidine_Ensemble
12   include("thymidine_sim/thymidine_sim.jl")
13   include("thymidine_sim/thymidine_lh.jl")
14   include("CMZ_sim/CMZ_lh.jl")
15   include("CMZ_sim/CMZ_model.jl")
16   include("CMZ_sim/CMZ_ensemble.jl")
17   export CMZ_Ensemble
18   include("slice_pop_sim/lens_model.jl")
19   export Lens_Model
20   include("slice_pop_sim/slice_lh.jl")
21   include("slice_pop_sim/slice_model.jl")
22   include("slice_pop_sim/slice_ensemble.jl")
23   export Slice_Ensemble
24
25
26 end # module

```

---

### 16.5.2 /src/CMZ\_sim/CMZ\_ensemble.jl

```

1 mutable struct CMZ_Ensemble <: GMC_NS_Ensemble
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
36     t_counter::Int64
37 end
38
39 CMZ_Elbo(path::String, no_models::Integer,
40           ← obs ::AbstractVector{<:Tuple{<:AbstractVector{<:Float64},<:AbstractVector{<:Float64}}}
41           ← priors ::AbstractVector{<:Distribution}, constants, box, GMC_settings;
42           ← sample_posterior ::Bool=true) =
43 CMZ_Elbo(
44     path,
45     construct_CMZ,
46     assemble_CMs(path, no_models, obs, priors, constants, box) ... ,
47     [-Inf], #L0 = 0
48         [0], #ie exp(0) = all of the prior is covered
49         [-Inf], #w0 = 0
50         [-Inf], #Liwi0 = 0
51         [-1e300], #Z0 = 0
52         [0], #H0 = 0,
53     obs,
54     priors,
55     constants, #T, popdist, voldist, mc_its, phs
56     box,
57     sample_posterior,
58     Vector{CMZ_Record}(),
59     GMC_settings ... ,
60     no_models+1)
61
62 function assemble_CMs(path::String, no_trajectories::Integer, obs,
63           ← priors, constants, box)
64     ensemble_records = Vector{CMZ_Record}()
65     !isdir(path) && mkpath(path)

```

```

62 phs=constants[6]
63 @showprogress 1 "Assembling CMZ_Model ensemble ... " for trajectory_no
64   in 1:no_trajectories
65   model_path = string(path,'/',trajectory_no,'.',1)
66   if !isfile(model_path)
67     proposal=rand.(priors)
68
69   proposal[2*phs+1:(2*phs+1)+(phs-2)]=sort(proposal[2*phs+1:(2*phs+1)+(
70
71 pos=to_unit_ball.(proposal,priors)
72 box_bound!(pos,box)
73 θvec=to_prior.(pos,priors)
74
75 model = CMZ_Model(trajectory_no, 1, θvec, pos, [0.], obs,
76   constants ... ; v_init=true)
77
78 serialize(model_path, model) #save the model to the ensemble
79   directory
80 push!(ensemble_records, CMZ_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, CMZ_Record(trajectory_no, 1,
85   model.pos,model_path,model.log_Li))
86 end
87 end
88
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::CMZ_Model, e::CMZ_Ensemble; progress=false)
95   T=e.constants[1]
96
97   catpobs=vcat([e.obs[t][1] for t in 1:length(T)]...)
98   ymax=max(maximum(m.disp_mat[:, :, 1]), maximum(catpobs))
99   ymin=min(minimum(m.disp_mat[:, :, 1]), minimum(catpobs))
100
101   plt=lineplot(T,m.disp_mat[:,2,1],title="CMZ_Model
102   $(m.trajectory).$(m.i), log_Li $(m.log_Li)",color=:green,name="μ
103   pop", ylim=[ymin,ymax])
104   lineplot!(plt,T,m.disp_mat[:,1,1],color=:magenta,name="95% CI")

```

```

96     lineplot!(plt,T,m.disp_mat[:,3,1],color=:magenta)
97
98     Ts=vcat([[T[n] for i in 1:length(e.obs[n][1])] for n in
99             ↳ 1:length(T)]...)
100    scatterplot!(plt,Ts, catpobs, color=:yellow, name="Obs")
101
102    show(io, plt)
103    println()
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    (progress && return nrows(plt.graphics)+27);
124 end

```

---

### 16.5.3 /src/CMZ\_sim/CMZ\_lh.jl

---

```

1 const MAXVAL=prevfloat(Inf)
2
3 function CMZ_mc_llh(popdist::Distribution, voldist::Distribution,
4                     ↳ volconst::Float64, phase_ends::Vector{Float64},
5                     ↳ pparams::Vector{Float64}, mc_its::Int64, T::Vector{<:AbstractFloat},
6                     ↳ obs::Vector{<:Tuple{<:AbstractVector{<:Float64},<:AbstractVector{<:Float64}}})
7         times=length(T)
8         pends=floor.(phase_ends)
9         events=Int64.(sort(unique(vcat(T,pends))))

```

```

7   results=zeros(mc_its,length(events),2)
8
9   nt=Threads.nthreads()
10  Threads.@threads for t in 1:nt
11      t==nt ? (idxs=1+(t-1)*Int(floor(mc_its/nt)):mc_its) :
12          (idxs=(1+(t-1)*Int(floor(mc_its/nt))):t*Int(floor(mc_its/nt)))
13
14      phase=1
15      cycle_time,exit_rate=pparams[1+((phase-1)*2):2+((phase-1)*2)]
16
17      results[idxs,1,1]•=rand.(popdist)
18      results[idxs,1,2]•=rand.(voldist)
19
20      last_ph_ch=1;
21      next_event=2;
22      while next_event ≤ length(events)
23          n=events[next_event]-events[last_ph_ch]
24
25          pop_factor=max(eps(),(2^(24/cycle_time)-exit_rate))
26          pop_factor=1. && (pop_factor=pop_factor+eps())
27
28          → vol_factor=volconst*exit_rate*(1-pop_factor^(n-1))/(1-pop_factor)
29
30
31          → results[idxs,next_event,1]•=max.(min.(lastpops.*pop_factor^n,MAXVAL),
32
33          → results[idxs,next_event,2]•=min.(lastvols.+(lastpops.*vol_factor),MAXVAL)
34
35          if events[next_event] in pends #after updating pop and vol to
36              → t, update phase parameters for next event, make t。 vol。
37              → and pop。 the phase change vals
38              phase+=1; last_ph_ch=next_event
39              phase ≤ length(pparams)/2 &&
40                  ((cycle_time,exit_rate)=pparams[1+((phase-1)*2):2+((phase-1)*2)])
41      end #advance phase if necessary
42
43      next_event+=1
44  end
45 end

```

```

43     tidxs=[findfirst(t→t=time,events) for time in T]
44
45     pop_lns=Vector{LogNormal}(undef,times)
46     vol_lns=Vector{LogNormal}(undef,times)
47     pop_lns[1]=popdist
48     vol_lns[1]=voldist
49
50     Threads.@threads for t in 2:times
51         pop_lns[t]=fit(LogNormal,results[:,tidxs[t],1])
52         vol_lns[t]=fit(LogNormal,results[:,tidxs[t],2])
53     end
54
55     pop_lhs=Vector{Float64}(undef,times-1)
56     vol_lhs=Vector{Float64}(undef,times-1)
57
58     Threads.@threads for t in 1:times-1
59         pop_lhs[t]=lps(logpdf(pop_lns[t+1],obs[t+1][1]))
60         vol_lhs[t]=lps(logpdf(vol_lns[t+1],obs[t+1][2]))
61     end
62
63     log_lh=lps(lps(pop_lhs),lps(vol_lhs))
64
65     disp_mat=zeros(times,3,2)
66     Threads.@threads for t in 1:times
67
68         → disp_mat[t,:,:]=quantile(pop_lns[t],.025),mean(pop_lns[t]),quantile(pop_
69         → disp_mat[t,:,:]=quantile(vol_lns[t],.025),mean(vol_lns[t]),quantile(vol_
70     end
71
72     return log_lh, disp_mat
73 end

```

---

#### 16.5.4 /src/CMZ\_sim/CMZ\_model.jl

---

```

1 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 mutable 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
20      → output
21
22    function CMZ_Model(trajectory::Int64, i::Int64, θ::Vector{Float64},
23      → pos::Vector{Float64}, v::Vector{Float64},
24      → obs::Vector{Tuple{Vector{Float64},Vector{Float64}}}),
25      → T::Vector{Float64}, popdist::Distribution, voldist::Distribution,
26      → volconst::Float64, mc_its::Int64, 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=CMZ_mc_llh(popdist, voldist, volconst,
30        → phase_ends, pparams, mc_its, T, obs)
31
32      v_init && (v=rand(MvNormal(length(θ),1.)))
33
34      new(trajectory, i, θ, log_lh, pos, v, disp_mat)
35    end
36  end
37
38 construct_CMZ(args ... ;kwargs ... ) = CMZ_Model(args ... ;kwargs ... )

```

---

### 16.5.5 /src/slice\_pop\_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)

```

```

9     sliceno=t1c/lm.section
10
11    t2c=lm.factor*(t2^lm.power)
12    nc=t2c-t1c
13    psg=nc/sliceno
14    return (psg/lm.section).*pops
15 end
16
17 function Base.length(lm::Lens_Model)
18     return 1
19 end

```

---

### 16.5.6 /src/slice\_pop\_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_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
36 t_counter::Int64
37 end

38
39 Slice_Elbo(path::String, no_models::Integer,
40   ↵ obs ::AbstractVector{<:AbstractVector{<:Float64}}),
41   ↵ priors ::AbstractVector{<:Distribution}, constants, box, GMC_settings;
42   ↵ sample_posterior::Bool=true) =
43 Slice_Elbo(
44   path,
45   construct_slice,
46   assemble_SMs(path, no_models, obs, priors, constants, box) ... ,
47   [-Inf], #L₀ = 0
48     [0], #ie exp(0) = all of the prior is covered
49     [-Inf], #w₀ = 0
50     [-Inf], #Liwi₀ = 0
51     [-1e300], #Z₀ = 0
52     [0], #H₀ = 0,
53   obs,
54   priors,
55   constants, #T, popdist, lens_model, mc_its, phs
56   box,
57   sample_posterior,
58   Vector{Slice_Record}(),
59   GMC_settings ... ,
60   no_models+1)

61
62 function assemble_SMs(path::String, no_trajectories::Integer, obs,
63   ↵ priors, constants, box)
64   ensemble_records = Vector{Slice_Record}()
65   !isdir(path) && mkpath(path)
66   phs=constants[5]
67   @showprogress 1 "Assembling Slice_Model ensemble..." for
68     ↵ trajectory_no in 1:no_trajectories
69       model_path = string(path,'/',trajectory_no,'.',1)
70       if !isfile(model_path)
71         proposal=rand.(priors)

```

```

67         → proposal[2*phs+1:(2*phs+1)+(phs-2)]=sort(proposal[2*phs+1:(2*phs+1)+(
68
69             pos=to_unit_ball.(proposal,priors)
70             box_bound!(pos,box)
71             θvec=to_prior.(pos,priors)
72
73             model = Slice_Model(trajecotry_no, 1, θvec, pos, [0.], obs,
74             → constants ... ; v_init=true)
75
76             serialize(model_path, model) #save the model to the ensemble
77             → directory
78             push!(ensemble_records, Slice_Record(trajecotry_no, 1,
79             → pos,model_path,model.log_Li))
80         else #interrupted assembly pick up from where we left off
81             model = deserialize(model_path)
82             push!(ensemble_records, Slice_Record(trajecotry_no, 1,
83             → model.pos,model_path,model.log_Li))
84         end
85     end
86
87     return ensemble_records, minimum([record.log_Li for record in
88             → ensemble_records])
89 end
90
91
92
93
94
95
96
97
98
99
100

```

function Base.show(io::IO, m::Slice\_Model, e::Slice\_Engineering;

→ progress=false)

T=e.constants[1]

catpobs=vcat([e.obs[t] for t in 1:length(T)] ... )

ymax=max(maximum(m.disp\_mat[:, :]), maximum(catpobs))

ymin=min(minimum(m.disp\_mat[:, :]), minimum(catpobs))

plt=lineplot(T,m.disp\_mat[:, 2], title="Slice\_Model  
 → \$(m.trajectory).\$(m.i), log\_Li \$(m.log\_Li)", color=:green, name="μ  
 → pop", ylim=[ymin,ymax])

lineplot!(plt,T,m.disp\_mat[:, 1], color=:magenta, name="95% CI")

lineplot!(plt,T,m.disp\_mat[:, 3], color=:magenta)

Ts=vcat([[T[n] for i in 1:length(e.obs[n])] for n in 1:length(T)] ... )

scatterplot!(plt,Ts, catpobs, color=:yellow, name="Obs")

```

101   show(io, plt)
102   println()
103   println("θ: $(m.θ)")
104   println("v: $(m.v)")

105
106   (progress && return nrows(plt.graphics)+16);
107 end

```

---

### 16.5.7 /src/slice\_pop\_sim/slice\_lh.jl

```

1 const MAXVAL=prevfloat(Inf)

2
3 function slice_mc_llh(popdist::Distribution, lm::Lens_Model,
4   ↳ phase_ends::Vector{Float64}, pparams::Vector{Float64}, mc_its::Int64,
5   ↳ T::Vector{Float64}, obs::Vector{Vector{Float64}})
6   times=length(T)
7   pends=floor.(phase_ends)
8   events=Int64.(sort(unique(vcat(T,pends))))
9   results=zeros(mc_its,length(events))

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    next_event=2;
19    while next_event≤length(events)
20      n=events[next_event]-events[next_event-1]
21
22      pop_factor=max(eps(),(2^(24/cycle_time)-exit_rate))
23      pop_factor=1. && (pop_factor=pop_factor+eps())
24
25      lastpops=view(results,idxs,next_event-1)
26
27      results[idxs,next_event]≈min.(lastpops.*pop_factor^n,MAXVAL)
28

```

```

29         → results[idxs,next_event]=max.(view(results,idxs,next_event))-circumf
30
31     if events[next_event] in pends #after updating pop and vol to
32         → t, update phase parameters for next event, make t。vol.
33         → and pop。the phase change vals
34         phase+=1;
35         phase ≤ length(pparams)/2 &&
36             → ((cycle_time,exit_rate)=pparams[1+((phase-1)*2):2+((phase-1)*2)])
37     end #advance phase if necessary
38
39
40     next_event+=1
41 end
42
43 end
44
45 tidxs=[findfirst(t→t=time,events) for time in T]
46
47 pop_lns=Vector{LogNormal}(undef,times)
48 pop_lns[1]=popdist
49 Threads.@threads for t in 2:times
50     pop_lns[t]=fit(LogNormal,results[:,tidxs[t]])
51 end
52
53 pop_lhs=Vector{Float64}(undef,times-1)
54
55 Threads.@threads for t in 1:times-1
56     pop_lhs[t]=lps(logpdf(pop_lns[t+1],obs[t+1]))
57 end
58
59 log_lh=lps(pop_lhs)
60
61 disp_mat=zeros(times,3)
62 Threads.@threads for t in 1:times
63
64     → disp_mat[t,:]=[quantile(pop_lns[t],.025),mean(pop_lns[t]),quantile(pop_lns[t],.975)]
65 end
66
67 return log_lh, disp_mat
68 end

```

---

### 16.5.8 /src/slice\_pop\_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 mutable 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
22    function Slice_Model(trajectory::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        new(trajectory, i, θ, log_lh, pos, v, disp_mat)
35    end
36 end
37
38 construct_slice(args ... ;kwargs ... ) = Slice_Model(args ... ;kwargs ... )

```

---

### 16.5.9 /src/thymidine\_sim/thymidine\_cell.jl

---

```

1 abstract type AbstractCell end
2
3 mutable struct LabelCell <: AbstractCell
4     time::Float64
5     tδ::Float64
6     s::Float64
7     label::Float64
8 end

```

---

### 16.5.10 /src/thymidine\_sim/thymidine\_ensemble.jl

---

```

1 mutable struct Thymidine_El ensome <: GMC_NS_El ensome
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}
13    Hi::Vector{Float64}
14
15    obs::Vector{Vector{<:Integer}}
16    priors::Vector{<:Distribution}
17    constants::Vector{<:Any}
18    #T, pulse, mc_its, end_time
19    box::Matrix{Float64}
20
21    sample_posterior::Bool
22    posterior_samples::Vector{Thymidine_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
36     t_counter::Int64
37 end
38
39 Thymidine_Engsemble(path::String, no_models::Integer,
40   ↳ obs::AbstractVector{<:AbstractVector{<:Integer}}),
41   ↳ priors::AbstractVector{<:Distribution}, constants, box, GMC_settings;
42   ↳ sample_posterior::Bool=true) =
43 Thymidine_Engsemble(
44   path,
45   thymidine_constructor,
46   assemble_TMs(path, no_models, obs, priors, constants, box) ... ,
47   [-Inf], #L₀ = 0
48   [0], #ie exp(0) = all of the prior is covered
49   [-Inf], #w₀ = 0
50   [-Inf], #Liwi₀ = 0
51   [-1e300], #Z₀ = 0
52   [0], #H₀ = 0,
53   obs,
54   priors,
55   constants, #T, pulse, mc_its, end_time
56   box,
57   sample_posterior,
58   Vector{Thymidine_Record}(),
59   GMC_settings ... ,
60   no_models+1)
61
62 function assemble_TMs(path::String, no_trajectories::Integer, obs,
63   ↳ priors, constants, box)
64   ensemble_records = Vector{Thymidine_Record}()
65   !isdir(path) && mkpath(path)
66   @showprogress 1 "Assembling Thymidine_Model ensemble ..." for
67   ↳ trajectory_no in 1:no_trajectories
68       model_path = string(path,'/trajectory_no,'.',1)
69       if !isfile(model_path)
70           proposal=rand.(priors)
71
72           pos=to_unit_ball.(proposal,priors)

```

```

68         box_bound!(pos,box)
69         θvec=to_prior.(pos,priors)
70
71         model = Thymidine_Model(trajectory_no, 1, θvec, pos, [0.],
72             ↪ obs, constants ... ; v_init=true)
73
74             serialize(model_path, model) #save the model to
75             ↪ the ensemble directory
76             push!(ensemble_records,
77                 Thymidine_Record(trajectory_no, 1,
78                 ↪ pos,model_path,model.log_Li))
79
80     else #interrupted assembly pick up from where we left off
81         model = deserialize(model_path)
82         push!(ensemble_records,
83             Thymidine_Record(trajectory_no, 1,
84             ↪ model.pos,model_path,model.log_Li))
85
86     end
87
88
89     return ensemble_records, minimum([record.log_Li for record in
90             ↪ ensemble_records])
91 end
92
93
94 function Base.show(io::IO, m::Thymidine_Model, e::Thymidine_Ensemble;
95     ↪ progress=false)
96     T=e.constants[1]
97
98     catobs=vcat(e.obs ... )
99     ymax=max(maximum(m.disp_mat),maximum(catobs))
100    ymin=min(minimum(m.disp_mat),minimum(catobs))
101
102    plt=lineplot(T,m.disp_mat[:,2],title="Thymidine_Model
103        ↪ $(m.trajectory).$(m.i), log_Li $(m.log_Li)",color=:green,name="μ
104        ↪ count", ylim=[ymin,ymax])
105    lineplot!(plt,T,m.disp_mat[:,1],color=:magenta,name="95% CI")
106    lineplot!(plt,T,m.disp_mat[:,3],color=:magenta)
107
108    Ts=vcat([[T[n] for i in 1:length(e.obs[n])] for n in 1:length(T)] ... )
109    scatterplot!(plt,Ts, catobs, color=:yellow, name="Obs")
110
111    show(io, plt)
112    println()
113    println("θ: $(m.θ)")

```

```

101
102     (progress && return nrows(plt.graphics)+7);
103 end

```

---

### 16.5.11 /src/thymidine\_sim/thymidine\_lh.jl

```

1 function thymidine_mc_llh(pparams, mc_its, end_time, pulse,
2     ↳ T::Vector{<:AbstractFloat}, obs::Vector{<:AbstractVector{<:Integer}})
3     n_pops=length(pparams); times=length(T)
4     popset_labelled=zeros(Int64,mc_its,n_pops,times)
5
6     Threads.@threads for it in 1:mc_its
7         for p in 1:n_pops
8             pop_dist,tc_dist,r,s,sis_frac = pparams[p]
9             n_lineages=Int64(max(1,round(rand(pop_dist))))
10            plv=view(popset_labelled,it,p,:)
11
12            for l in 1:Int64(floor(n_lineages/2))
13                sim_lin_pair!(plv, T, end_time, pulse, tc_dist, r, s,
14                  ↳ sis_frac)
15            end
16
17            for l in 1:n_lineages%2
18                sim_lineage!(plv, T, end_time, pulse, tc_dist, r, s,
19                  ↳ sis_frac)
20            end
21        end
22    end
23
24    pop_DNPs=Matrix{DiscreteNonParametric}(undef,n_pops,times)
25    for p in 1:n_pops
26        Threads.@threads for t in 1:times
27
28            ↳ pop_DNPs[p,t]=fit(DiscreteNonParametric,popset_labelled[:,p,t])
29        end
30    end
31
32    if n_pops > 1
33        joints=Vector{DiscreteNonParametric}(undef,times)
34        Threads.@threads for t in 1:times
35            joints[t]=joint_DNP_sum(pop_DNPs[:,t])
36        end

```

```

33     else
34         joints=pop_DNPs[1,:]
35     end
36
37     log_lhs=Vector{Float64}(undef,times)
38     Threads.@threads for t in 1:times
39         log_lhs[t]=lps(logpdf(joints[t],obs[t]))
40     end
41
42     log_lh=lps(log_lhs)
43
44     disp_mat=zeros(times,3)
45     Threads.@threads for t in 1:times
46
47         → disp_mat[t,:]=[quantile(joints[t],.025),mean(joints[t]),quantile(joints[t],.975)]
48     end
49     return log_lh, disp_mat
50 end
51
52 function joint_DNP_sum(cs)
53     npops=length(cs)
54     ct_supports=Vector{Vector{Int64}}()
55     for c in cs
56         push!(ct_supports,c.support[[c.p ... ].>0.])
57     end
58
59     possible_sums=collect(Iterators.product(ct_supports ... ))
60     possible_totals=unique(sum.(possible_sums))
61     total_mat=sum.(possible_sums)
62
63     probs=zeros(length(possible_totals))
64
65     Threads.@threads for (n,pt) in collect(enumerate(possible_totals))
66         psums=possible_sums[findall(t→t==pt,total_mat)]
67         probs[n]=logsumexp([lps([logpdf(cs[p],ps[p]) for p in 1:npops])
68             → for ps in psums])
69     end
70
71     return DiscreteNonParametric(possible_totals, exp.(probs))
72 end
73

```

---

### 16.5.12 /src/thymidine\_sim/thymidine\_model.jl

```

1 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 mutable 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_Model(trajectory, i, θ, pos, v, obs, T, pulse, mc_its,
24   → end_time; v_init=false)
25   mod(length(θ),8)≠0 && throw(ArgumentError("θ must contain 8 values
26   → per lineage population!"))
27   pulse<0 && throw(ArgumentError("pulse length must be ≥ 0!"))
28
29   n_pops=length(θ)/8
30   pparams=Vector{Tuple{LogNormal, LogNormal, Float64, Normal,
31   → Float64}}()
32
33   for pop in 1:n_pops
34     lppμ, lpσ², r, tcμ, tcσ², sμ, σ², sis_frac=
35     → θ[Int64(1+((pop-1)*8)):Int64(8*pop)]
36     lpσ=sqrt(lpσ²); tcσ=sqrt(tcσ²); σ=sqrt(σ²)
37
38     → push!(pparams,(LogNormal(lppμ,lpσ),LogNormal(tcμ,tcσ),r,Normal(sμ,σ),sis_
39   end

```

```

34
35     log_lh,disp_mat=thymidine_mc_llh(pparams, mc_its, end_time, pulse, T,
36     ↳ obs)
37
38     v_init && (v=rand(MvNormal(length(θ),1.)))
39
40     Thymidine_Model(trajectory, i, θ, log_lh, pos, v, disp_mat)
41 end
42 thymidine_constructor(params ... ) = Thymidine_Model(params ... )

```

---

### 16.5.13 /src/thymidine\_sim/thymidine\_sim.jl

```

1 const DETECTION_THRESHOLD=.01
2
3 function sim_lin_pair!(plv, T, end_time, pulse, Tc, r, s, sis_frac)
4     tδ1, s1 = refractory_cycle_model(r, Tc, s)
5     tδ2, s2 = (1+rand([-1,1])*sis_frac).*tδ1, s1)
6
7     birth_time=rand(Uniform(-min(tδ1,tδ2),0.))
8
9     cells1=[LabelCell(birth_time,tδ1,s1,0.)]
10    cells2=[LabelCell(birth_time,tδ2,s2,0.)]
11
12    for cellvec in [cells1,cells2]
13        ci=1
14        while ci≤length(cellvec)
15            cycle_sim!(plv, ci, cellvec, T, end_time, pulse, Tc,r,s,
16            ↳ sis_frac)
17            ci+=1
18        end
19    end
20
21
22 function sim_lineage!(plv, T, end_time, pulse, Tc, r, s, sis_frac)
23     tδ1, s1 = refractory_cycle_model(r, Tc, s)
24     birth_time=rand(Uniform(-tδ1,0.))
25     cells=[LabelCell(birth_time,tδ1,s1,0.)]
26
27     ci=1
28     while ci≤ length(cells)

```

```

29         cycle_sim!(plv, ci, cells, T, end_time, pulse, Tc, r, s,
30             ↵ sis_frac)
31         ci+=1
32     end
33 end

34 function cycle_sim!(plv, ci::Int64, cells::Vector{LabelCell},
35     ↵ T::Vector{Float64}, end_time::Float64, pulse::Float64, Tc::LogNormal,
36     ↵ r::Float64, sd::Normal, sis_frac::Float64)
37     n_times=length(T);

38     cell=cells[ci]
39     t=cell.time; tδ[] = cell.tδ[]; s[] = cell.s[]; l=cell.label

40     first_cycle=true
41     while t<end_time
42         tδ,s=0.,0.
43         first_cycle ? (tδ=tδ[];s=s[];first_cycle=false) : ((tδ,
44             ↵ s)=refractory_cycle_model(r, Tc, sd))

45         s_start=t+r
46         s_end=s_start+s
47         cycle_mitosis=t+tδ
48         cycle_events=[s_start,s_end,cycle_mitosis]

49         oldl=l; cycle_label=false
50         if (s_start ≤ pulse && s_end ≥ 0) #some s-phase overlaps with
51             ↵ the pulse in this case
52             cycle_label=true
53             l=update_label(l, s, s_start, s_end, pulse)
54         end

55         tidxs=t.<T.<cycle_mitosis
56         n_idxssum(tidxs)
57         (length(tidxs)>0 && (oldl > DETECTION_THRESHOLD || l >
58             ↵ DETECTION_THRESHOLD))

59         if n_idxss>0 && (oldl > DETECTION_THRESHOLD || l >
60             ↵ DETECTION_THRESHOLD)
61             cycle_label ? (label_outcomes=[oldl,-Inf,l]) :
62                 ↵ (label_outcomes=[oldl,oldl,oldl])
63             cubuf=false(n_idxss)
64             for (n, timept) in enumerate(T[tidxs])

```

```

64         count_labelled_at_T!(n,cubuf,timept, cycle_events,
65             ↳ label_outcomes, oldl, s_start, s, pulse)
66     end
67     plv[tidxs]+=cubuf
68 end
69
70 l/=2 #mitosis halves label for the daughters
71
72 tδ2, s2 = (1+rand([-1,1])*sis_frac).*(tδ, s)
73
74 cycle_mitosis < end_time &&
75     ↳ push!(cells,LabelCell(cycle_mitosis,tδ2,s2,l))
76     t+=tδ
77 end
78 end
79
80 function refractory_cycle_model(refractory, Tc, s_dist)
81     cycle_time=rand(Tc)
82     s=max(eps(),rand(s_dist))
83     tδ=max(refractory + s, cycle_time)
84     return tδ, s
85 end
86
87 function update_label(label, s, s_start, s_end, pulse)
88     s_overlap=min(pulse,s_start+s_end)-max(0,s_start)
89     nuc_label_frac=s_overlap/s
90     return min(nuc_label_frac+label,1.) #label before mitosis time
91         ↳ will be what's been accumulated from past + this cycle's s
92         ↳ phase
93 end
94
95 function count_labelled_at_T!(n, cubuf, timept, cycle_events,
96     ↳ label_outcomes, oldl, s_start, s, pulse)
97     outcome_idx=timept.<cycle_events
98     timept_label=label_outcomes[outcome_idx][1]
99     timept_label=-Inf &&
100        ↳ (timept_label=partial_label(timept,oldl,s_start,s,pulse))
101     timept_label > DETECTION_THRESHOLD && (cubuf[n]=true)
102 end
103
104 function partial_label(timept, label, s_start, s, pulse)
105     s_overlap=min(pulse,timept)-max(0.,s_start)
106     nuc_label_frac=s_overlap/s

```

```

101     return min(nuc_label_frac+label,1.)
102 end

```

---

## 16.6 BioBackgroundModels

### 16.6.1 /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

```

---

### 16.6.2 /src/API/EM\_master.jl

```

1 #function to setup an BHMM chains dictionary and RemoteChannel for
  ↵ learning jobs, given a vector of state #s, order_nos, replicates to
  ↵ train, the dictionary to fill, the RemoteChannel and the training
  ↵ sequences
2 #resumes any existing non-converged chains, otherwise initialises hmms
  ↵ for new chains given provided constants
3 function setup_EM_jobs!(job_ids::Vector{Chain_ID},
  ↵ obs_sets::Dict{String,Vector{LongSequence{DNAAlphabet{2}}}};
  ↵ delta_thresh::Float64=1e-3,
  ↵ chains::Dict{Chain_ID,Vector{EM_step}}=Dict{Chain_ID,Vector{EM_step}}(),
  ↵ init_function::Function=autotransition_init)
  ↵ #argument checking
  ↵ length(job_ids) < 1 && throw(ArgumentError("Empty job id vector!"))
  ↵ length(obs_sets) < 1 && throw(ArgumentError("Empty observation
    ↵ sets!"))
7
8   no_input_hmms = length(job_ids)
9   input_channel= RemoteChannel(()→Channel{Tuple}(no_input_hmms*3))
  ↵ #channel to hold BHMM learning jobs
10  output_channel= RemoteChannel(()→Channel{Tuple}(Inf)) #channel to
    ↵ take EM iterates off of

```

```

12   code_dict = code_job_obs(job_ids, obs_sets)
13
14   @showprogress 1 "Setting up HMMs ... " for id in job_ids #for each
15     → jobid, add an initial BHMM to input_channel for EM_workers
16     if haskey(chains, id) && length(chains[id]) > 0 #true if resuming
17       → from incomplete chain
18       chain_end=chains[id][end]
19       if !chain_end.converged || (chain_end.converged &&
20         → chain_end.delta > delta_thresh)#push the last hmm iterate
21         → for nonconverged chains to the input channel with coded
22         → observations and values for chain resumption
23         put!(input_channel, (id, chain_end.iterate,
24           → chain_end.hmm, chain_end.log_p, code_dict[(id.obs_id,
25             → id.order)]))
26       else #skip any jobs that have converged from previous runs
27         no_input_hmms -= 1
28     end
29
30   else
31     hmm = init_function(id.K, id.order, id.obs_id) #initialise
32       → first BHMM in chain
33     chains[id] = Vector{EM_step}() #initialise the relevant chain
34     obs=code_dict[(id.obs_id,id.order)]
35     lh=obs_lh_given_hmm(obs,hmm,linear=false)
36     put!(input_channel, (id, 1, hmm, lh, obs))
37   end
38 end
39
40 return no_input_hmms, chains, input_channel, output_channel
41 end
42
43 function execute_EM_jobs!(worker_pool::Vector{Int64},
44   → no_input_hmms::Integer, chains::Dict{Chain_ID,Vector{EM_step}},
45   → input_channel::RemoteChannel, output_channel::RemoteChannel,
46   → chains_path::String; load_dict=Dict{Int64,LoadConfig}(),
47   → EM_func::Function=linear_step, delta_thresh=1e-3, max_iterates=5000,
48   → verbose=false)
49   #argument checking
50   length(worker_pool) < 1 && throw(ArgumentError("Worker pool must
51     → contain one or more worker IDs!"))
52   no_input_hmms < 1 && throw(ArgumentError("Zero input HMMs reported,
53     → likely continuing from chains already converged beyond default
54     → delta_thresh for setup_EM_jobs"))

```

```

38     length(chains) < 1 && throw(ArgumentError("No chains supplied, likely
  ↵   job set from setup_EM_jobs passed incorrectly"))
39     !isready(input_channel) && throw(ArgumentError("BHMM input channel
  ↵   has no contents, likely job set from setup_EM_jobs already
  ↵   executed"))
40
41 #SEND BHMM FIT JOBS TO WORKERS
42 if isready(input_channel) > 0
43     @info "Fitting HMMs.."
44     #WORKERS FIT HMMS
45     for worker in worker_pool
46         if worker in keys(load_dict)
47             remote_do(EM_converge!, worker, input_channel,
  ↵             output_channel, no_input_hmms,
  ↵             load_config=load_dict[worker], EM_func=EM_func,
  ↵             delta_thresh=delta_thresh, max_iterates=max_iterates,
  ↵             verbose=verbose)
48         else
49             remote_do(EM_converge!, worker, input_channel,
  ↵             output_channel, no_input_hmms, EM_func=EM_func,
  ↵             delta_thresh=delta_thresh, max_iterates=max_iterates,
  ↵             verbose=verbose)
50     end
51 end
52 else
53     @warn "No input HMMs (all already converged?), skipping
  ↵   fitting.."
54 end
55
56 #GET LEARNT HMMS OFF REMOTECHANNEL, SERIALISE AT EVERY ITERATION,
  ↵ UPDATE PROGRESS METERS
57 job_counter=no_input_hmms
58 learning_meters=Dict{Chain_ID, ProgressHMM}()
59 overall_meter=Progress(no_input_hmms,"Overall batch progress:")
60
61 while job_counter > 0
62     wait(output_channel)
63     workerid, jobid, iterate, hmm, log_p, delta, converged, steptime
  ↵     = take!(output_channel)
64     #either update an existing ProgressHMM meter or create a new one
  ↵     for the job
65     if haskey(learning_meters, jobid) && iterate > 2
66         update!(learning_meters[jobid], delta, steptime)

```

```

67     else
68         offset = workerid - 1
69         if iterate ≤ 2
70             learning_meters[jobid] = ProgressHMM(delta_thresh,
71                 ↳ "$jobid on Wk $workerid:", offset, 2)
72             update!(learning_meters[jobid], delta, steptime)
73         else
74             learning_meters[jobid] = ProgressHMM(delta_thresh,
75                 ↳ "$jobid on Wk $workerid:", offset, iterate)
76             update!(learning_meters[jobid], delta, steptime)
77         end
78     end
79     #push the hmm and related params to the results_dict
80     push!(chains[jobid], EM_step(iterate, hmm, log_p, delta,
81         ↳ converged))
82     #try to serialize the results; catch interrupts and other errors
83     ↳ to prevent corruption
84     try
85         serialize(chains_path, chains)
86     catch e
87         @warn "Serializing failed!"
88         println(e)
89     end
90
91     #decrement the job counter, update overall progress meter, and
92     ↳ save the current results dict on convergence or max iterate
93     if converged || iterate == max_iterates
94         job_counter -= 1
95         ProgressMeter.update!(overall_meter,
96             ↳ (no_input_hmms-job_counter))
97         if !isready(input_channel) #if there are no more jobs to be
98             learnt, retire the worker
99             workerid!=1 && rmprocs(workerid)
100        end
101    end
102
103    #count converged & unconverged jobs, report results
104    converged_counter = 0
105    unconverged_counter = 0
106    for (id, chain) in chains
107        chain[end].converged = true ? (converged_counter += 1) :
108            (unconverged_counter += 1)

```

```

102     end
103
104     @info "Background HMM batch learning task complete,
105           ↳ $converged_counter converged jobs, $unconverged_counter jobs
106           ↳ failed to converge in $max_iterates iterates since job start."
107   end

```

---

### 16.6.3 /src/API/genome\_sampling.jl

```

1 #function to partition genome and set up Distributed RemoteChannels so
2   ↳ partitions can be sampled simultaneously
3 function setup_sample_jobs(genome_path::String,
4   ↳ genome_index_path::String, gff3_path::String,
5   ↳ sample_set_length::Integer, sample_window_min::Integer,
6   ↳ sample_window_max::Integer, perigenic_pad::Integer;
7   ↳ deterministic::Bool=false)
8   #argument checking
9   !ispather(genome_path) && throw(ArgumentError("Bad genome path!"))
10  !ispather(genome_index_path) && throw(ArgumentError("Bad genome index
11    ↳ path!"))
12  !ispather(gff3_path) && throw(ArgumentError("Bad gff3 path!"))
13  sample_set_length < 1 && throw(ArgumentError("Sample set length must
14    ↳ be a positive integer!"))
15  sample_window_min < 1 || sample_window_max < 1 &&
16    ↳ throw(ArgumentError("Sample window minimum and maximum bounds
17    ↳ must be positive integers!"))
18  sample_window_min ≥ sample_window_max && throw(ArgumentError("Sample
19    ↳ window minimum size must be smaller than maximum size"))
20  perigenic_pad < 0 && throw(ArgumentError("Perigenic pad must be 0 or
21    ↳ positive!"))

22  coordinate_partitions = partition_genome_coordinates(gff3_path,
23    ↳ perigenic_pad)
24  sample_sets = DataFrame[]
25  input_sample_jobs =
26    ↳ RemoteChannel(()→Channel{Tuple}(length(coordinate_partitions)))
27    ↳ #channel to hold sampling jobs
28  completed_sample_jobs =
29    ↳ RemoteChannel(()→Channel{Tuple}(length(coordinate_partitions)))
30    ↳ #channel to hold completed sample dfs
31  for (partitionid, partition) in coordinate_partitions
32    add_metacoordinates!(partition)

```

```

18     put!(input_sample_jobs, (genome_path, genome_index_path,
19         ← partition, partitionid, sample_set_length, sample_window_min,
20         ← sample_window_max, deterministic))
21   end
22   progress_channel = RemoteChannel(()→Channel{Tuple}(20))
23   return (input_sample_jobs, completed_sample_jobs, progress_channel,
24           ← sample_set_length)
25 end
26
27 function
28   ← execute_sample_jobs(channels::Tuple{RemoteChannel,RemoteChannel,RemoteChannel,Int}
29   ← worker_pool::Vector{Int64}; partitions::Integer=3)
30   input_sample_channel, completed_sample_channel, progress_channel,
31   ← sample_set_length = channels
32
33   #argument checking
34   length(worker_pool) < 1 && throw(ArgumentError("Worker pool must
35   ← contain one or more worker IDs!"))
36   !isready(input_sample_channel) && throw(ArgumentError("Input sample
37   ← channel not ready, likely channel set from setup_sample_jobs
38   ← passed incorrectly"))
39   sample_set_length < 1 && throw(ArgumentError("Sample set length must
40   ← be a positive integer, likely channel set from setup_sample_jobs
41   ← passed incorrectly"))
42
43   #send sampling jobs to workers
44   if isready(input_sample_channel) > 0
45     @info "Sampling.."
46     #WORKERS SAMPLE
47     for (n, worker) in enumerate(worker_pool)
48       if n ≤ partitions #no more workers than partitions to be
49         ← used
50         remote_do(get_sample_set, worker, input_sample_channel,
51         ← completed_sample_channel, progress_channel)
52       end
53     end
54   else
55     @error "No sampling jobs!"
56   end
57
58   #progress meters for sampling
59   sampling_meters=Dict{String, Progress}()

```

```
47     overall_sampling_meter=Progress(partitions,"Overall sampling
48         ↪ progress:")
49     completed_counter = 0
50     ProgressMeter.update!(overall_sampling_meter, completed_counter)
51     sampling_offset = ones(Bool, partitions)
52
53     #collect progress updates while waiting on completion of sampling
54         ↪ jobs
55     while completed_counter < partitions
56         wait(progress_channel)
57         partition_id, progress = take!(progress_channel)
58         if haskey(sampling_meters, partition_id)
59             ProgressMeter.update!(sampling_meters[partition_id],
60                 ↪ progress)
61         else
62             offset = findfirst(sampling_offset)[1]
63             sampling_meters[partition_id] = Progress(sample_set_length,
64                 ↪ "Sampling partition $partition_id:", offset)
65             ProgressMeter.update!(sampling_meters[partition_id],
66                 ↪ progress)
67             sampling_offset[offset] = false
68         end
69     if progress == sample_set_length
70         completed_counter += 1
71         ProgressMeter.update!(overall_sampling_meter,
72             ↪ completed_counter)
73     end
74
75     #collect sample dfs by partition id when ready
76     sample_record_dfs = Dict{String,DataFrame}()
77     collected_counter = 0
78     while collected_counter < partitions
79         wait(completed_sample_channel)
80         partition_id, sample_df = take!(completed_sample_channel)
81         sample_record_dfs[partition_id] = sample_df
82         collected_counter += 1
83         @info "Partition $partition_id completed sampling..."
84     end
85
86     return sample_record_dfs
87 end
```

### 16.6.4 /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}}}})
10    #check chains dict for problems
11
12    length(chains)==0 && throw(ArgumentError("Empty chains dict!"))
13    any(chain→length(chain)<2, values(chains)) &&
14        ↳ throw(ArgumentError("Some chains are too short (<2 EM steps)!")
15        ↳ "Probably not all chains have been operated on by EM workers yet.
16        ↳ Try EM_converge first!"))
17    unconv=sum(!chain[end].converged for chain in values(chains))
18    unconv > 0 && @warn "Not all chains are converged to the selected
19        ↳ step delta."
20    length(test_sets)==0 && throw(ArgumentError("Empty test_sets dict!"))
21
22    chain_reports = report_chains(chains, test_sets)
23    partition_reports = report_partitions(chain_reports)
24
25    report_folders=Dict{String, Report_Folder}()
26
27    for part_report in partition_reports
28        rep_report=report_replicates(part_report.best_repset, chains)
29        chain_subset=Dict{Chain_ID,Chain_Report}()
30        for id in keys(chain_reports)
31            id.obs_id=part_report.partition_id &&
32                ↳ (chain_subset[id]=chain_reports[id])
33        end
34
35        ↳ report_folders[part_report.partition_id]=Report_Folder(part_report.partition_id,
36        ↳ part_report, rep_report, chain_subset)
37    end
38
39    return report_folders
40 end

```

---

### 16.6.5 /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 ````julia
14 hmm = BHMM([0.9 0.1; 0.1 0.9], [Normal(0,1), Normal(10,1)])
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
26 """
27     assert_BHMM(a, A, B)
28 Throw an `ArgumentError` if the initial state distribution and the
29         ↳ transition matrix rows does not sum to 1,

```

```

30 and if the observations distributions does not have the same dimensions.
31 """
32 function assert_BHMM(a::AbstractVector,
33                         A::AbstractMatrix,
34                         B::AbstractVector{<:Categorical})
35     !isprobvec(a) && throw(ArgumentError("Initial state vector a is not a
36                             ↳ valid probability vector!"))
37     !istransmat(A) && throw(ArgumentError("Transition matrix A is not
38                             ↳ valid!"))
39     !all([length(d.p) for d in B] .== length(B[1].p)) &&
40                             ↳ throw(ArgumentError("All distributions must have the same
41                             ↳ dimensions"))
42     !(length(a) == size(A,1) == length(B)) && throw(ArgumentError("Length
43                             ↳ of initial state vector a, dimension of transition matrix A, and
44                             ↳ number of distributions B are not the same!"))
45     return true
46 end
47
48 function Base.show(io::IO, hmm::BHMM)
49     println(io, "Background BHMM")
50     println(io, "State Initial and Transition Probabilities")
51     print(io, "a: ")
52     show(io, hmm.a)
53     println(io)
54     print(io, "A: ")
55     display(hmm.A)
56     println(io)
57     println(io, "INFORMATIVE SYMBOLS BY STATE")
58     for (n,d) in enumerate(hmm.B)
59         print(io, "K$n ")
60         print_emitters(d)
61     end
62 end

```

---

### 16.6.6 /src/EM/EM\_converge.jl

```

26     for i in curr_iterate:max_iterates
27         start=time()
28         new_hmm, norm = EM_func(new_hmm, observations, obs_lengths)
29         curr_iterate += 1
30         delta = abs(lps(norm, -last_norm))
31         if delta < delta_thresh
32             put!(output_hmms, (workerid, jobid, curr_iterate,
33             ↪ new_hmm, norm, delta, true, time()-start))
34             verbose && @info "$jobid converged after
35             ↪ $(curr_iterate-1) EM steps"
36             break
37         else
38             put!(output_hmms, (workerid, jobid, curr_iterate,
39             ↪ new_hmm, norm, delta, false, time()-start))
40             verbose && @info "$jobid EM step $(curr_iterate-1) delta
41             ↪ $delta"
42             last_norm = norm
43         end
44     end
45 end
46

```

---

### 16.6.7 /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 """
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
  ↵ the longest T
19
20 #transforms to cut down log_ξ, log_γ assignment times
21 lls = permutedims(lls, [2,3,1]) # from (K,T,O) to (T,O,K)
22 log_α = permutedims(log_α, [2,3,1])
23 log_β = permutedims(log_β, [2,3,1])
24
25 # E-step
26 log_ξ = fill(-Inf, Tmaxplus1,O,K,K)
27 log_γ = fill(-Inf, Tmaxplus1,O,K)
28 log_pobs = zeros(O)
29
30 @inbounds for o = 1:O
31     log_pobs[o] = logsumexp(lps.(log_α[1,o,:], log_β[1,o,:]))
32 end
33
34 Threads.@threads for idx in [(i,j,o) for i=1:K, j=1:K, o=1:O]
35     i,j,o = idx[1],idx[2],idx[3]
36     obsl = obs_lengths[o]
37     @inbounds for t = 1:obsl-1 #log_ξ & log_γ calculated to T-1 for
      ↵ each o
38         log_ξ[t,o,i,j] = lps(log_α[t,o,i], log_A[i,j], log_β[t+1,o,j],
          ↵ lls[t+1,o,j], -log_pobs[o])
39         log_γ[t,o,i] = lps(log_α[t,o,i], log_β[t,o,i], -log_pobs[o])
40     end
41     t=obsl #log_ξ @ T = 0
42     log_ξ[t,o,i,j] = 0
43     log_γ[t,o,i] = lps(log_α[t,o,i], log_β[t,o,i], -log_pobs[o])
44 end
45
46 ξ = similar(log_ξ)
47 ξ .= exp.(log_ξ)
48 Σk_ξ = sum(ξ, dims=[3,4])
49 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
93 end
94
95 #Multisequence competent log implementations of forward
# and backwards algos
96 function messages_forwards_log(init_distn, trans_matrix,
97     → log_likelihoods, obs_lengths)
98     log_alphas = zeros(size(log_likelihoods))
99     log_trans_matrix = log.(trans_matrix)

```

```

96     log_alphas[:,1,:] = log.(init_distn) .+
97     ↳ log_likelihoods[:,1,:]
98 Threads.@threads for o in 1:size(log_likelihoods)[3]
99     @inbounds for t in 2:obs_lengths[o], j in
100        ↳ 1:size(log_likelihoods)[1]
101
102         ↳ log_alphas[j,t,o]=logsumexp(log_alphas[:,t-1,o]
103         ↳ .+ log_trans_matrix[:,j]) +
104         ↳ log_likelihoods[j,t,o]
105     end
106 end
107     return log_alphas
108 end
109
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
131 function t_categorical_mle(::Type{<:Categorical},
132     ↳ k::Integer, x::AbstractArray{T}, w::AbstractArray{F})
133     ↳ where T<:Integer where F<:AbstractFloat
134
135         ↳ Categorical(t_pnormalize!(t_add_categorical_counts!(zeros(k),
136         ↳ x, w)))
137
138     end
139
140 t_pnormalize!(v::Vector) = (v ./= sum(v); v)
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
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
617
618
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
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
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
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
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
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
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
987
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
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
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
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
1147
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1167
1168
1169
1170
1171
1172
1173
1174
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1186
1187
1188
1189
1189
1190
1191
1192
1193
1193
1194
1195
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1214
1215
1216
1217
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1237
1237
1238
1239
1239
1240
1241
1242
1243
1243
1244
1245
1245
1246
1247
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1265
1266
1267
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1277
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1293
1294
1295
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1305
1306
1307
1307
1308
1309
1309
1310
1311
1312
1313
1314
1314
1315
1316
1316
1317
1318
1318
1319
1320
1320
1321
1322
1322
1323
1324
1324
1325
1326
1326
1327
1328
1328
1329
1330
1330
1331
1332
1332
1333
1334
1334
1335
1336
1336
1337
1338
1338
1339
1340
1340
1341
1342
1342
1343
1344
1344
1345
1346
1346
1347
1348
1348
1349
1350
1350
1351
1352
1352
1353
1354
1354
1355
1356
1356
1357
1358
1358
1359
1360
1360
1361
1362
1362
1363
1364
1364
1365
1366
1366
1367
1368
1368
1369
1370
1370
1371
1372
1372
1373
1374
1374
1375
1376
1376
1377
1378
1378
1379
1380
1380
1381
1382
1382
1383
1384
1384
1385
1386
1386
1387
1388
1388
1389
1390
1390
1391
1392
1392
1393
1394
1394
1395
1396
1396
1397
1398
1398
1399
1400
1400
1401
1402
1402
1403
1404
1404
1405
1406
1406
1407
1408
1408
1409
1410
1410
1411
1412
1412
1413
1414
1414
1415
1416
1416
1417
1418
1418
1419
1420
1420
1421
1422
1422
1423
1424
1424
1425
1426
1426
1427
1428
1428
1429
1430
1430
1431
1432
1432
1433
1434
1434
1435
1436
1436
1437
1438
1438
1439
1440
1440
1441
1442
1442
1443
1444
1444
1445
1446
1446
1447
1448
1448
1449
1450
1450
1451
1452
1452
1453
1454
1454
1455
1456
1456
1457
1458
1458
1459
1460
1460
1461
1462
1462
1463
1464
1464
1465
1466
1466
1467
1468
1468
1469
1470
1470
1471
1472
1472
1473
1474
1474
1475
1476
1476
1477
1478
1478
1479
1480
1480
1481
1482
1482
1483
1484
1484
1485
1486
1486
1487
1488
1488
1489
1490
1490
1491
1492
1492
1493
1494
1494
1495
1496
1496
1497
1498
1498
1499
1500
1500
1501
1502
1502
1503
1504
1504
1505
1506
1506
1507
1508
1508
1509
1510
1510
1511
1512
1512
1513
1514
1514
1515
1516
1516
1517
1518
1518
1519
1520
1520
1521
1522
1522
1523
1524
1524
1525
1526
1526
1527
1528
1528
1529
1530
1530
1531
1532
1532
1533
1534
1534
1535
1536
1536
1537
1538
1538
1539
1540
1540
1541
1542
1542
1543
1544
1544
1545
1546
1546
1547
1548
1548
1549
1550
1550
1551
1552
1552
1553
1554
1554
1555
1556
1556
1557
1558
1558
1559
1560
1560
1561
1562
1562
1563
1564
1564
1565
1566
1566
1567
1568
1568
1569
1570
1570
1571
1572
1572
1573
1574
1574
1575
1576
1576
1577
1578
1578
1579
1580
1580
1581
1582
1582
1583
1584
1584
1585
1586
1586
1587
1588
1588
1589
1590
1590
1591
1592
1592
1593
1594
1594
1595
1596
1596
1597
1598
1598
1599
1600
1600
1601
1602
1602
1603
1604
1604
1605
1606
1606
1607
1608
1608
1609
1610
1610
1611
1612
1612
1613
1614
1614
1615
1616
1616
1617
1618
1618
1619
1620
1620
1621
1622
1622
1623
1624
1624
1625
1626
1626
1627
1628
1628
1629
1630
1630
1631
1632
1632
1633
1634
1634
1635
1636
1636
1637
1638
1638
1639
1640
1640
1641
1642
1642
1643
1644
1644
1645
1646
1646
1647
1648
1648
1649
1650
1650
1651
1652
1652
1653
1654
1654
1655
1656
1656
1657
1658
1658
1659
1660
1660
1661
1662
1662
1663
1664
1664
1665
1666
1666
1667
1668
1668
1669
1670
1670
1671
1672
1672
1673
1674
1674
1675
1676
1676
1677
1678
1678
1679
1680
1680
1681
1682
1682
1683
1684
1684
1685
1686
1686
1687
1688
1688
1689
1690
1690
1691
1692
1692
1693
1694
1694
1695
1696
1696
1697
1698
1698
1699
1700
1700
1701
1702
1702
1703
1704
1704
1705
1706
1706
1707
1708
1708
1709
1710
1710
1711
1712
1712
1713
1714
1714
1715
1716
1716
1717
1718
1718
1719
1720
1720
1721
1722
1722
1723
1724
1724
1725
1726
1726
1727
1728
1728
1729
1730
1730
1731
1732
1732
1733
1734
1734
1735
1736
1736
1737
1738
1738
1739
1740
1740
1741
1742
1742
1743
1744
1744
1745
1746
1746
1747
1748
1748
1749
1750
1750
1751
1752
1752
1753
1754
1754
1755
1756
1756
1757
1758
1758
1759
1760
1760
1761
1762
1762
1763
1764
1764
1765
1766
1766
1767
1768
1768
1769
1770
1770
1771
1772
1772
1773
1774
1774
1775
1776
1776
1777
1778
1778
1779
1780
1780
1781
1782
1782
1783
1784
1784
1785
1786
1786
1787
1788
1788
1789
1790
1790
1791
1792
1792
1793
1794
1794
1795
1796
1796
1797
1798
1798
1799
1800
1800
1801
1802
1802
1803
1804
1804
1805
1806
1806
1807
1808
1808
1809
1810
1810
1811
1812
1812
1813
1814
1814
1815
1816
1816
1817
1818
1818
1819
1820
1820
1821
1822
1822
1823
1824
1824
1825
1826
1826
1827
1828
1828
1829
1830
1830
1831
1832
1832
1833
1834
1834
1835
1836
1836
1837
1838
1838
1839
1840
1840
1841
1842
1842
1843
1844
1844
1845
1846
1846
1847
1848
1848
1849
1850
1850
1851
1852
1852
1853
1854
1854
1855
1856
1856
1857
1858
1858
1859
1860
1860
1861
1862
1862
1863
1864
1864
1865
1866
1866
1867
1868
1868
1869
1870
1870
1871
1872
1872
1873
1874
1874
1875
1876
1876
1877
1878
1878
1879
1880
1880
1881
1882
1882
1883
1884
1884
1885
1886
1886
1887
1888
1888
1889
1890
1890
1891
1892
1892
1893
1894
1894
1895
1896
1896
1897
1898
1898
1899
1900
1900
1901
1902
1902
1903
1904
1904
1905
1906
1906
1907
1908
1908
1909
1910
1910
1911
1912
1912
1913
1914
1914
1915
1916
1916
1917
1918
1918
1919
1920
1920
1921
1922
1922
1923
1924
1924
1925
1926
1926
1927
1928
1928
1929
1930
1930
1931
1932
1932
1933
1934
1934
1935
1936
1936
1937
1938
1938
1939
1940
1940
1941
1942
1942
1943
1944
1944
1945
1946
1946
1947
1948
1948
1949
1950
1950
1951
1952
1952
1953
1954
1954
1955
1956
1956
1957
1958
1958
1959
1960
1960
1961
1962
1962
1963
1964
1964
1965
1966
1966
1967
1968
1968
1969
1970
1970
1971
1972
1972
1973
1974
1974
1975
1976
1976
1977
1978
1978
1979
1980
1980
1981
1982
1982
1983
1984
1984
1985
1986
1986
1987
1988
1988
1989
1990
1990
1991
1992
1992
1993
1994
1994
1995
1996
1996
1997
1998
1998
1999
2000
2000
2001
2002
2002
2003
2004
2004
2005
2006
2006
2007
2008
2008
2009
2010
2010
2011
2012
2012
2013
2014
2014
2015
2016
2016
2017
2018
2018
2019
2020
2020
2021
2022
2022
2023
2024
2024
2025
2026
2026
2027
2028
2028
2029
2030
2030
2031
2032
2032
2033
2034
2034
2035
2036
2036
2037
2038
2038
2039
2040
2040
2041
2042
2042
2043
2044
2044
2045
2046
2046
2047
2048
2048
2049
2050
2050
2051
2052
2052
2053
2054
2054
2055
2056
2056
2057
2058
2058
2059
2060
2060
2061
2062
2062
2063
2064
2064
2065
2066
2066
2067
2068
2068
2069
2070
2070
2071
2072
2072
2073
2074
2074
2075
2076
2076
2077
2078
2078
2079
2080
2080
2081
2082
2082
2083
2084
2084
2085
2086
2086
2087
2088
2088
2089
2090
2090
2091
2092
2092
2093
2094
2094
2095
2096
2096
2097
2098
2098
2099
2100
2100
2101
2102
2102
2103
2104
2104
2105
2106
2106
2107
2108
2108
2109
2110
2110
2111
2112
2112
2113
2114
2114
2115
2116
2116
2117
2118
2118
2119
2120
2120
2121
2122
2122
2123
2124
2124
2125
2126
2126
2127
2128
2128
2129
2130
2130
2131
2132
2132
2133
2134
2134
2135
2136
2136
2137
2138
2138
2139
2140
2140
2141
2142
2142
2143
2144
2144
2145
2146
2146
2147
2148
2148
2149
2150
2150
2151
2152
2152
2153
2154
```

```

125     function t_add_categorical_counts!(h::Vector{F},
126         x::AbstractArray{T}, w::AbstractArray{F}) where
127             T<:Integer where F<:AbstractFloat
128                 n = length(x)
129                 if n ≠ length(w)
130                     throw(DimensionMismatch("Inconsistent array
131                         → lengths."))
132                 end
133                 hlock = ReentrantLock()
134                 Threads.@threads for i=1:n
135                     @inbounds xi = x[i]
136                     @inbounds wi = w[i]
137                     lock(hlock)
138                     h[xi] += wi    # cannot use @inbounds, as no
139                         → guarantee that x[i] is in bound
140                     unlock(hlock)
141                 end
142                 return h
143             end
144         end

```

---

### 16.6.8 /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     order < 0 && throw(ArgumentError("Chain_ID symbol order must be zero
15         → or a positive integer!"))
16     replicate < 1 && throw(ArgumentError("Chain_ID replicate must be a
17         → positive integer!"))
18     return true
19 end
20
21

```

```

16 struct EM_step
17     iterate::Integer
18     hmm::BHMM
19     log_p::AbstractFloat
20     delta::AbstractFloat
21     converged::Bool
22
23     → EM_step(iterate,hmm,log_p,delta,converged)=assert_step(iterate,hmm,log_p)
24     → && new(iterate, hmm, log_p, delta, converged)
25 end
26
27 function assert_step(iterate, hmm, log_p)
28     iterate < 1 && throw(ArgumentError("EM_step iterate number must be a
29     → positive integer!"))
30     assert_hmm(hmm.a, hmm.A, hmm.B)
31     log_p > 0.0 && throw(ArgumentError("EM_step log probability value
32     → must be 0 or negative!"))
33     return true
34 end

```

---

### 16.6.9 /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
14        observations[o, obs_lengths[o]] = y && m == i && (Eoyim_T[o, y,
15            → i, m] = 0)
16    end
17    Tijm_T = fill(-Inf,O,N,N,N); Tijm_t = fill(-Inf,O,N,N,N) #Ti,j(T,m) =
18        → 0 for all m; in logspace
19
20    #RECURRENCE

```

```

15     ↵   βoi_T,Tijm_T,Eoyim_T=backwards_sweep!(hmm,A,N,Γ,βoi_T,βoi_t,Tijm_T,Tijm_t,Eoy
16
17 #TERMINATION
18 lls = c_llhs(hmm,observations[:,1])
19 α1om = lps.lls,a #first position forward msgs
20 log_pobs = [c_lse(lps.(α1om[o,:], βoi_T[o,:])) for o in 1:O]
21
22 Toij = [c_lse([lps(Tijm_T[o,i,j,m], α1om[o,m], -log_pobs[o]) for m in
23     ↵ 1:N]) for o in 1:O, i in 1:N, j in 1:N] #terminate Tijs with
24     ↵ forward messages
25 Eoiy = [c_lse([lps(Eoyim_T[o,y,i,m], α1om[o,m], -log_pobs[o]) for m
26     ↵ in 1:N]) for o in 1:O, i in 1:N, y in 1:Γ] #terminate Eids with
27     ↵ forward messages
28
29 #INTEGRATE ACROSS OBSERVATIONS AND SOLVE FOR NEW BHMM PARAMS
30 #INITIAL STATE DIST
31 a_o=α1om.+βoi_T.-c_lse.(eachrow(α1om.+βoi_T)) #estimate a for each o
32 obs_penalty=log(0) #broadcast subtraction to normalise log prob vals
33     ↵ by obs number
34 new_a=c_lse.(eachcol(a_o))-obs_penalty #sum over obs and normalise
35     ↵ by number of obs
36 #TRANSITION MATRIX
37 new_A = [c_lse(view(Toij,:,:i,j)) for i in 1:N, j in
38     ↵ 1:N]-c_lse.(eachcol(c_lse.([view(Toij,o,i,:) for o in 1:O, i in
39     ↵ 1:N])))
40 #EMISSION MATRIX
41 new_b=[c_lse(view(Eoiy,:,:i,y)) for i in 1:N, y in
42     ↵ 1:Γ]-c_lse.(eachcol(c_lse.([view(Eoiy,o,i,:) for o in 1:O, i in
43     ↵ 1:N])))
44 new_B=[Categorical(exp.(new_b[i,:])) for i in 1:N]
45
46 return BHMM(exp.(new_a), exp.(new_A), new_B, hmm.partition),
47     ↵ lps(log_pobs)
48 end
49 #LINEAR_STEP SUBFUNCS
50 function backwards_sweep!(hmm, A, N, Γ, βoi_T, βoi_t,
51     ↵ Tijm_T, Tijm_t, Eoyim_T, Eoyim_t, observations, mask,
52     ↵ obs_lengths)
53     for t in maximum(obs_lengths)-1:-1:1
54         last_β=copy(βoi_T)
55         lls = c_llhs(hmm,observations[:,t+1])
56         omask = findall(mask[:,t+1])

```

```

44     βoi_T[omask,:] .+= view(lls,omask,:)
45     Threads.@threads for m in 1:N
46         βoi_t[omask,m] =
47             → c_lse.(eachrow(view(βoi_T,omask,:).+transpose(view(A,
48                                         for j in 1:N, i in 1:N
49                                         Tijm_t[omask, i, j, m] .=
50                                         → c_lse.(eachrow(lps.(view(Tijm_T,omask,i,j,:),
51                                         → view(lls,omask,:),
52                                         → transpose(view(A,m,:))))))
53                                         i==m && (Tijm_t[omask, i, j, m] .=
54                                         → logaddexp.(Tijm_t[omask, i, j, m],
55                                         → lps.(last_β[omask,j],
56                                         → A[m,j],lls[omask,j]))))
57                                         end
58                                         for i in 1:N, y in 1:Γ
59                                         Eoyim_t[omask, y, i, m] .=
60                                         → c_lse.(eachrow(lps.(view(Eoyim_T,omask,y,i,:),view(
61                                         if i==m
62                                         symmask =
63                                         → findall(observations[:,t].==y)
64                                         Eoyim_t[symmask, y, i, m] .=
65                                         → logaddexp.(Eoyim_t[symmask, y, i,
66                                         → m], βoi_t[symmask,m]))
67                                         end
68                                         end
69                                         end
70                                         βoi_T=copy(βoi_t); Tijm_T=copy(Tijm_t); Eoyim_T =
71                                         → copy(Eoyim_t);
72                                         end
73                                         return βoi_T, Tijm_T, Eoyim_T
74                                     end
75
76                                     #logsumexp
77                                     function c_lse(X::AbstractArray{T}; dims=:) where
78                                         {T<:Real}
79                                         u=maximum(X)
80                                         u isa AbstractArray || isfinite(u) || return float(u)
81                                         return u + log(sum(x → exp(x-u), X))
82                                     end
83
84                                     #log likelihoods
85                                     function c_llhs(hmm, observation)
86                                         lls = zeros(length(observation),length(hmm.B))
87                                         for i in 1:length(observation)
88                                             for j in 1:length(hmm.B)
89                                                 lls[i,j] = logsumexp(c_lse(hmm.B[j,:]*hmm.T[:,observation[i]]))
90                                             end
91                                         end
92                                         return lls
93                                     end

```

```

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

```

---

### 16.6.10 /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))
28             pos_start = parse(Int, desc_array[2])
29             pos_end = parse(Int, desc_array[4])
30             scaffold_end = scaffold_df.End[findfirst(isequal(scaffold),
31                 ← scaffold_df.SeqID)]
32
33             pad_start=max(1, pos_start-pad)
34             pad_length= pos_start - pad_start

```

```

27         seq_offset = pad - pad_length
28         padded_seq = fetch_sequence(scaffold, scaffold_seq_dict,
29             → pad_start, pos_end, '+')
30
31     if !hasambiguity(padded_seq)
32         push!(position_df, [scaffold, pos_start, pos_end,
33             → padded_seq, pad_start, pad_length, seq_offset])
34     end
35 end
36
37 close(position_reader)
38 close(genome_reader)
39 return position_df
40
41 function add_partition_masks!(position_df::DataFrame, gff3_path::String,
42     → perigenic_pad::Integer=500,
43     → columns::Tuple{Symbol,Symbol,Symbol}=(:SeqID, :PadSeq, :PadStart))
44     partitions=["exon", "periexonic", "intergenic"]
45     partition_coords_dict = partition_genome_coordinates(gff3_path,
46         → perigenic_pad)
47     partitioned_scaffolds =
48         → divide_partitions_by_scaffold(partition_coords_dict)
49     maskcol = [zeros(Int64,0,0) for i in 1:size(position_df,1)]
50     position_df.MaskMatrix=maskcol
51
52     @Threads.threads for entry in eachrow(position_df)
53         scaffold = entry[columns[1]]
54         maskLength = length(entry[columns[2]])
55         seqStart = entry[columns[3]]
56
57         scaffold_coords_dict = Dict{String,DataFrame}()
58
59         for ((partition, part_scaffold), df) in partitioned_scaffolds
60             if scaffold == part_scaffold
61                 scaffold_coords_dict[partition] = df
62             end
63         end
64
65         entry.MaskMatrix=mask_sequence_by_partition(maskLength, seqStart,
66             → scaffold_coords_dict)
67     end

```

```

63
64 end
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
98
99     → seqMask[mask_position:min(maskLength,mask_position
100    → + partition_extent),1] .= partition_code
101    if position_strand == '+'
102      → seqMask[mask_position:min(maskLength,mask_position
103      → + partition_extent),2] .= 1

```

```

90         elseif position_strand == '-'
91             ↵     seqMask[mask_position:min(maskLength,mask_position
92                 ↵     + partition_extent),2] *= -1
93         else
94
95             ↵     seqMask[mask_position:min(maskLength,mask_position
96                 ↵     + partition_extent),2] *= 0
97         end
98
99         position += partition_extent + 1
100    end
101
102    return seqMask
103 end
104
105 function find_position_partition(position::Integer,
106     ↵ partition_dict::Dict{String, DataFrame})
107     foundPos = false
108     position_partition_id = ""
109     three_prime_extent = 0
110     sample_strand = 0
111     for (partition_id, partition) in partition_dict
112         hitindex = findfirst(x→x≥position,
113             ↵     partition.End)
114         if hitindex ≠ nothing
115             if position ≥ partition.Start[hitindex]
116                 foundPos=true
117                 position_partition_id = partition_id
118                 three_prime_extent =
119                     ↵     partition.End[hitindex] - position
120                 if partition_id == "exon" || partition_id
121                     ↵     == "perixonic"
122                     sample_strand =
123                         ↵     partition.Strand[hitindex]
124                 end
125             end
126         end
127     end
128
129     if foundPos == false
130         throw(DomainError("Position $position not found
131             ↵     among partition coordinates!"))
132     end

```

```

123         else
124             return position_partition_id, three_prime_extent,
125                 ↪ sample_strand
126         end
127     end
128 #function to partition a genome into coordinate sets of:
129 #merged exons
130 #"periexonic" sequences (genes with 5' and 3' boundaries projected
131   ↪ -/+perigenic_pad bp, minus exons) - includes promoter elements,
132   ↪ introns, 3' elements
133 #intergenic sequences (everything else)
134 #given a valid gff3
135 function partition_genome_coordinates(gff3_path::String,
136   ↪ perigenic_pad::Integer=500)
137     # construct dataframes of scaffolds and metacoordinates
138     scaffold_df = build_scaffold_df(gff3_path)
139
140     #partition genome into intragenic, periexonic, and exonic coordinate
141       ↪ sets
142     #assemble exonic featureset
143     exon_df = DataFrame(SeqID = String[], Start = Integer[], End =
144       ↪ Integer[], Strand=Char[])
145     build_feature_df!(gff3_path, "CDS", "MT", exon_df)
146
147     #project exon coordinates onto the scaffold bitwise, merging
148       ↪ overlapping features and returning the merged dataframe
149     merged_exon_df = DataFrame(SeqID = String[], Start = Integer[], End =
150       ↪ Integer[], Strand=Char[])
151     @showprogress 1 "Partitioning exons..." for scaffold_subframe in
152       ↪ DataFrames.groupby(exon_df, :SeqID) # for each scaffold subframe
153       ↪ that has exon features
154       scaffold_id = scaffold_subframe.SeqID[1] #get the scaffold id
155       scaffold_bitarray = init_scaffold_bitarray(scaffold_id,
156         ↪ scaffold_df, false, stranded=true) #init a stranded bitarray
157         ↪ of scaffold length
158
159       ↪ project_features_to_bitarray!(scaffold_subframe,scaffold_bitarray)
160       ↪ #project features as Trues on bitarray of falses
161     merged_subframe = get_feature_df_from_bitarray(scaffold_id,
162       ↪ scaffold_bitarray) #get a feature df from the projected
163       ↪ bitarray

```

```

149      append!(merged_exon_df,merged_subframe) #append the merged
150      → scaffold df to the overall merged df
151
152      end
153
154      #assemble gene featureset
155      gene_df = DataFrame(SeqID = String[], Start = Integer[], End =
156      → Integer[], Strand = Char[])
157      build_feature_df!(gff3_path, "gene", "MT", gene_df)
158
159      perigenic_pad > 0 && add_pad_to_coordinates!(gene_df, scaffold_df,
160      → perigenic_pad) #if a perigenic pad is specified (to capture
161      → promoter/downstream elements etc in the periexonic set), apply it
162      → to the gene coords
163
164      #build intergenic coordinate set by subtracting gene features from
165      → scaffold bitarrays
166      intergenic_df = DataFrame(SeqID = String[], Start = Integer[], End =
167      → Integer[])
168      @showprogress 1 "Partitioning intergenic regions ... " for
169      → scaffold_subframe in DataFrames.groupby(scaffold_df, :SeqID) #
170      → for each scaffold subframe that has exon features
171      scaffold_id = scaffold_subframe.SeqID[1] #get the scaffold id
172      scaffold_bitarray = init_scaffold_bitarray(scaffold_id,
173      → scaffold_df, true) #init a bitarray of scaffold length
174      if any(isequal(scaffold_id),gene_df.SeqID) #if any genes on the
175      → scaffold
176      scaffold_genes=gene_df[findall(isequal(scaffold_id),
177      → gene_df.SeqID), :] #get the gene rows by finding the
178      → scaffold_id
179
180      → subtract_features_from_bitarray!(scaffold_genes,scaffold_bitarray)
181      → #subtract the gene positions from the scaffold bitarray
182
183      end
184      intragenic_subframe = get_feature_df_from_bitarray(scaffold_id,
185      → scaffold_bitarray) #get a feature df from the projected
186      → bitarray
187      append!(intergenic_df,intragenic_subframe) #append the merged
188      → scaffold df to the overall merged df
189
190      end
191
192      #build periexonic set by projecting gene coordinates onto the
193      → scaffold bitwise, then subtracting the exons

```

```

172 periexonic_df = DataFrame(SeqID = String[], Start = Integer[], End =
173   ↪ Integer[], Strand=Char[])
174 @showprogress 1 "Partitioning periexonic regions ... " for
175   ↪ gene_subframe in DataFrames.groupby(gene_df, :SeqID) # for each
176   ↪ scaffold subframe that has exon features
177   scaffold_id = gene_subframe.SeqID[1] #get the scaffold id
178   scaffold_bitarray = init_scaffold_bitarray(scaffold_id,
179     ↪ scaffold_df, false, stranded=true) #init a bitarray of
180   ↪ scaffold length
181   project_features_to_bitarray!(gene_subframe,scaffold_bitarray)
182   ↪ #project features as Trues on bitarray of falses
183   if any(isequal(scaffold_id),merged_exon_df.SeqID)
184     scaffold_exons=merged_exon_df[findall(isequal(scaffold_id),
185       ↪ merged_exon_df.SeqID), :]
186
187     ↪ subtract_features_from_bitarray!(scaffold_exons,scaffold_bitarray)
188   end
189
190   periexonic_subframe = get_feature_df_from_bitarray(scaffold_id,
191     ↪ scaffold_bitarray) #get a feature df from the projected
192   ↪ bitarray
193   append!(periexonic_df,periexonic_subframe) #append the merged
194   ↪ scaffold df to the overall merged df
195
196 end

197
198 #partition_genome_coordinates() SUBFUNCTIONS
199 #BITARRAY SCAFFOLD REPRESENTATION SUBFUNCTIONS
200 function init_scaffold_bitarray(scaffold_id::String,
201   ↪ scaffold_df, value::Bool; stranded::Bool=false)
202   scaffold_length =
203     ↪ scaffold_df.End[findfirst(isequal(scaffold_id),
204       ↪ scaffold_df.SeqID)]
205   if stranded
206     value ? (return rscaffold_bitarray =
207       ↪ trues(scaffold_length,2)) : (return
208       ↪ rscaffold_bitarray = falses(scaffold_length,2))
209   else
210     value ? (return rscaffold_bitarray =
211       ↪ trues(scaffold_length,1)) : (return
212       ↪ rscaffold_bitarray = falses(scaffold_length,1))

```

```

196         end
197     end
198
199     function
200         → project_features_to_bitarray!(scaffold_feature_sf :: SubDataFrame,
201         → scaffold_bitarray :: BitArray)
202         @inbounds for item in eachrow(scaffold_feature_sf)
203             scaffold_bitarray[item.Start:item.End,1] = [true for
204                 → base in item.Start:item.End]
205             if size(scaffold_bitarray)[2] == 2 #if the bitarray
206                 → is stranded
207                 if item.Strand == '+'
208                     scaffold_bitarray[item.Start:item.End,2] =
209                         → [true for base in item.Start:item.End]
210             end
211         end
212     end
213
214     function
215         → subtract_features_from_bitarray!(scaffold_feature_sf :: DataFrame,
216         → scaffold_bitarray :: BitArray)
217         @inbounds for item in eachrow(scaffold_feature_sf)
218             scaffold_bitarray[item.Start:item.End,1] = [false for
219                 → base in item.Start:item.End]
220         end
221
222     function get_feature_df_from_bitarray(scaffold_id :: String,
223         → scaffold_bitarray :: BitArray)
224         size(scaffold_bitarray)[2] == 2 ? scaffold_feature_df =
225             → DataFrame(SeqID = String[], Start = Integer[], End =
226                 → Integer[], Strand=Char[]) : scaffold_feature_df =
227             → DataFrame(SeqID = String[], Start = Integer[], End =
228                 → Integer[])
229
230         new_feature_start = findnext(view(scaffold_bitarray,:,:1),
231             → 1)
232         while new_feature_start ≠ nothing # while new features
233             → are still found on the bitarray
234                 if size(scaffold_bitarray)[2] == 2 #if stranded,
235                     → get strand info

```

```

222             scaffold_bitarray[new_feature_start,2] == 1 ?
223                 new_feature_strand = '+'
224                 new_feature_strand = '-'
225             end
226         if
227             → findnext(!eval,view(scaffold_bitarray,:,:1),new_feature_start)
228             → ≠ nothing
229             new_feature_end =
230                 → findnext(!eval,view(scaffold_bitarray,:,:1),new_feature_start)
231                 → #find next false after feature start and
232                 → subtract 1 for feature end
233             else
234                 new_feature_end = size(scaffold_bitarray)[1]
235                     → #if none is found, the end of the feature
236                     → is the end of the scaffold
237             end
238             size(scaffold_bitarray)[2] == 2 ?
239                 → push!(scaffold_feature_df,[scaffold_id,
240                 → new_feature_start, new_feature_end,
241                 → new_feature_strand]) :
242                 → push!(scaffold_feature_df,[scaffold_id,
243                 → new_feature_start, new_feature_end]) #push
244                 → stranded feature info as appropriate
245             new_feature_start =
246                 → findnext(view(scaffold_bitarray,:,:1),
247                 → new_feature_end+1)
248             end
249         return scaffold_feature_df
250     end

```

---

### 16.6.11 /src/genome\_sampling/sequence\_sampler.jl

---

```

1 #####SAMPLING FUNCTIONS#####
2 #function for a Distributed worker to produce a set of samples of given
3 # parameters from genomic sequences
4 function get_sample_set(input_sample_jobs :: RemoteChannel,
5                         completed_sample_jobs :: RemoteChannel,
6                         progress_updates :: RemoteChannel)
7     while isready(input_sample_jobs)
8         genome_path, genome_index_path, partition_df, partitionid,
9             sample_set_length, sample_window_min, sample_window_max,
10            deterministic = take!(input_sample_jobs)

```

```

6
7     stranded::Bool = get_strand_dict()[partitionid]
8     scaffold_sequence_record_dict::Dict{String,LongSequence} =
9         → build_scaffold_seq_dict(genome_path, genome_index_path)
10
11    sample_df =
12        → DataFrame(SampleScaffold=String[], SampleStart=Integer[], SampleEnd=Integer[])
13    metacoordinate_bitarray = trues(partition_df.MetaEnd[end])
14    sample_set_counter = 0
15
16    while sample_set_counter < sample_set_length #while we don't yet
17        → have enough sample sequence
18        sample_scaffold::String, sample_Start::Integer,
19            → sample_End::Integer, sample_metaStart::Integer,
20            → sample_metaEnd::Integer, sample_sequence::LongSequence,
21            → strand::Char = get_sample(metacoordinate_bitarray,
22                → sample_window_min, sample_window_max, partition_df,
23                → scaffold_sequence_record_dict; stranded=stranded,
24                → deterministic=deterministic)
25        push!(sample_df,[sample_scaffold, sample_Start, sample_End,
26            → sample_sequence, strand]) #push the sample to the df
27        sample_length = sample_End - sample_Start + 1
28        sample_set_counter += sample_length #increase the counter by
29            → the length of the sampled sequence
30        metacoordinate_bitarray[sample_metaStart:sample_metaEnd] =
31            → [false for base in 1:sample_length] #mark these residues
32            → as sampled
33        put!(progress_updates, (partitionid,
34            → min(sample_set_counter,sample_set_length)))
35    end
36    put!(completed_sample_jobs,(partitionid, sample_df))
37
38 end
39 end
40
41 #get_sample_set() SUBFUNCTIONS
42 #function defining whether partitions respect stranding
43     → upon fetching sequence (ie is the sequence fetched in
44     → the feature strand orientation, or are we agnostic
45     → about the strand we sample?)
46 function get_strand_dict()
47     return
48         → Dict("exon"⇒true,"periexonic"⇒true,"intergenic"⇒false)
49 end

```

```

30 #function to obtain a dict of scaffold sequences from a
31   ↵ FASTA reader
32
33 function build_scaffold_seq_dict(genome_fa, genome_index)
34   genome_reader = open(FASTA.Reader, genome_fa,
35     ↵ index=genome_index)
36   seq_dict::Dict{String, LongSequence} =
37     ↵ Dict{String,FASTA.Record}()
38   @inbounds for record in genome_reader
39     id = FASTA.identifier(record)
40
41       ↵ seq_dict[rectify_identifier(id)]=FASTA.sequence(record)
42   end
43   close(genome_reader)
44   return seq_dict
45 end
46
47
48 # function to convert scaffold ID from that observed by
49   ↵ the masked .fna to the more legible one observed by
50   ↵ the GRCz11 GFF3
51
52 function rectify_identifier(scaffold_id::String)
53   if length(scaffold_id) ≥ 4 && scaffold_id[1:4] ==
54     ↵ "CM00" #marks chromosome scaffold
55     chr_code = scaffold_id[5:10]
56     chr_no = "$(Int((parse(Float64,chr_code)) -
57       ↵ 2884.2))"
58     return chr_no
59   else
60     return scaffold_id
61   end
62 end
63
64
65 #function to produce a single sample from a
66   ↵ metacoordinate set and the feature df
67
68 function get_sample(metacoordinate_bitarray::BitArray,
69   ↵ sample_window_min::Integer,
70   ↵ sample_window_max::Integer, partition_df::DataFrame,
71   ↵ scaffold_seq_dict::Dict{String,LongSequence};
72   ↵ stranded::Bool=false, deterministic::Bool=false)
73   proposal_acceptance = false
74   sample_metaStart = 0
75   sample_metaEnd = 0
76   sample_Start = 0
77   sample_End = 0

```

```

60             sample_sequence = LongSequence{DNAAlphabet{2}}("")  

61             sample_scaffold = ""  

62             strand = nothing  

63  

64             while proposal_acceptance == false  

65                 available_indices =  

66                     → findall(metacoordinate_bitarray) #find all  

67                     → unsampled indices  

68                 window = "FAIL"  

69                 start_index,feature_metaStart,feature_metaEnd,  

70                     → feature_length = 0,0,0,0  

71                 strand = '0'  

72                 while window == "FAIL"  

73                     start_index = rand(available_indices)  

74                         → #randomly choose from the unsampled  

75                         → indices  

76                     feature_metaStart, feature_metaEnd, strand =  

77                         → get_feature_params_from_metacoord(start_index,  

78                         → partition_df, stranded)  

79                     feature_length =  

80                         → length(feature_metaStart:feature_metaEnd)  

81                         → #find the metaboundaries of the feature  

82                         → the index occurs in  

83                     if feature_length ≥ sample_window_min #don't  

84                         → bother finding windows on features  

85                         → smaller than min  

86                     window =  

87                         → determine_sample_window(feature_metaStart,  

88                         → feature_metaEnd, start_index,  

89                         → metacoordinate_bitarray,  

90                         → sample_window_min, sample_window_max)  

91                         → #get an appropriate sampling window  

92                         → around the selected index, given the  

93                         → feature boundaries and params  

94                 end  

95             end  

96  

97             sample_scaffoldid, sample_scaffold_start,  

98                 → sample_scaffold_end =  

99                 → meta_to_feature_coord(window[1],window[2],partition_df)

```

```

80         proposal_sequence =
81             ↳ fetch_sequence(sample_scaffoldid,
82                             ↳ scaffold_seq_dict, sample_scaffold_start,
83                             ↳ sample_scaffold_end, strand;
84                             ↳ deterministic=deterministic) #get the
85                             ↳ sequence associated with the sample window
86
87
88
89
90     end
91
92     return sample_scaffold, sample_Start, sample_End,
93             ↳ sample_MetaStart, sample_MetaEnd,
94             ↳ sample_sequence, strand
95
96
97
98
99
100
101
102
103
#function to find a valid sampling window
function
    ↳ determine_sample_window(feature_MetaStart::Integer,
    ↳ feature_MetaEnd::Integer, metacoord::Integer,
    ↳ metacoord_bitarray::BitArray,
    ↳ sample_window_min::Integer,
    ↳ sample_window_max::Integer)
        window_start = 0
        window_end = 0
        feature_size = feature_MetaEnd - feature_MetaStart +
            ↳ 1
        feature_sampled = any(!eval,
            ↳ metacoord_bitarray[feature_MetaStart:feature_MetaEnd])
#attempt to construct the sample window by taking the
    ↳ whole feature
if !feature_sampled && feature_size <
    ↳ sample_window_max && feature_size >
    ↳ sample_window_min
        window_start = feature_MetaStart

```

```

104         window_end = feature_metaEnd
105         return window_start, window_end
106     else #if this fails, build the biggest window we can
107         ← from the sampling point, within the feature
108         featurepos = metacoord - feature_metaStart + 1
109         next_sampled_index = findnext(!eval,
110             ← metacoord_bitarray[feature_metaStart:feature_metaEnd],
111             ← featurepos)
112         prev_sampled_index = findprev(!eval,
113             ← metacoord_bitarray[feature_metaStart:feature_metaEnd],
114             ← featurepos)
115         next_sampled_index ≡ nothing ? (window_end =
116             ← feature_metaEnd) : (window_end =
117             ← next_sampled_index + feature_metaStart - 1)
118         prev_sampled_index ≡ nothing ? (window_start =
119             ← feature_metaStart) : (window_start =
120             ← prev_sampled_index + feature_metaStart - 1)
121         windowsize = window_end - window_start + 1
122         #check to see if this window is bigger than the
123         ← min, if not, return a failure code
124         windowsize < sample_window_min && return "FAIL"
125         #check to see if this window is bigger than the
126         ← max, if so, trim it before returning,
127         ← removing as evenly as possible around the
128         ← metacoordinate
129         if windowsize > sample_window_max
130             bases_to_trim = windowsize -

```

```

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
150 ######SHARED SEQUENCE FETCHER#####
151 # function to get proposal sequence from dict of scaffold sequences,
152 #→ given coords and scaffold id
153 function fetch_sequence(scaffold_id::String,
154     → scaffold_seq_dict::Dict{String, LongSequence},
155     → proposal_start::Integer, proposal_end::Integer, strand::Char;
156     → deterministic=false)
157     if strand == '0' #unstranded samples may be returned with no
158         → preference in either orientation
159         deterministic ? strand = '+' : (rand(1)[1] ≤ .5 ? strand = '+' :
160             → strand = '-')
161     end
162     if strand == '+'
163         proposal_sequence =
164             → scaffold_seq_dict[scaffold_id][proposal_start:proposal_end]
165     elseif strand == '-'
166         proposal_sequence =
167             → reverse_complement(scaffold_seq_dict[scaffold_id][proposal_start:proposal_end])
168     else
169         throw(ArgumentError("Invalid sample code! Must be '+', '-' , or
170             → '0' (random strand)"))
171     end
172
173     return proposal_sequence
174 end

```

```

162
163 #####SHARED BASIC COORDINATE SUBFUNCTIONS#####
164 # function to push scaffold ID, start, and end points of given
165 #       ↳ featuretype to supplied dataframe
166 function build_feature_df!(GFF3_path::String, feature_type::String,
167   ↳ scaffold_exclusion::String, feature_df::DataFrame)
168   reader = open(GFF3.Reader, GFF3_path) # access the GFF3
169   @inbounds for record in reader # iterate over Gff3 records
170     if GFF3.isfeature(record) # if the record is a feature
171       if GFF3.featuretype(record) == feature_type #if the features
172         ↳ is of the requested type, get the following info
173         seqID = GFF3.seqid(record)
174         if seqID != scaffold_exclusion
175           seq_start = GFF3.seqstart(record)
176           seq_end = GFF3.seqend(record)
177           if feature_type == "CDS" || feature_type == "gene"
178             seq_strand = convert(Char, GFF3.strand(record))
179             push!(feature_df, [seqID, seq_start, seq_end,
180               ↳ seq_strand])
181           else
182             push!(feature_df, [seqID, seq_start, seq_end]) #
183               ↳ push relevant info to the df
184           end
185         end
186       end
187     end
188     close(reader)
189   end
190
191   #function to assemble dataframe of scaffold coords + metacoords given
192   ↳ gff3
193   function build_scaffold_df(gff3_path)
194     scaffold_df = DataFrame(SeqID = String[], Start = Integer[], End =
195       ↳ Integer[])
196     build_feature_df!(gff3_path, "supercontig", "MT", scaffold_df)
197     build_feature_df!(gff3_path, "chromosome", "MT", scaffold_df)
198     add_metacoordinates!(scaffold_df)
199     return scaffold_df
200   end
201
202   #function to add pad to either side of some featureset

```

```

197 function add_pad_to_coordinates!(feature_df::DataFrame,
198   → scaffold_df::DataFrame, pad_size::Integer;
199   → col_symbols::Array{Symbol}=[:Start, :End])
200   pad_start_array = zeros(Integer,size(feature_df,1))
201   pad_end_array = zeros(Integer,size(feature_df,1))
202   feature_df[!, col_symbols[1]] = [max(feature_df.Start[i]-pad_size,1)
203     → for i in 1:size(feature_df,1)] #truncate pads at beginning and
204     → end of scaffolds
205   feature_df[!, col_symbols[2]] =
206     → [min(feature_df.End[i]+pad_size,scaffold_df.End[findfirst(isequal(feature_df.
207       → .Start, scaffold_df.Start))],1)
208     → for i in 1:size(feature_df,1)]
209 end
210
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   feature_df.MetaStart = meta_start_array # metacoordinate contains
226     → 'start' metaposition across all genomic material
227   feature_df.MetaEnd = meta_end_array # metacoordinate contains 'end'
228     → metaposition across all genomic material
229 end
230
231
232 # function to convert metacoordinate set to scaffold-relative coordinates
233 function meta_to_feature_coord(meta_start::Integer, meta_end::Integer,
234   → feature_df::DataFrame)
235   feature_row = get_feature_row_index(feature_df, meta_start)
236   seqid = feature_df.SeqID[feature_row]
237   scaffold_start = feature_df.Start[feature_row] + (meta_start -
238     → feature_df.MetaStart[feature_row])
239   scaffold_end = feature_df.End[feature_row] -
240     → (feature_df.MetaEnd[feature_row] - meta_end)
241   return seqid, scaffold_start, scaffold_end
242 end
243

```

```

228
229 #function to obtain the feature boundaries and strand of the feature that
   ↳ a metacoordinate falls within
230 function get_feature_params_from_metacoord(metacoordinate::Integer,
   ↳ feature_df::DataFrame, stranded::Bool)
231     feature_row = get_feature_row_index(feature_df, metacoordinate)
232     feature_metaStart = feature_df.MetaStart[feature_row]
233     feature_metaEnd = feature_df.MetaEnd[feature_row]
234     stranded ? feature_strand = feature_df.Strand[feature_row] :
   ↳ feature_strand = '0'
235     return feature_metaStart, feature_metaEnd, feature_strand
236 end
237
238 #function obtain the index of a feature given its metacoordinate
239 function get_feature_row_index(feature_df::DataFrame,
   ↳ metacoordinate::Integer)
240     total_feature_bases = feature_df.MetaEnd[end]
241     if metacoordinate == total_feature_bases #edge case of metacoord at
       ↳ end of range
         return size(feature_df,1) #last index in df
242     else
243         index =
244             findfirst(end_coord→end_coord>metacoordinate, feature_df.MetaEnd)
245             ↳ #find the index of the feature whose end metacoord is > the
               ↳ query metacoord
246             if index > 1 && metacoordinate == feature_df.MetaEnd[index-1] #if
               ↳ the metacoordinate is the last base of the previous feature
247                 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
248             end
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             else
251                 throw(DomainError("Unexpected metacoordinate $metacoordinate
                   ↳ in partition of $total_feature_bases bases, with feature
                   ↳ start $(feature_df.MetaStart[index]), end
                   ↳ $(feature_df.MetaEnd[index]))")
252             end
253         end
254     end

```

---

 255 end
 

---

### 16.6.12 /src/likelihood\_funcs/bg\_lh\_matrix.jl

```

1 #function to obtain positional likelihoods for a sequence under a given
2   ↳ BHMM.
3 function get_BGHMM_symbol_lh(seq::AbstractMatrix, hmm::BHMM)
4   @assert size(seq)[1] == 1
5   (seq=Array(transpose(seq))) # one sequence at a time only
6   symbol_lhs = zeros(length(seq))
7   length_mask = [length(seq)-1]
8
9   lls = bw_llhs(hmm, seq) #obtain log likelihoods for sequences and
10    ↳ states
11  log_α = messages_forwards_log(hmm.a, hmm.A, lls, length_mask) #get
12    ↳ forward messages
13  log_β = messages_backwards_log(hmm.A, lls, length_mask) # get
14    ↳ backwards messages
15
16  #calculate observation probability and γ weights
17  K,Tmaxplus1,Strand = size(lls) #the last T value is the 0 end marker
18    ↳ of the longest T
19
20  #transforms to cut down log_ξ, log_γ assignment times
21  lls = permutedims(lls, [2,1,3]) # from (K,T) to (T,K)
22  log_α = permutedims(log_α, [2,1,3])
23  log_β = permutedims(log_β, [2,1,3])
24
25  log_γ = fill(-Inf, Tmaxplus1,K)
26  log_pobs = logsumexp(lps.(log_α[1,:], log_β[1,:]))
27
28  @inbounds for i = 1:K, t = 1:length_mask[1]
29      log_γ[t,i] = lps(log_α[t,i],log_β[t,i],-log_pobs)
30  end
31
32  for t in 1:length_mask[1]
33      symbol_lh::AbstractFloat = -Inf #ie log(p=0)
34      for k = 1:K #iterate over states
35          state_symbol_lh::AbstractFloat = lps(log_γ[t,k],
36            ↳ log(hmm.B[k].p[seq[t]])) #state symbol likelihood is
37            ↳ the γ weight * the state symbol probability (log
38            ↳ implementation)

```

```

31         symbol_lh = logaddexp(symbol_lh, state_symbol_lh) #sum
32             ↵ the probabilities over states
33     end
34     symbol_lhs[t] = symbol_lh
35   end
36
37   return symbol_lhs[1:end-1] #remove trailing index position
38 end
39
40 #function to calculate BGHMM from an observation set and a dict of BGHMMs
41 function BGHMM_likelihood_calc(observations::DataFrame, BGHMM_dict::Dict,
42   ↵ code_partition_dict = get_partition_code_dict(false); symbol=:PadSeq)
43   lh_matrix_size = ((findmax(length.(collect(values(observations[!,,
44   ↵ symbol]))))[1]), length(observations[!, symbol]))
45   BGHMM_lh_matrix = zeros(lh_matrix_size) #T, Strand, 0
46
47   #@showprogress 1 "Writing frags to matrix.."
48   Threads.@threads for (jobid, frag) in BGHMM_fragments
49     (frag_start, o, partition, strand) = jobid
50
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   return BGHMM_lh_matrix

```

```

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}}}()

76   frag = LongSequence{DNAAlphabet{2}}()

77   frag_end=0
78   frag_start = 1

79   while frag_start < length(obs_seq) # while we're not at the
80     ↳ sequence end
81     curr_partition = mask[frag_start,1] #get the partition code
82     ↳ of the frag start
83     curr_strand = mask[frag_start,2] #get the strand of the frag
84     ↳ start

85     #JOBID COMPOSED HERE
86     jobid = (frag_start, o, curr_partition, curr_strand) #compose
87     ↳ an identifying index for this frag

88     findnext(!isequal(curr_partition),mask[:,1],frag_start) ≠
89     ↳ nothing ? frag_end =
90     ↳ findnext(!isequal(curr_partition),mask[:,1],frag_start)
91     ↳ -1 : frag_end = length(obs_seq) #find the next position
92     ↳ in the frag that has a different partition mask value
93     ↳ from hte current one and set that position-1 to frag end,
94     ↳ alternately frag end is end of the overall sequence
95     frag = obs_seq[frag_start:frag_end] #get the frag bases
96     if curr_strand == -1 #if the fragment is reverse stranded
97       ↳ reverse_complement!(frag) #use the reverse complement
98       ↳ sequence
99     end

100    push!(likelihood_jobs,(jobid, frag)) #put the frag in the
101      ↳ jobs vec
102    frag_start = frag_end + 1 #move on
103
104 end

```

```

95     end
96     return likelihood_jobs
97 end

```

---

### 16.6.13 /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
6 function bw_likelihood(observations, hmm)
7     obs_lengths = [findfirst(iszero, observations[:,o])-1 for o in
8         ↳ 1:size(observations,2)]
9
10    lls = bw_llhs(hmm, observations)
11    log_α = messages_forwards_log(hmm.a, hmm.A, lls, obs_lengths)
12    log_β = messages_backwards_log(hmm.A, lls, obs_lengths)
13
14    O = size(lls)[3] #the last T value is the 0 end marker of the longest
15    ↳ T
16    log_pobs = zeros(O)
17    Threads.@threads for o in 1:O
18        log_pobs[o] = logsumexp(lps.(log_α[:,1,o], log_β[:,1,o]))
19    end
20
21    return sum(log_pobs)
22 end
23
24 function linear_likelihood(observations,hmm)
25     O= size(observations,1);
26     obs_lengths = [findfirst(iszero,observations[o,:])-1 for o in
27         ↳ 1:size(observations,1)] #mask calculations here rather than
28         ↳ mle_step to prevent recalculation every iterate
29
30     a = transpose(log.(hmm.a)); A = log.(hmm.A)
31     N = length(hmm.B)
32     mask=observations.≠0
33     #INITIALIZATION
34     βoi_T = zeros(0,N); βoi_t = zeros(0,N) #log betas at T initialised as
35         ↳ zeros

```

```

31     #RECURRENCE
32     βoi_T = backwards_lh_sweep!(hmm, A, N, βoi_T, βoi_t, observations,
33                               → mask, obs_lengths)
34
35     #TERMINATION
36     lls = c_llhs(hmm,observations[:,1])
37     α1om = lls .+ a #first position forward msgs
38
39     return lps([logsumexp(lps.(α1om[o,:], βoi_T[o,:])) for o in 1:0])
40 end
41
42     #LINEAR_STEP SUBFUNCS
43     function backwards_lh_sweep!(hmm, A, N, βoi_T, βoi_t,
44                               → observations, mask, obs_lengths)
45         for t in maximum(obs_lengths)-1:-1:1
46             last_β=copy(βoi_T)
47             lls = c_llhs(hmm,observations[:,t+1])
48             omask = findall(mask[:,t+1])
49             βoi_T[omask,:] .+= view(lls,omask,:)
50             Threads.@threads for m in 1:N
51                 βoi_t[omask,m] =
52                     → logsumexp.(eachrow(view(βoi_T,omask,:).+transpose(view(
53                         end
54                         βoi_T=copy(βoi_t)
55
56                         end
57                         return βoi_T
58
59                         end

```

---

### 16.6.14 /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 println(io, "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)\n")
21 report.test_lh > report.naive_lh ? (lh_str="greater";
22   → lh_color=:green) : (lh_str="less"; lh_color=:red)
23 println(io, "Test logP(0|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   → -----
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   → lh_plot=lineplot([zidx:length(lh_vec)+zidx-1 ... ],lh_vec;title="Chain
37   → likelihood evolution", xlabel="Training iterate",
38   → xlim=(0,length(lh_vec)+zidx+1), ylim=
39   → (floor(minimum(lh_vec),sigdigits=2),ceil(maximum(lh_vec),sigdigits=2)),
40   → name="logP(0|θ)")
41 lineplot!(lh_plot,[report.naive_lh for i in 1:length(lh_vec)],
42   → color=:magenta,name="naive")
43 show(io, lh_plot)
44 println(io, "\n")
45 k1vec=Vector([report.convergence_values["K1"].data ... ])

```

```

40     k_plot=lineplot(k1vec, title="State p(Auto) evolution",
41         ← xlabel="Training iterate", ylabel="prob",
42         ← name="K1",ylim=(0,1),xlim=(0,length(k1vec)))
43     for k in 2:report.id.K
44         kvec=Vector([report.convergence_values["K$k"].data ... ])
45         lineplot!(k_plot,kvec,name="K$k")
46     end
47     show(io,k_plot)
48     println(io)
49
50     printstyled(io, "\nConvergence Diagnostics\n",bold=true)
51     if report.convergence_diagnostic.name == "short"
52         printstyled(io, "Convergence diagnostics unavailable for chains
53             ← <10 steps!\n", color=:yellow)
54     elseif report.convergence_diagnostic.name == "error"
55         printstyled(io, "Convergence diagnostics errored! Zeros in
56             ← autotransition matrix?\n", color=:red)
57     else
58         all(Bool.(report.convergence_diagnostic.nt.stationarity)) &&
59             all(Bool.(report.convergence_diagnostic.nt.test)) ?
60                 printstyled(io, "Likelihood and autotransition probabilites
61                     ← converged and passing tests.\n", color=:green) :
62                     printstyled(io, "Not all parameters converged or passing
63                         ← tests!\n",color=:red)
64         display(report.convergence_diagnostic)
65     end
66   end
67 end
68
69 function latex_report(report::Chain_Report)
70
71 end
72
73 function report_chains(chains::Dict{Chain_ID,Vector{EM_step}},
74     ← test_sets::Dict{String,Vector{LongSequence{DNAAlphabet{2}}}});
75     ← naive_hmm=BHMM(ones(1,1),[Categorical(4)])
76     job_ids=[id for (id,chain) in chains] # make list of chain_ids
77     partitions=unique([id.obs_id for id in job_ids]) #get list of unique
78         ← partitions represented among chain_ids
79     naive_order = Integer(log(4,length(naive_hmm.B[1].support))-1)
80     for partition in partitions #check that all partitions are available
81         ← for testing and make sure codes for naive hmm are generated by
82         ← adding job_ids

```

```

68     !(partition in keys(test_sets)) &&
69         → throw(ArgumentError("Observation set $partition required by
70             → chains for testing not present in test sets!"))
71
72     push!(job_ids,
73         → Chain_ID(partition,length(naive_hmm.a),naive_order, 1))
74 end
75
76 code_dict=code_job_obs(job_ids, test_sets) #code all the obs sets
77     → that are necessary ot perform our tests
78 naive_lhs=Dict{String,Float64}() #construct dict of naive likelihoods
79     → for all partitions to be tested prior to individual tests
80 @showprogress 1 "Testing naive hmm..." for
81     → ((partition,order),obs_set) in code_dict
82     order=naive_order &&
83         → (naive_lhs[partition]=obs_lh_given_hmm(code_dict[(partition,naive_order)])
84 end
85
86 reports=Dict{Chain_ID,Chain_Report}()
87 @showprogress 1 "Testing chains..." for (id, chain) in chains
88     chains_array=zeros(length(chain),1+id.K)
89     for (n,step) in enumerate(chain)
90         chains_array[n,1]=step.log_p
91         chains_array[n,2:end]=get_diagonal_array(step.hmm)
92     end
93     convergence_values=Chains(chains_array,[ "logP(0|θ)",
94         → ["K"*string(i) for i in 1:id.K] ... ])
95     if length(chain) ≥ 10
96         try
97             convergence_diagnostic=heideldiag(convergence_values)[1]
98         catch
99             → convergence_diagnostic=ChainDataFrame("error",(:;a⇒[1.]))
100         end
101     else
102         convergence_diagnostic=ChainDataFrame("short",(:;a⇒[1.]))
103     end
104
105     id.K > 1 ?
106         → (mean_run_lengths=sim_run_lengths(get_diagonal_array(chain[end].hmm),1000
107         → : (mean_run_lengths=[Inf])
108         test_lh=obs_lh_given_hmm(code_dict[(id.obs_id,id.order)],
109         → chain[end].hmm)

```

```

98     reports[id]=Chain_Report(id, chain[end].hmm, test_lh,
99         ← naive_lhs[id.obs_id], chain[end].delta, mean_run_lengths,
100        ← convergence_values, convergence_diagnostic,
101        ← chain[end].converged)
102    end
103
104   return reports
105 end
106
107
108 function get_diagonal_array(hmm::BHMM)
109     k = length(hmm.a)
110     diagonal = zeros(k)
111     @inbounds for i in 1:k
112         diagonal[i] = hmm.A[i,i]
113     end
114     return diagonal
115 end
116
117 #function to simulate run lengths for vector of diagonal values
118 function sim_run_lengths(diagonal_value::AbstractArray, samples::Integer)
119     mean_run_lengths = zeros(length(diagonal_value))
120     for (i, value) in enumerate(diagonal_value)
121         if value == 0.
122             mean_run_lengths[i]=0.
123         elseif value ==1.
124             mean_run_lengths[i]=Inf
125         else
126             runlengths = zeros(Integer, samples)
127             for s in 1:samples
128                 run = true
129                 runlength = 0
130                 while run
131                     runlength += 1
132                     if rand(1)[1] > value
133                         run = false
134                     end
135                 end
136                 runlengths[s] = runlength
137             end
138             mean_run_lengths[i] = mean(runlengths)
139         end
140     end
141     return mean_run_lengths

```

138 end

### 16.6.15 /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(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
37         else
38             println("All chains have converged")
39         end
40     end
41 end

```

```

28         ordplot=scatterplot(ordreport.converged_K,
29             ←  ordreport.converged_lh; name="Converged", title="Order
30             $order HMMs", xlabel="K# States", ylabel="P(O|θ)",
31             ←  color=:green, xlim=(1,maximum(ordreport.converged_K)),
32             ←  ylim=(floor(min(minimum(ordreport.converged_lh),report.naive_lh),sigd
33             lineplot!(ordplot, [report.naive_lh for i in
34                 ←  1:maximum(ordreport.converged_K)],
35                 ←  name="Naive",color=:magenta)
36             show(ordplot)
37             println()
38         end
39     end
40 end
41
42 function report_partitions(chain_reports::Dict{Chain_ID,Chain_Report})
43     partitions=Vector{String}()
44     orders=Vector{Integer}()
45     Ks=Vector{Integer}()
46     reps=Vector{Integer}()
47     for id in keys(chain_reports)
48         !in(id.obs_id, partitions) && push!(partitions, id.obs_id)
49         !in(id.order, orders) && push!(orders, id.order)
50         !in(id.K, Ks) && push!(Ks, id.K)
51         !in(id.replicate, reps) && push!(reps, id.replicate)
52     end
53
54     reports=Vector{Partition_Report}()
55     for partition in partitions
56         best_lh,best_rep=-Inf,Chain_ID("empty",1,0,1)
57         naive_lh=1.
58         orddict=Dict{Integer,Order_Report}()
59         for order in orders
60             conv_statevec=Vector{Float64}()
61             fail_statevec=Vector{Float64}()
62             conv_lh_vec=Vector{Float64}()
63             fail_lh_vec=Vector{Float64}()
64             for (id,report) in chain_reports
65                 if id.obs_id==partition && id.order==order
66                     report.test_lh > best_lh &&
67                         (best_lh=report.test_lh;best_rep=id)
68                     naive_lh=1. && (naive_lh=chain_reports[id].naive_lh)
69                     chain_reports[id].converged ?
70                         (push!(conv_statevec,id.K);

```

```

63         push!(conv_lh_vec,chain_reports[id].test_lh)) :
64             ↳ (push!(fail_statevec,id.K);
65             push!(fail_lh_vec,chain_reports[id].test_lh))
66     end
67
68         ↳ orddict[order]=Order_Report(conv_statevec,conv_lh_vec,fail_statevec,f
69     end
70     best_model=(best_rep,chain_reports[best_rep].final_hmm)
71     best_repset=[Chain_ID(partition, best_rep.K, best_rep.order, rep)
72         ↳ for rep in reps]
73
74         ↳ push!(reports,Partition_Report(partition,naive_lh,orddict,best_model,best_
75     end
76     return reports
77 end

```

---

### 16.6.16 /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 end
18
19 function assert_repreport(ids::Vector{Chain_ID})
20     Ks=Vector{Int64}()
21     orders=Vector{Int64}()
22     replicates=Vector{Int64}()

```

```

17     obsids=Vector{String}()
18
19     for id in ids
20         push!(Ks,id.K)
21         push!(orders,id.order)
22         push!(obsids,id.obs_id)
23         push!(replicates, id.replicate)
24     end
25
26     length(unique(Ks)) > 1 && throw(ArgumentError("Replicate set includes
27     ↳ chains with different K state numbers! All chains must have HMMs
28     ↳ with the same number of states."))
29     length(unique(orders)) > 1 && throw(ArgumentError("Replicate set
30     ↳ includes chains with different order coding numbers! All chains
31     ↳ must have HMMs with the same order coding."))
32     !allunique(replicates) && throw(ArgumentError("Replicate set includes
33     ↳ chains with the same replicate #. Replicates should be unique!"))
34
35     return true
36   end
37
38   function
39     ↳ report_replicates(repset::Vector{Chain_ID},chains::Dict{Chain_ID,Vector{EM_step}})
40     assert_repreport(repset) #check that the repset will pass its
41     ↳ constructor before doing the work
42     emission_arrays=Dict{Chain_ID,AbstractArray{AbstractFloat}}()
43     state_arrays=Dict{Chain_ID,AbstractArray{AbstractFloat}}()
44
45     for id in repset
46       emission_array=zeros(length(chains[id]),4^(id.order+1),id.K)
47       state_array=zeros(length(chains[id]),id.K)
48       for (it, step) in enumerate(chains[id])
49         for k in 1:id.K
50           emission_array[it,:,k]=step.hmm.B[k].p
51           state_array[it,k]=step.hmm.A[k,k]
52         end
53       end
54       emission_arrays[id]=emission_array
55       state_arrays[id]=state_array
56     end
57
58     println(repset)
59     println(emission_arrays)

```

```

53   println(state_arrays)
54
55   → sort_dicts,sorted_id1_states,sorted_symbols=sort_emitters_by_distance!(repset
56
57   return
58     → Replicate_Report(repset,state_arrays,emission_arrays,sorted_id1_states,
59     → sort_dicts,sorted_symbols)
60 end
61
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)
75   for rep in 1:length(repset)-1
76     for k in 1:id1.K, j in 1:id1.K
77
78       → distances[rep,k,j]=euclidean(final_emissions[1,:,:k],final_emissions[1
79     end
80   end
81
82   sortdicts=Dict{Chain_ID,Dict{Integer,Integer}}()
83   sorted_id1_states=Vector{Integer}()
84   sorted_symbols=Dict{Integer,Vector{Integer}}()
85   for rep in 1:length(repset)-1
86     sortdicts[repset[rep+1]]=Dict{Integer,Integer}()
87   end
88
89   for i in 1:id1.K
90     id1_mindist_K=findmin([sum([minimum(distances[rep,k,:]) for rep
91       → in 1:length(repset)-1] for k in 1:id1.K])[2]#find the state
92       → from the first replicate that has minimum cumulative distance
93       → to the closest state in the other replicates, excluding
94       → already-chosen states by spiking their values
95     push!(sorted_id1_states,id1_mindist_K)
96   for rep in 1:length(repset)-1

```

```

86         rep_k=findmin(distances[rep,id1_mindist_K,:])[2]
87         sortdicts[repset[rep+1]][id1_mindist_K]=rep_k
88         distances[rep,:,:rep_k] *= 1.0 #mask
89         distances[1,id1_mindist_K,:] *= 1.0 #mask
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 end
97
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)
103   println(" -----")
104   → -----
105   → -----
106   for i in 1:top_states
107     id=report.ids[1]
108     id1_state=report.sorted_id1_states[i]
109     itmax=maximum([size(report.state_vecs[id],1) for id in
110       → report.ids])
111
112     autoplt=lineplot(report.state_vecs[id][:,id1_state],
113       title="Autotransition prob. convergence",
114       name="$(report.ids[1]) K$id1_state",
115       xlim=(0,ceil(itmax, sigdigits=2)),
116       ylim=(0,1),
117       xlabel="iterate",
118       ylabel="p")
119
120     → symplot=lineplot(report.emission_arrays[id][:,report.sorted_symbols[id1_st

```

```
121         ylim=(0,1),  
122         xlabel="Symbol 1 p",  
123         ylabel="S2 p")  
124  
125     for (n,id) in enumerate(report.ids)  
126         if n > 1  
127             lineplot!(autoplt,  
128                     report.state_vecs[id][:,report.sort_dicts[id][id1_state]],  
129                     name="$(report.ids[n])"  
130                     ↪ K$(report.sort_dicts[id][id1_state]))  
131             lineplot!(symplot,  
132                     ↪ report.emission_arrays[id][:,report.sorted_symbols[id]]  
133                     ↪ report.emission_arrays[id][:,report.sorted_symbols[id]]  
134                     name="$(report.ids[n])"  
135                     ↪ K$(report.sort_dicts[id][id1_state]))  
136         end  
137     end  
138  
139     show(autoplt)  
140     println("\n")  
141     show(symplot)  
142     println("\n")  
143 end  
144 end
```

### 16.6.17 /src/utilities/BBG analysis.jl

```
1 #function to produce matrix of hmm chain parameter coordinate evolution-
  ↵ selects 3 matched emission parameters from 3+ state HMMs, one per
  ↵ state for 3 states. Matching is performed by minimizing euclidean
  ↵ distance between the state emission probability vectors B for the
  ↵ states to be matched (as EM-optimised replicates will have similar
  ↵ states organized in different orders)

2

3 function chain_3devo_coords(chains :: Vector{Vector{Any}})
  ↵ length(chains) ≤ 0 && throw(ArgumentError, "Argument must be vector of
  ↵ more than one hmm chain")
  ↵ coords=[Vector{Tuple{AbstractFloat, AbstractFloat, AbstractFloat}}]()
  ↵ for i in 1:length(chains)]
```

```

7   for step in chains[1]
8     hmm=step[2]
9     length(hmm.B)<3 && throw(ArgumentError, "3- or greater state hmms
10    ↪ required")
11    push!(coords[1], (hmm.B[1].p[1],hmm.B[2].p[1],hmm.B[3].p[1]))
12  end
13
14  for (c, chain) in enumerate(chains[2:end])
15    ref_idxs=Vector{Integer}()
16
17    ↪ ref_vecs=[chains[1][end][2].B[1].p,chains[1][end][2].B[2].p,chains[1][end]
18    end_hmm=chain[end][2]
19    length(end_hmm.B)<3 && throw(ArgumentError, "3- or greater state
20    ↪ hmms required")
21
22    for vec in ref_vecs
23      euclideans=Vector{AbstractFloat}()
24      for d in 1:length(end_hmm.B)
25        push!(euclideans, euclidean(vec, end_hmm.B[d].p))
26      end
27      min_euc_idx = findmin(euclideans)[2]
28      while min_euc_idx in ref_idxs
29        euclideans[min_euc_idx]=1.
30        min_euc_idx = findmin(euclideans)[2]
31      end
32      push!(ref_idxs,findmin(euclideans)[2])
33    end
34
35    for step in chain
36      hmm=step[2]
37      push!(coords[c+1],
38        ↪ (hmm.B[ref_idxs[1]].p[1],hmm.B[ref_idxs[2]].p[1],hmm.B[ref_idxs[3]].p
39      end
40    end
41
42  return coords
43 end

```

---

### 16.6.18 /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
22 function ProgressHMM{T}(thresh;
23                           dt::Real=0.1,
24                           desc::AbstractString="Progress: ",
25                           color::Symbol=:green,
26                           output::IO=stderr,
27                           offset::Integer=0,
28                           start_it::Integer=1) where T
29
30   tfirst = tlast = time()
31   printed = false
32   new{T}(thresh, dt, typemax(T), start_it, false, tfirst, tlast,
33         → printed, desc, color, output, 0, offset, 0.0)
34 end
35
36 ProgressHMM(thresh::Real, dt::Real=0.1, desc::AbstractString="Progress:
37   → ",
38   color::Symbol=:green, output::IO=stderr;
39   offset::Integer=0, start_it::Integer=1) =
40     → ProgressHMM{typeof(thresh)}(thresh, dt=dt, desc=desc,
41     → color=color, output=output, offset=offset,
42     → start_it=start_it)
43
44 ProgressHMM(thresh::Real, desc::AbstractString, offset::Integer=0,
45   → start_it::Integer=1) = ProgressHMM{typeof(thresh)}(thresh, desc=desc,
46   → offset=offset, start_it=start_it)

```

```

35
36 function update!(p::ProgressHMM, val, steptime; options ... )
37     p.val = val
38     p.counter += 1
39     p.steptime = steptime
40     updateProgress!(p; options ... )
41 end
42
43 function updateProgress!(p::ProgressHMM; showvalues = Any[], valuecolor =
44     :blue, offset::Integer = p.offset, keep = (offset == 0))
45     p.offset = offset
46     t = time()
47     if p.val ≤ p.thresh && !p.triggered
48         p.triggered = true
49         if p.printed
50             p.triggered = true
51             dur = ProgressMeter.durationstring(t-p.tfirst)
52             msg = @sprintf "%s Time: %s (%d iterations)" p.desc dur
53             → p.counter
54             print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
55             ProgressMeter.move_cursor_up_while_clearing_lines(p.output,
56             → p.numprintedvalues)
57             ProgressMeter.printover(p.output, msg, p.color)
58             ProgressMeter.printvalues!(p, showvalues; color = valuecolor)
59             if keep
60                 println(p.output)
61             else
62                 print(p.output, "\r\u1b[A" ^ (p.offset +
63                 → p.numprintedvalues))
64             end
65         end
66         return
67     end
68     if t > p.tlast+p.dt && !p.triggered
69         elapsed_time = t - p.tfirst
70         msg = @sprintf "%s (convergence: %g ⇒ %g, iterate: %g, step
71             → time: %s)" p.desc p.val p.thresh p.counter hmss(p.steptime)
72         print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
73         ProgressMeter.move_cursor_up_while_clearing_lines(p.output,
74             → p.numprintedvalues)
75         ProgressMeter.printover(p.output, msg, p.color)
76         ProgressMeter.printvalues!(p, showvalues; color = valuecolor)
77     end

```

```

72     print(p.output, "\r\u1b[A" ^ (p.offset + p.numprintedvalues))
73     # Compensate for any overhead of printing. This can be
74     # especially important if you're running over a slow network
75     # connection.
76     p.tlast = t + 2*(time()-t)
77     p.printed = true
78   end
79 end
80
81   function hmss(dt)
82     isnan(dt) && return "NaN"
83     (h,r) = divrem(dt,60*60)
84     (m,r) = divrem(r, 60)
85     (isnan(h)||isnan(m)||isnan(r)) && return "NaN"
86     string(Int(h),":",Int(m),":",Int(ceil(r)))
87   end

```

---

### 16.6.19 /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
14 end
15
16   #function to construct BHMM transition matrix with strong
17   ↪ priors on auto-transition
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

```

```

15         transition_matrix[k,:] =
16             ↳ rand(Dirichlet(dirichlet_params))
17     end
18     return transition_matrix
19 end
20
21 #function to construct BHMM state emission distribution from
22     ↳ uninformative dirichlet over the alphabet size
23 function generate_emission_dist(no_emission_symbols,
24     ↳ prior=Dirichlet(ones(no_emission_symbols)/no_emission_symbols))
25     return Categorical(rand(prior))
26 end

```

---

### 16.6.20 /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 function assert_loadconfig(k_range,o_range)
14     k_range[1] < 1 && throw(ArgumentError("Minimum value of LoadConfig
15         ↳ k_range is 1!"))
16     o_range[1] < 0 && throw(ArgumentError("Minimum value of LoadConfig
17         ↳ o_range is 0!"))
18     return true
19 end
20
21 #subfunc to handle balancing memory load on dissimilar machines in
22     ↳ cluster
23 function load_balancer(no_models::Integer, hmm_jobs::RemoteChannel,
24     ↳ config::LoadConfig, timeout::Integer=300)
25     jobid::Chain_ID, start_iterate::Integer, hmm::BHMM,
26     ↳ job_norm::AbstractFloat, observations::Matrix = take!(hmm_jobs)

```

```

21     starttime=time()
22     while (jobid.K < config.k_range[1] || jobid.K > config.k_range[end]
23         || jobid.order < config.o_range[1] || jobid.order >
24         config.o_range[end] || jobid in config.blacklist ||
25         (length(config.whitelist)>0 && !(jobid in config.whitelist))) &&
26         (time()-starttime ≤ timeout) #while a job prohibited by load
27         table, keep putting the job back and drawing a new one
28         put!(hmm_jobs, (jobid, start_iterate, hmm, job_norm,
29             observations))
30         jobid, start_iterate, hmm, job_norm, observations =
31             take!(hmm_jobs)
32     end
33     (time()-starttime > timeout) ? (return 0,0,0,0,0) : (return jobid,
34         start_iterate, hmm, job_norm, observations)
35 end

```

---

### 16.6.21 /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
7      function lps(base, adjuvants ... )
8          prob = base+sum(adjuvants) ; isnan(prob) ? -Inf :
9              prob
10     end

```

---

### 16.6.22 /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

```

```

7     dummy_p=state.p.+10^-99 #prevent -Inf or NaN from zero probability
   ↵ symbols
8     infoscore=(bits+sum(x*log(2,x) for x in dummy_p))
9     infovec=[x*infoscore for x in dummy_p]
10    infovec./=bits
11    print("<<< ")
12    for (symbol,symbol_prob) in enumerate(infovec)
13        if symbol_prob ≥ .05
14            str=string(alphabet.integers[symbol])
15            if symbol_prob ≥ .7
16                for char in str
17                    printstyled(stdout, char; bold=true,
   ↵ color=cs_dict[char])
18            end
19            elseif .7 > symbol_prob ≥ .25
20                for char in str
21                    printstyled(stdout, char; color=cs_dict[char])
22                end
23            elseif .25 > symbol_prob
24                for char in str
25                    printstyled(stdout, lowercase(char);
   ↵ color=cs_dict[char])
26                end
27            end
28            print(" ")
29        end
30    end
31    println(">>>")
32 end

```

---

### 16.6.23 /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
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
49         ← the sequences in trainable format
50         code_dict[(obs_id, order)] = coded_seqs
51     end

```

```

49     return code_dict
50 end
51
52 #from a vector of LongSequences, get
53 function get_order_n_seqs(seqs::Vector{LongSequence{DNAAlphabet{2}}}),
54     ↪ order_no::Integer, base_tuple::Tuple=ACGT)
55     kmer_vecs = Vector{Vector{Mer{DNAAlphabet{2}}}}()
56     length_vec = Vector{Integer}()
57     window = order_no + 1
58
59     for seq in seqs
60         kmer_vec = Vector{Mer{DNAAlphabet{2}}}()
61         @inbounds for (i, kmer) in
62             ↪ collect(each(Mer{DNAAlphabet{2}}, window), seq))
63             push!(kmer_vec, kmer)
64         end
65
66         push!(kmer_vecs, kmer_vec)
67         push!(length_vec, length(seq))
68     end
69
70     return Nordseqs = N_Order_ntSequence(CompoundAlphabet(base_tuple,
71         ↪ order_no), length_vec, kmer_vecs)
72 end
73
74 #convert tuple kmers to symbol codes
75 function code_seqs(input::N_Order_ntSequence, offsets::Vector=[0 for i in
76     ↪ 1:length(input.order_kmers)]; sorted::Bool=false)
77     symbol_no=length(input.alphabet.symbols)
78     if symbol_no ≤ typemax(UInt8)
79         integer_type = UInt8
80     elseif typemax(UInt8) < symbol_no < typemax(UInt16)
81         integer_type = UInt16
82     else
83         integer_type = UInt32
84     end
85
86     alphabet = input.alphabet
87     output = zeros(integer_type, length(input.order_kmers),
88         ↪ (maximum([length(seq) for seq in input.order_kmers])+1)) #leave 1
89         ↪ missing value after the longest sequence for indexing sequence
90         ↪ length in CLHMM messages
91     sorted && (sort_idxs = sortperm(input.seq_lengths, rev=true))

```

```

85     sorted ? input_seqs = input.order_kmers[sort_idxs] : input_seqs =
86             ↵ input.order_kmers
87
88     for (i, seq) in enumerate(input_seqs)
89         for t in 1:length(seq)
90             curr_kmer = seq[t]
91             curr_code = alphabet.symbols[curr_kmer]
92             output[i,t+offsets[i]]=curr_code
93         end
94     end
95     return output
96 end

```

---

### 16.6.24 /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 function split_obs_sets(sample_dfs::Dict{String,DataFrame})
9     training_sets = Dict{String,Vector{LongSequence{DNAAlphabet{2}}}}()
10    test_sets = Dict{String,Vector{LongSequence{DNAAlphabet{2}}}}()
11
12    for (partition_id, partition) in sample_dfs
13        partition = partition[shuffle(1:size(partition, 1)),:] #shuffle
14            ↵ the samples to avoid any effect from sampling without
15            ↵ replacement
16        partition.sampleLength = (partition.SampleEnd -
17            ↵ partition.SampleStart) .+ 1
18        midway = sum(partition.sampleLength)÷2
19        split_index = 0
20        counter = 0
21        while split_index == 0
22            ↵ counter += 1
23            ↵ length_sum = sum(partition.sampleLength[1:counter])
24            ↵ if length_sum > midway
25                ↵     split_index = counter
26            ↵ end
27        end
28    end
29
30 end

```

```

23     training_sets[partition_id] =
24         ↳ partition.SampleSequence[1:split_index-1]
25     test_sets[partition_id] =
26         ↳ partition.SampleSequence[split_index:end]
27   end
28   return training_sets, test_sets
29 end

```

---

### 16.6.25 /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(O,N); β1oi = zeros(O,N); Eoi = zeros(O,N,B); Toij =
8         ↳ zeros(O,N,N); Aoi = zeros(O,N); log_pobs=zeros(O); yt=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        #RECURRANCE
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]; yt = 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

```

```

27         Tijt[i, j, m] = logsumexp([lps(a[m,n], TijT[i, j,
28             ↵ n], b[n,Γ]) for n in 1:N])
29         i==m && (Tijt[i, j, m] = logaddexp(Tijt[i, j, m],
30             ↵ lps(βoi_T[j], a[m,j], b[j, Γ])))
31     end
32     for y in 1:B
33         Eit[y, i, m] = logsumexp([lps(b[n,Γ], a[m,n],
34             ↵ EiT[y, i, n]) for n in 1:N])
35         i==m && y==yt && (Eit[y, i, m] = logaddexp(Eit[y,
36             ↵ i, m], βoi_t[m])))
37     end
38     end
39
40     βoi_T = βoi_t; TijT = Tijt; EiT = Eit
41 end
42
43 #TERMINATION
44
45     Γ = observations[1,o]
46     α1oi[o,:]= [lps(a[i], b[i, Γ]) for i in 1:N]
47     β1oi[o,:]= βoi_T
48     log_pobs[o]= logsumexp(lps.(α1oi[o,:], βoi_T[:]))
49     Eoi[o,:,:]= [logsumexp([lps(EiT[y,i,m], a[m], b[m,yt]) for m in
50             ↵ 1:N]) for i in 1:N, y in 1:B]
51     Toij[o,:,:,:]= [logsumexp([lps(TijT[i,j,m], a[m], b[m,yt]) for m
52             ↵ in 1:N]) for i in 1:N, j in 1:N]
53 end
54
55 #INTEGRATE ACROSS OBSERVATIONS AND SOLVE FOR NEW BHMM PARAMS
56
57     obs_penalty=log(0)
58     a_o=α1oi.+β1oi.-logsumexp.(eachrow(α1oi.+β1oi))
59     new_a=logsumexp.(eachcol(a_o))-obs_penalty
60
61     a_int = Toij.-logsumexp.([Toij[o,i,:] for o in 1:O, i in 1:N])
62     new_a = logsumexp.([a_int[:,i,j] for i in 1:N, j in
63             ↵ 1:N])-obs_penalty
64
65     e_int=Eoi.-logsumexp.([Eoi[o,j,:] for o in 1:O, j in 1:N])
66     new_b=logsumexp.([e_int[:,j,d] for d in 1:B, j in 1:N])-obs_penalty
67     new_D=[Categorical(exp.(new_b[:,i])) for i in 1:N]
68
69
70     return typeof(hmm)(exp.(new_a), exp.(new_a), new_D), lps(log_pobs)
71 end

```

```

63
64 #HMMBase
65 function mouchet_mle_step(hmm::BHMM, observations) where F
66     # NOTE: This function works but there is room for improvement.
67
68     log_likelihoods = mouchet_log_likelihoods(hmm, observations)
69
70     log_α = mouchet_messages_forwards_log(hmm.a, hmm.A, log_likelihoods)
71     log_β = mouchet_messages_backwards_log(hmm.A, log_likelihoods)
72     log_A = log.(hmm.A)
73
74     normalizer = logsumexp(log_α[1,:] + log_β[1,:])
75
76     # 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     typeof(hmm)(new_a, new_A, B), normalizer
106 end
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)
116   # OPTIMIZE
117   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
126 end
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   end
139   log_betas
139 end
```

---

### 16.6.26 /test/runtests.jl

---

```

1 using
  ↵ BioBackgroundModels,BioSequences,DataFrames,Distributed,Distributions,FASTX,Progr
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 !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" begin
97     @test isapprox(lps([.1, .1, .1]), .3)
98     @test lps([-Inf, .1, .1]) == -Inf
99     @test isapprox(lps(.1, .1, .1), .3)
100    @test lps(-Inf, .1, .1) == -Inf
101    @test lps(.1, -Inf, .1) == -Inf
102 end
103
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     no_input_hmms, chains, input_channel, output_channel =
129       ↪ setup_EM_jobs!(job_ids,obs_sets)
130     @test load_balancer(no_input_hmms, input_channel, K_exclusive_config,
131       ↪ 3) = (0,0,0,0,0)
132
133     no_input_hmms, chains, input_channel, output_channel =
134       ↪ setup_EM_jobs!(job_ids,obs_sets)
135     @test load_balancer(no_input_hmms, input_channel, O_exclusive_config,
136       ↪ 3) = (0,0,0,0,0)
137
138     no_input_hmms, chains, input_channel, output_channel =
139       ↪ setup_EM_jobs!(job_ids,obs_sets)
140     high_set=load_balancer(no_input_hmms, input_channel, high_config)
141
142     @test high_set[1]=Chain_ID("test",6,2,1)
143     @test high_set[2]=1

```

```

142 @test typeof(high_set[3])=BHMM{Float64}
143
144 @test
145     → high_set[4]=obs_lh_given_hmm(high_set[5],high_set[3],linear=false)
146 @test typeof(high_set[5])=Matrix{UInt8}
147 end
148
149 @testset "utilities/observation_coding.jl Order coding structs and
150     → functions" begin
151     compound_alphabet=CompoundAlphabet(ACGT, 2)
152     @test
153         → typeof(compound_alphabet.symbols)=Dict{Mer{DNAAlphabet{2}},Integer}
154     @test length(compound_alphabet.symbols)=64
155     @test compound_alphabet.symbols[Mer{DNAAlphabet{2}}("AAA")]=1
156     @test compound_alphabet.symbols[Mer{DNAAlphabet{2}}("TTT")]=64
157
158     test_seqs =
159         → [BioSequences.LongSequence{DNAAlphabet{2}}("ACGTACGTACGTACGT"),BioSequences.L
160     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 4 0 0 0 0 0
161         → 0 0 0 0 0]
162     target2= [37 58 15 20 37 58 15 20 37 58 15 20 37 58 0; 64 64 64 64 64
163         → 0 0 0 0 0 0 0 0 0]
164
165     order0_seqs = get_order_n_seqs(test_seqs,0)
166     @test typeof(order0_seqs)=BioBackgroundModels.N_Order_ntSequence
167     for (mer, code) in CompoundAlphabet(ACGT,0).symbols
168         @test order0_seqs.alphabet.symbols[mer]==code
169     end
170     @test order0_seqs.seq_lengths==[16,7]
171     @test order0_seqs.order_kmers[1][1]==Mer{DNAAlphabet{2}}("A")
172     code0_seqs = code_seqs(order0_seqs)
173     @test target0 == code0_seqs
174
175     order2_seqs = get_order_n_seqs(test_seqs,2)
176     @test typeof(order2_seqs)=BioBackgroundModels.N_Order_ntSequence
177     for (mer, code) in CompoundAlphabet(ACGT,2).symbols
178         @test order2_seqs.alphabet.symbols[mer]==code
179     end
180     @test order2_seqs.seq_lengths==[16,7]
181     @test order2_seqs.order_kmers[1][1]==Mer{DNAAlphabet{2}}("ACG")
182     code2_seqs = code_seqs(order2_seqs)
183     @test target2 == code2_seqs
184
185     bw_o0_seqs = get_order_n_seqs(test_seqs[end:-1:1], 0)

```

```

179     sorted_code_seqs = code_seqs(bw_o0_seqs, sorted=true)
180     @test target0 == sorted_code_seqs
181
182     obs_sets=Dict("test1"⇒test_seqs,
183                   "test2"⇒test_seqs)
184
185     → job_ids=[Chain_ID("test1",4,0,1),Chain_ID("test2",4,0,1),Chain_ID("test1",2,2)
186
187     code_dict=code_job_obs(job_ids,obs_sets)
188     @test ("test1",0) in keys(code_dict)
189     @test ("test1",2) in keys(code_dict)
190     @test ("test2",0) in keys(code_dict)
191     @test !(("test2",2) in keys(code_dict))
192     @test code_dict[("test1",0)]==target0
193     @test code_dict[("test1",2)]==target2
194     @test code_dict[("test2",0)]==target0
195
196     #test type selection for high order numbers
197     o5_nos=get_order_n_seqs(test_seqs,5)
198     o5_codes=code_seqs(o5_nos)
199     @test typeof(o5_codes)==Matrix{UInt16}
200
201     o7_nos=get_order_n_seqs(test_seqs,7)
202     o7_codes=code_seqs(o7_nos)
203     @test typeof(o7_codes)==Matrix{UInt32}
204
205 end
206
207 @testset "genome_sampling/partition_masker.jl functions" begin
208     synthetic_seq =
209         → BioSequences.LongSequence{DNAAlphabet{2}}(generate_synthetic_seq())
210     position_start = 501
211     position_length=141
212     position_pad=350
213     perigenic_pad = 250
214
215     position_df = make_padded_df(posfasta, gff, genome, index,
216         → position_pad)
217     add_partition_masks!(position_df, gff, perigenic_pad)
218
219     @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+position_
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
225     mm = position_df.MaskMatrix[1]
226     idx=1#pos 151
227     @test sum(mm[idx:idx+99,1] .== 1)==length(mm[idx:idx+99,1])
228     idx+=100#pos 251
229     @test sum(mm[idx:idx+259,1] .== 2)==length(mm[idx:idx+259,1])
230     idx+=260 #pos 511
231     @test sum(mm[idx:idx+59,1] .== 3)==length(mm[idx:idx+59,1])
232     idx+=60#pos 571
233     @test sum(mm[idx:idx+29,1] .== 2)==length(mm[idx:idx+29,1])
234     idx+=30#pos601
235     @test sum(mm[idx:idx+40,1] .== 3)==length(mm[idx:idx+40,1])
236
237     #test exception
238     position_df = make_padded_df(posfasta, gff, genome, index,
239     ↳ position_pad)
240
241     partition_coords_dict=partition_genome_coordinates(gff,
242     ↳ perigenic_pad)
243
244     ↳ partitioned_scaffolds=divide_partitions_by_scaffold(partition_coords_dict)
245     scaffold_coords_dict = Dict{String,DataFrame}()
246     scaffold="1"
247     for ((partition, part_scaffold), df) in partitioned_scaffolds
248         if scaffold == part_scaffold
249             scaffold_coords_dict[partition] = df
250         end
251     end
252
253     @test_throws DomainError
254     ↳ mask_sequence_by_partition(1001,151,scaffold_coords_dict)
255 end
256
257 @testset "genome_sampling/sequence_sampler.jl functions &
258     ↳ API/genome_sampling.jl interface" begin
259     #rectifier tests

```

```

250 @test rectify_identifier("1")=="1"
251
252 partitions = 3 #exonic, periexonic, intragenic
253 sample_set_length=100
254 min_sample_window=5
255 max_sample_window=25
256 perigenic_pad=250
257 syn_intergenic_starts = [1]
258 syn_intergenic_ends = [250]
259 syn_periexonic_starts = [251,571,661,761,861,961]
260 syn_periexonic_ends = [510,600,700,800,900,1000]
261 syn_periexonic_strands = ['-', '-', '--', '--', '--', '-']
262 syn_exonic_starts = [511,601,701,801,901]
263 syn_exonic_ends = [570,660,760,860,960]
264 syn_exonic_strands = ['+', '--', '--', '--', '--']
265 partition_lengths =
266   ↳ Dict("exon"⇒5*60,"intergenic"⇒250,"periexonic"⇒450)
267
268 # "Testing sequence sampler fns ... "
269 synthetic_seq =
270   ↳ BioSequences.LongSequence{DNAAlphabet{2}}(generate_synthetic_seq())
271
272 # "Partitioning synthetic genome coordinates ... "
273 coordinate_partitions = partition_genome_coordinates(gff,
274   ↳ perigenic_pad)
275
276 @test coordinate_partitions["intergenic"].Start =
277   ↳ syn_intergenic_starts
278 @test coordinate_partitions["intergenic"].End = syn_intergenic_ends
279
280 @test coordinate_partitions["periexonic"].Start =
281   ↳ syn_periexonic_starts
282 @test coordinate_partitions["periexonic"].End = syn_periexonic_ends
283 @test coordinate_partitions["periexonic"].Strand =
284   ↳ syn_periexonic_strands
285
286 @test coordinate_partitions["exon"].Start = syn_exonic_starts
287 @test coordinate_partitions["exon"].End = syn_exonic_ends
288 @test coordinate_partitions["exon"].Strand = syn_exonic_strands
289
290 # "Checking sampling functions at all synthetic indices ... "

```

```

286     input_test_channel, completed_test_channel, progress_channel,
287     ↵ setlength = setup_sample_jobs(genome, index, gff,
288     ↵ sample_set_length, min_sample_window, max_sample_window,
289     ↵ perigenic_pad; deterministic=true)
290     while isready(input_test_channel)
291         genome_path, genome_index_path, partition_df, partitionid,
292         ↵ sample_set_length, sample_window_min, sample_window_max,
293         ↵ deterministic = take!(input_test_channel)
294
295         for feature in eachrow(partition_df)
296             seqid, scaffold_start, scaffold_end =
297                 ↵ meta_to_feature_coord(feature.MetaStart, feature.MetaEnd,
298                 ↵ partition_df)
299             @test scaffold_start == feature.Start && scaffold_end ==
300                 ↵ feature.End
301         end
302
303         stranded = get_strand_dict()[partitionid]
304         @test typeof(stranded) == Bool
305
306         scaffold_sequence_record_dict = build_scaffold_seq_dict(genome,
307             ↵ index)
308         @test scaffold_sequence_record_dict["1"] == synthetic_seq
309
310         partition_length = partition_df.MetaEnd[end]
311         @test partition_length == partition_lengths[partitionid]
312
313         metacoordinate_bitarray = trues(partition_df.MetaEnd[end])
314
315         for bitindex in findall(metacoordinate_bitarray)
316             feature_metaStart, feature_metaEnd, strand =
317                 ↵ get_feature_params_from_metacoord(bitindex, partition_df,
318                 ↵ stranded)
319             @test 1 ≤ feature_metaStart < feature_metaEnd ≤
320                 ↵ length(metacoordinate_bitarray)
321             @test feature_metaStart in partition_df.MetaStart
322             @test feature_metaEnd in partition_df.MetaEnd
323             if partitionid == "periexonic"
324                 @test strand == '-'
325             elseif partitionid == "exon"
326                 @test strand in ['-','+']
327             end
328             feature_length = length(feature_metaStart:feature_metaEnd)

```

```

317         window = determine_sample_window(feature_metaStart,
318                                         → feature_metaEnd, bitindex, metacoordinate_bitarray,
319                                         → sample_window_min, sample_window_max) #get an appropriate
320                                         → sampling window around the selected index, given the
321                                         → feature boundaries and params
322
323     @test 1 ≤ feature_metaStart ≤ window[1] <
324         → window[1]+sample_window_min-1 <
325         → window[1]+sample_window_max-1 ≤ window[2] ≤
326         → feature_metaEnd ≤ length(metacoordinate_bitarray)
327
328     sample_scaffoldid, sample_scaffold_start, sample_scaffold_end
329         → = meta_to_feature_coord(window[1],window[2],partition_df)
330
331     @test sample_scaffoldid = "1"
332
333     @test 1 ≤ sample_scaffold_start ≤
334         → sample_scaffold_start+sample_window_min ≤
335         → sample_scaffold_end ≤
336         → min(sample_scaffold_start+sample_window_max,1000) ≤ 1000
337
338
339     strand = '-' ?
340         → target_seq=reverse_complement(synthetic_seq[sample_scaffold_start:sample_scaffold_end])
341         → :
342
343     target_seq =
344         → synthetic_seq[sample_scaffold_start:sample_scaffold_end]
345
346
347     proposal_sequence = fetch_sequence(sample_scaffoldid,
348                                         → scaffold_sequence_record_dict, sample_scaffold_start,
349                                         → sample_scaffold_end, strand; deterministic=deterministic)
350                                         → #get the sequence associated with the sample window
351
352
353     @test proposal_sequence == target_seq
354
355 end
356
357
358 # "Verifying sampling channels ... "
359
360
361 input_sample_channel, completed_sample_channel, progress_channel,
362     → setlength = setup_sample_jobs(genome, index, gff,
363     → sample_set_length, min_sample_window, max_sample_window,
364     → perigenic_pad; deterministic=true)
365 get_sample_set(input_sample_channel, completed_sample_channel,
366     → progress_channel)
367
368
369 #collect sample dfs by partition id when ready
370 collected_counter = 0

```

```

339     sample_record_dfs = Dict{String,DataFrame}()
340     while collected_counter < partitions
341         wait(completed_sample_channel)
342         partition_id, sample_df = take!(completed_sample_channel)
343         sample_record_dfs[partition_id] = sample_df
344         collected_counter += 1
345     end
346
347     #get_sample entire feature selection
348     coordinate_partitions = partition_genome_coordinates(gff,
349     ↳ perigenic_pad)
350     partition_id="exon"
351     partition=coordinate_partitions[partition_id]
352     add_metacoordinates!(partition)
353     metacoordinate_bitarray=true.(partition.MetaEnd[end])
354     scaffold_sequence_record_dict = build_scaffold_seq_dict(genome,
355     ↳ index)
356
357     #test fetch_sequence strand char ArgumentError
358     @test_throws ArgumentError fetch_sequence("1",
359     ↳ Dict{String,LongSequence}("1"⇒LongSequence{DNAAlphabet{2}}("AAAAAAAAAAAAAA"))
360     ↳ 1, 5, 'L')
361
362     #test get_feature_row_index ArgumentError
363     @test_throws DomainError get_feature_row_index(partition,0)
364
365     @test length(get_sample(metacoordinate_bitarray, 1, 250, partition,
366     ↳ scaffold_sequence_record_dict)[6])=60
367
368     synthetic_seq =
369     ↳ BioSequences.LongSequence{DNAAlphabet{2}}(generate_synthetic_seq())
370     for (partid, df) in sample_record_dfs
371         for sample in eachrow(df)
372             target_seq =
373             ↳ synthetic_seq[sample.SampleStart:sample.SampleEnd]
374             strand = sample.Strand
375             if sample.Strand == '-'
376                 target_seq = reverse_complement(target_seq)
377             end
378             @test sample.SampleSequence == target_seq
379         end
380     end
381
382 
```

```

375 #execute_sample_jobs API
376 wkpool=addprocs(1)
377 @everywhere using BioBackgroundModels
378
379 channels = setup_sample_jobs(genome, index, gff, sample_set_length,
380   ↳ min_sample_window, max_sample_window, perigenic_pad;
381   ↳ deterministic=true)
380 records=execute_sample_jobs(channels, wkpool)
381
382 rmprocs(wkpool)
383 end
384
385 @testset "likelihood_funcs/hmm.jl & bg_lh_matrix.jl functions" begin
386     pvec = [.4,.3,.2,.1]
387     trans = ones(1,1)
388     B = [Categorical(pvec)]
389     hmm = BHMM(trans, B)
390
391     testseq=zeros(Int64,1,5)
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
401     pvec=[.25,.25,.25,.25]
402     trans = [.9 .1
403               .1 .9]
404     B = [Categorical(pvec), Categorical(pvec)]
405     hmm = BHMM(trans, B)
406
407     @test
408         ↳ isapprox(obs_lh_given_hmm(testseq,hmm),obs_lh_given_hmm(testseq,hmm,linear=false))
409     @test isapprox(sum(get_BGHMM_symbol_lh(testseq,hmm)),
410         ↳ obs_lh_given_hmm(testseq,hmm))
411     @test isapprox(sum(get_BGHMM_symbol_lh(testseq,hmm)),
412         ↳ obs_lh_given_hmm(testseq,hmm,linear=false))
413
414     testseq=zeros(Int64,1,1001)
415     testseq[1,1:1000]=rand(1:4,1000)

```

```

410
411     @test isapprox(sum(get_BGHMM_symbol_lh(testseq,
412         → hmm)),obs_lh_given_hmm(testseq,hmm))
413
414     Dex = [Categorical([.3, .1, .3, .3]),Categorical([.15, .35, .35,
415         → .15])]
416     Dper = [Categorical([.15, .35, .35, .15]),Categorical([.4, .1, .1,
417         → .4])]
418     Dint = [Categorical([.4, .1, .1, .4]),Categorical([.45, .05, .05,
419         → .45])]
420     BGHMM_dict = Dict{String,BHMM}()
421     BGHMM_dict["exon"] = BHMM(trans, Dex)
422     BGHMM_dict["perixonic"] = BHMM(trans, Dper)
423     BGHMM_dict["intergenic"] = BHMM(trans, Dint)
424
425     position_length=141;perigenic_pad=250;
426     position_df = make_padded_df(posfasta, gff, genome, index,
427         → position_length)
428
429     reader=open(FASTA.Reader, genome, index=index)
430     seq=FASTA.sequence(reader["CM002885.2.1"])
431     @test position_df.Start[1]==501
432     @test position_df.End[1]==641
433     @test position_df.PadSeq[1]==seq[360:641]
434     @test position_df.PadStart[1]==360
435     @test position_df.RelStart[1]==141
436     @test position_df.SeqOffset[1]==0
437
438     offset_position_df = make_padded_df(posfasta, gff, genome, index,600)
439     @test offset_position_df.Start[1]==501
440     @test offset_position_df.End[1]==641
441     @test offset_position_df.PadSeq[1]==seq[1:641]
442     @test offset_position_df.PadStart[1]==1
443     @test offset_position_df.RelStart[1]==500
444     @test offset_position_df.SeqOffset[1]==100
445
446     add_partition_masks!(position_df, gff, perigenic_pad)
447     @test all(position_df.MaskMatrix[1][1:151,1]==2)
448     @test all(position_df.MaskMatrix[1][152:211,1]==3)
449     @test all(position_df.MaskMatrix[1][212:241,1]==2)
450     @test all(position_df.MaskMatrix[1][242:end,1]==3)
451     @test all(position_df.MaskMatrix[1][1:151,2]==-1)
452     @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
467 @testset "EM/baum-welch.jl MS_HMMBase equivalency and functions" begin
468     A = [.5 .5
469             .5 .5]
470     B = [Categorical(ones(4)/4), Categorical([.7,.1,.1,.1])]
471     hmm = BHMM(A, B)
472     log_A = log.(hmm.A)
473
474     obs = zeros(UInt8, 22,1)
475     obs[1:21] = [4,3,3,2,3,2,1,1,2,3,3,3,4,4,2,3,2,3,4,3,2]
476     obs_lengths=[21]
477
478     lls = BioBackgroundModels.bw_llhs(hmm,obs)
479     m_lls = mouchet_log_likelihoods(hmm,obs[1:21])
480
481     K,Tmaxplus1,0 = size(lls)
482     T=Tmaxplus1-1
483
484     #TEST LOG LIKELIHOOD FN
485     @test transpose(lls[:,1:end-1,1]) == m_lls
486
487     log_α = BioBackgroundModels.messages_forwards_log(hmm.a, hmm.A, lls,
488             → obs_lengths)
489     m_log_α = mouchet_messages_forwards_log(hmm.a, hmm.A, m_lls)
490
491

```

```

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,O) to (T,O,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:O
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)

```

```

529     end
530
531     ξ = exp.(log_ξ)
532     k_ξ = sum(ξ, dims=[3,4])
533     nan_mask = k_ξ .== 0; k_ξ[nan_mask] .= Inf #prevent NaNs in dummy
      ↵ renorm arising from multiple sequences indexing
534     ξ ./= k_ξ #dummy renorm across K to keep numerical creep from
      ↵ causing isprobvec to fail on new new_A during hmm creation
535
536     m_ξ = exp.(m_log_ξ)
537     m_ξ ./= sum(m_ξ, dims=[2,3])
538
539     #check equivalency of xi calc methods
540     @test isapprox(reshape(ξ,Tmaxplus1,K,K)[1:21,:,:],m_ξ)
541
542     γ = exp.(log_γ)
543     k_γ = sum(γ, dims=3); k_γ[nan_mask[:, :, :]] .= Inf #prevent NaNs in
      ↵ dummy renorm
544     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(γ,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(γ[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_y[:,i]))
588 end
589
590 obs_mask = .!nan_mask
591 obs_collection = obs[obs_mask[:, :]]
592
593 B = Distribution{F}[]
594 @inbounds for (i, d) in enumerate(hmm.B)
595     # Super hacky ...
596     # https://github.com/JuliaStats/Distributions.jl/issues/809
597     γ_d = y[:, :, i]
598     push!(B, fit_mle(eval(typeof(d).name.name), obs_collection,
599         ↳ γ_d[obs_mask[:, :]]))
600     #slowest call by orders of magnitude
601 end
602
603 #test maximization equivalency
604 for (d, dist) in enumerate(B)
605     @test isapprox(dist.support, m_D[d].support)
606     @test isapprox(dist.p, m_D[d].p)
607 end
608
609 # "Testing mle_step ... "
610
611

```

```

609  #verify that above methods independently produce equivalent output,
  ↵  and that this is true of multiple identical obs, but not true of
  ↵  different obs sets
610  mouchet_hmm = mouchet_mle_step(hmm, obs[1:21])
611
612  new_hmm = BioBackgroundModels.bw_step(hmm, obs, obs_lengths)
613
614  dblobs = zeros(UInt8, 22,2)
615  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]
616  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]
617  dblobs_lengths=[21,21]
618  dbl_hmm = BioBackgroundModels.bw_step(hmm, dblobs, dblobs_lengths)
619
620  otherobs = dblobs
621  otherobs[1:21,2] = [1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1]
622  other_hmm = BioBackgroundModels.bw_step(hmm, otherobs,
  ↵  dblobs_lengths)
623
624  for n in fieldnames(typeof(new_hmm[1]))
625      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),
  ↵  getfield(mouchet_hmm[1],n))
640          @test isapprox(getfield(new_hmm[1],n),
  ↵  getfield(dbl_hmm[1],n))
641          @test !isapprox(getfield(new_hmm[1],n),
  ↵  getfield(other_hmm[1],n))
642      end
643  end
644
645  @test new_hmm[2] == mouchet_hmm[2] ≠ dbl_hmm[2] ≠ other_hmm[2]

```

```

646
647 # "Testing fit_mle! ... "
648
649 # "test fit_mle! function"
650 input_hmms= RemoteChannel(()→Channel{Tuple}(1))
651 output_hmms = RemoteChannel(()→Channel{Tuple}(30))
652 chainid=Chain_ID("Test",2,0,1)
653 put!(input_hmms, (chainid, 2, hmm, 0.0, transpose(obs)))
654 BioBackgroundModels.EM_converge!(input_hmms, output_hmms, 1;
655     → EM_func=BioBackgroundModels.bw_step, max_iterates=4,
656     → verbose=true)
657 wait(output_hmms)
658 workerid, jobid, iterate, hmm3, log_p, epsilon, converged =
659     → take!(output_hmms)
660 @test jobid == chainid
661 @test iterate == 3
662 @test assert_hmm(hmm3.a, hmm3.A, hmm3.B)
663 @test size(hmm3) == size(hmm) == (2,4)
664 @test log_p < 1
665 @test log_p == obs_lh_given_hmm(transpose(obs),hmm, linear=false)
666 @test converged == false
667 wait(output_hmms)
668 workerid, jobid, iterate, hmm4, log_p, epsilon, converged =
669     → take!(output_hmms)
670 @test jobid == chainid
671 @test iterate == 4
672 @test assert_hmm(hmm4.a, hmm4.A, hmm4.B)
673 @test size(hmm4) == size(hmm) == (2,4)
674 @test log_p < 1
675 @test log_p == obs_lh_given_hmm(transpose(obs),hmm3,linear=false)
676 @test converged == false
677
678 # "Test convergence.. "
679 obs=zeros(UInt8, 101, 2)
680 for i in 1:size(obs)[2]
681     obs[1:100,i]=rand(1:4,100)
682 end
683 input_hmms= RemoteChannel(()→Channel{Tuple}(1))
684 output_hmms = RemoteChannel(()→Channel{Tuple}(30))
685 put!(input_hmms, (chainid, 2, hmm, 0.0, transpose(obs)))
686 BioBackgroundModels.EM_converge!(input_hmms, output_hmms, 1;
687     → EM_func=BioBackgroundModels.bw_step, delta_thresh=.05,
688     → max_iterates=100, verbose=true)

```

```

683     wait(output_hmms)
684     workerid, jobid, iterate, hmm4, log_p, epsilon, converged =
685         → take!(output_hmms)
686     while isready(output_hmms)
687         workerid, jobid, iterate, hmm4, log_p, epsilon, converged =
688             → take!(output_hmms)
689     end
690     @test converged==1
691
692     #test t_add_categorical_counts DimensionMismatch
693     @test_throws DimensionMismatch t_add_categorical_counts!(zeros(4),
694         → zeros(UInt8,2,3), zeros(2,2))
695
696 end
697
698
699 @testset "EM/churbanov.jl Baum-Welch equivalency and functions" begin
700     # "Setting up for MLE function tests.."
701     A = fill((1/6),6,6)
702     B = [Categorical(ones(4)/4),
703         → Categorical([.7,.05,.15,.1]),Categorical([.15,.35,.4,.1]),
704         → Categorical([.6,.15,.15,.1]),Categorical([.1,.4,.4,.1]),
705         → Categorical([.2,.2,.3,.3])]
706     hmm = BHMM(A, B)
707     log_A = log.(hmm.A)
708
709     obs = zeros(Int64,1,250)
710     obs[1:249] = rand(1:4,249)
711     obs_lengths=[249]
712     # "Testing mle_step ... "
713
714     #verify that above methods independently produce equivalent output,
715     → and that this is true of multiple identical obs, but not true of
716     → different obs sets
717     mouchet_hmm = mouchet_mle_step(hmm, obs[1:249])
718
719     new_hmm = linear_step(hmm, obs, obs_lengths)
720
721     ms_sng = BioBackgroundModels.bw_step(hmm, Array(transpose(obs)),
722         → obs_lengths)
723
724     dblobs = zeros(Int64, 2,250)
725     dblobs[1,1:249] = obs[1:249]
726     dblobs[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,
722         → Array(transpose(dbllobs)),dblobs_lengths)
723
724     otherobs = deepcopy(dbllobs)
725     otherobs[2,1:249] = rand(1:4,249)
726     if otherobs[1,1]==otherobs[2,1]
727         otherobs[1,1]=1&&(otherobs[2,1]==2)
728         otherobs[1,1]=2&&(otherobs[2,1]==3)
729         otherobs[1,1]=3&&(otherobs[2,1]==4)
730         otherobs[1,1]=4&&(otherobs[2,1]==1)
731     end
732
733     other_hmm = linear_step(hmm, otherobs, dbllobs_lengths)
734
735     ms_hmm = BioBackgroundModels.bw_step(hmm,
736         → Array(transpose(otherobs)),dblobs_lengths)
737
738     for n in fieldnames(typeof(new_hmm[1]))
739         if n == :B
740             for (d, dist) in enumerate(new_hmm[1].B)
741                 @test dist.support==mouchet_hmm[1].B[d].support
742                 @test isapprox(dist.p,mouchet_hmm[1].B[d].p)
743                 @test dist.support==ms_sng[1].B[d].support
744                 @test isapprox(dist.p,ms_sng[1].B[d].p)
745                 @test dist.support==dbl_hmm[1].B[d].support
746                 @test isapprox(dist.p,dbl_hmm[1].B[d].p)
747                 @test dist.support==other_hmm[1].B[d].support
748                 @test ms_hmm[1].B[d].support==other_hmm[1].B[d].support
749                 @test !isapprox(dist.p, other_hmm[1].B[d].p)
750                 @test isapprox(ms_hmm[1].B[d].p, other_hmm[1].B[d].p)
751             end
752         elseif n == :partition
753             @test getfield(new_hmm[1],n) == getfield(mouchet_hmm[1],n)
754             @test getfield(new_hmm[1],n) == getfield(dbl_hmm[1],n)
755             @test getfield(new_hmm[1],n) == getfield(other_hmm[1],n)
756         else
757

```

```

758         @test isapprox(getfield(new_hmm[1],n),
759                         getfield(mouchet_hmm[1],n))
760
761         @test isapprox(getfield(new_hmm[1],n),
762                         getfield(dbl_hmm[1],n))
763
764         @test isapprox(getfield(new_hmm[1],n), getfield(ms_dbl[1],n))
765
766     end
767
768 end
769
770 @test ms_sng[2] == new_hmm[2] == mouchet_hmm[2] != dbl_hmm[2] ==
771     → ms_dbl[2] != other_hmm[2] == ms_hmm[2]
772
773 # "Testing fit_mle! ... "
774
775 #test fit_mle! function
776 input_hmms= RemoteChannel(()→Channel{Tuple}(1))
777 output_hmms = RemoteChannel(()→Channel{Tuple}(30))
778 chainid=Chain_ID("Test",6,0,1)
779 put!(input_hmms, (chainid, 2, hmm, 0.0, obs))
780 BioBackgroundModels.EM_converge!(input_hmms, output_hmms, 1;
781     → max_iterates=4, verbose=true)
782 wait(output_hmms)
783 workerid, jobid, iterate, hmm3, log_p, delta, converged =
784     → take!(output_hmms)
785 @test jobid == chainid
786 @test iterate == 3
787 @test BioBackgroundModels.assert_hmm(hmm3.a, hmm3.A, hmm3.B)
788 @test size(hmm3) == size(hmm) == (6,4)
789 @test log_p < 1
790 @test log_p == linear_likelihood(obs, hmm)
791 @test converged == false
792 wait(output_hmms)
793 workerid, jobid, iterate, hmm4, log_p, delta, converged =
794     → take!(output_hmms)
795 @test jobid == chainid
796 @test iterate == 4

```

```

793     @test BioBackgroundModels.assert_hmm(hmm4.a, hmm4.A, hmm4.B)
794     @test size(hmm4) == size(hmm) == (6,4)
795     @test log_p < 1
796     @test log_p == linear_likelihood(obs, hmm3)
797     @test converged == false

798
799     # "Test convergence.."
800     obs=zeros(Int64, 4, 1001)
801     for o in 1:size(obs)[1]
802         obsl=rand(100:1000)
803         obs[o,1:obsl]=rand(1:4,obsl)
804     end
805     input_hmms= RemoteChannel(()→Channel{Tuple}(1))
806     output_hmms = RemoteChannel(()→Channel{Tuple}(Inf))
807     put!(input_hmms, (chainid, 2, hmm, 0.0, obs))
808     BioBackgroundModels.EM_converge!(input_hmms, output_hmms, 1;
809         → delta_thresh=.05, max_iterates=100, verbose=true)
810     wait(output_hmms)
811     workerid, jobid, iterate, hmm4, log_p, delta, converged =
812         → take!(output_hmms)
813     while isready(output_hmms)
814         workerid, jobid, iterate, hmm4, log_p, delta, converged =
815             → take!(output_hmms)
816     end
817     @test converged==1
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

```

```

830     channels = setup_sample_jobs(genome, index, gff, sample_set_length,
831         ↳ min_sample_window, max_sample_window, perigenic_pad;
832         ↳ deterministic=true)
833     sample_record_dfs=execute_sample_jobs(channels, wkpool)
834     training_sets, test_sets = split_obs_sets(sample_record_dfs)
835
836
837     job_ids=Vector{Chain_ID}()
838     for (obs_id, obs) in training_sets, K in Ks, order in order_nos, rep
839         ↳ in 1:replicates
840             push!(job_ids, Chain_ID(obs_id, K, order, rep))
841     end
842
843     no_input_hmms, chains, input_hmms, output_hmms =
844         ↳ setup_EM_jobs!(job_ids, training_sets)
845     @test no_input_hmms ==
846         ↳ replicates*length(Ks)*length(order_nos)*length(training_sets)
847
848     while isready(input_hmms)
849         jobid, start_iterate, hmm, last_norm, observations =
850             ↳ take!(input_hmms)
851         @test last_norm == linear_likelihood(observations, hmm)
852         obs_lengths = [findfirst(iszero, observations[o,:])-1 for o in
853             ↳ 1:size(observations)[1]]
854         #make sure input HMMs are valid and try to mle_step them and
855             ↳ ensure their 1-step children are valid
856         @test assert_hmm(hmm.a, hmm.A, hmm.B)
857         new_hmm, prob = linear_step(hmm, observations, obs_lengths)
858         @test assert_hmm(hmm.a, hmm.A, hmm.B)
859         @test prob < 0
860     end
861
862     genome_reader = open(FASTA.Reader, genome, index=index)
863
864         ↳ seq=LongSequence{DNAAlphabet{2}}(FASTA.sequence(genome_reader["CM002885.2.1"]))
865     seqdict = Dict("test"⇒[seq for i in 1:3])
866     seqdict["blacklist"] = seqdict["test"]
867
868     job_ids=Vector{Chain_ID}()
869     push!(job_ids, Chain_ID("blacklist", 1, 0, 1))
870     push!(job_ids, Chain_ID("blacklist", 1, 0, 2))

```

```

864 Ks=[1,2]; order_nos=[0,1]
865 for K in Ks, order in order_nos, replicate in 1:2
866     push!(job_ids, Chain_ID("test", K, order, replicate))
867 end
868
869 wkpool=addprocs(2, topology=:master_worker)
870 @everywhere using BioBackgroundModels
871
872 load_dict=Dict{Int64,LoadConfig}()
873 for (n,wk) in enumerate(wkpool)
874     n==1 &&
875         (load_dict[wk]=LoadConfig(1:2,0:1,blacklist=[Chain_ID("blacklist",1,0,1)])
876     n==2 && (load_dict[wk]=LoadConfig(1:2,0:1))
877 end
878 #test work resumption, load configs
879
880 em_jobset=setup_EM_jobs!(job_ids, seqdict, delta_thresh=10000.)
881 execute_EM_jobs!(wkpool, em_jobset..., "testchains",
882     → delta_thresh=10000., verbose=true, load_dict=load_dict)
883
884 for (chain_id,chain) in em_jobset[2]
885     @test chain[end].converged = true
886     @test chain[end].delta ≤ 10000
887     @test typeof(chain[end].hmm)=BHMM{Float64}
888     @test 1 ≤ chain[end].iterate ≤ 5000
889 end
890
891 wkpool=addprocs(2, topology=:master_worker)
892 @everywhere using BioBackgroundModels
893
894 em_jobset=setup_EM_jobs!(job_ids, seqdict,
895     → chains=deserialize("testchains"), delta_thresh=.1)
896 execute_EM_jobs!(wkpool, em_jobset..., "testchains", delta_thresh=.1,
897     → verbose=true)
898
899 for (chain_id,chain) in em_jobset[2]
900     @test chain[end].converged = true
901     @test chain[end].delta ≤ .1
902     @test typeof(chain[end].hmm)=BHMM{Float64}
903     @test 1 ≤ chain[end].iterate ≤ 5000
904 end
905
906 #test for already converged warning

```

```

903     wkpool=addprocs(2, topology=:master_worker)
904     @everywhere using BioBackgroundModels
905     @test_throws ArgumentError execute_EM_jobs!(wkpool, em_jobset..., 
906         → "testchains", delta_thresh=.01, verbose=true)
907
908     rm("testchains")
909
910     @test_throws ArgumentError
911         → generate_reports(Dict{Chain_ID,Vector{EM_step}}(),seqdict)
912     @test_throws ArgumentError
913         → generate_reports(em_jobset[2],Dict{String,Vector{LongSequence{DNAAlphabet{2}}}}
914
915     @test_throws ArgumentError
916         → BioBackgroundModels.report_chains(em_jobset[2],Dict{String,Vector{LongSequence{DNAAlphabet{2}}}})
917
918     report_folders=generate_reports(em_jobset[2],seqdict)
919     folder=report_folders["test"]
920     @test "test"==folder.partition_id
921     show(folder.partition_report)
922     show(folder.replicate_report)
923     for id in folder.partition_report.best_repset
924         show(folder.chain_reports[id])
925     end
926
927     end
928
929     rm(genome)
930     rm(index)
931     rm(gff)
932     rm(posfasta)

```

---

### 16.6.27 /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")

```

```

8 end
9
10 function print_synthetic_position(path::String)
11     write(path, ">1 start 501 end 641 smt_pos 111 smt_value 205.11821
12     ↳ fuzziness_score
13     ↳ 43.53698\n", generate_synthetic_seq()[501:641], "\n")
14 end
15
16
17
18
19
20
21
22 1 Ensembl chromosome 1 1000 .
23 ###
24 1 ensembl_havana gene 501 1000 . - .
25 ↳ gene
26 ↳ [Source:FAKESRC];gene_id=ENSDARG00000000001;logic_name=ensembl_havana_gene;version=1
27 1 ensembl_havana mRNA 501 1000 .
28 1 ensembl_havana three_prime_UTR 501 510 .
29 1 ensembl_havana exon 501 570 .
30 1 ensembl_havana CDS 511 570 .
31 1 ensembl_havana exon 601 660 .
32 1 ensembl_havana CDS 601 660 .
33 1 ensembl_havana exon 701 760 .
34 1 ensembl_havana CDS 701 760 .
35 1 ensembl_havana exon 801 860 .
36 1 ensembl_havana CDS 801 860 .
37 1 ensembl_havana five_prime_UTR 901 975 .
38 ###
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
49     @assert length(iseq) == 2
50     @assert length(pseq) == 2
51     @assert length(eseq) == 3
52     @assert mod(gene_start-1,2) == 0
53     @assert mod(exon_length,3) == 0
54     @assert mod(intron_length,2) == 0
55     gene_length=gene_start+UTR3L+(no_exons*exon_length)+((no_exons-1)*intron_length)
56     @assert gene_length ≤ OAL
57     seq=""
58
59     for i in 1:((gene_start-1) / 2)
60         seq *= iseq
61     end
62     verbose && @info "Intergenic length $(length(seq))"
63
64     for i in 1:UTR3L / 2
65         seq *= pseq
66     end
67     verbose && @info "Added 3'UTR ... $(length(seq))"
68
69     for ex in 1:no_exons-1
70         for i in 1:(exon_length/3)
71             seq*=eseq
72         end
73         ex == 1 ? ilength = 30 : ilength = 40
74         for i in 1:(ilength/2)
75             seq*=pseq
76         end
77     end
78     for i in 1:(exon_length/3)
79         seq*=eseq
80     end
81     verbose && @info "Added exons and introns ... $(length(seq))"
82
83     for i in 1:UTR5L / 2
84         seq *= pseq
85     end
86     verbose && @info "Added 5'UTR ... $(length(seq))"
87
88     for i in 1:((OAL-length(seq))/2)
89         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

```

---

## 16.7 BioMotifInference

### 16.7.1 /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 #both are needed as logsumexp
6     ↪  for two terms is deprecated
7     import Random: rand, seed!, shuffle!
8     import Distances: euclidean
9
10    #CONSTANTS AND PERMUTE FUNCTION ARGUMENT DEFAULTS GIVING RISE TO
10    ↪  IMPLEMENTATION-SPECIFIC SAMPLING EFFECTS
11    global MOTIF_EXPECT=1. #motif expectation- this value to be divided
11    ↪  by obs lengths to obtain penalty factor for scoring
12
13    global REVCOMP=true #calculate scores on both strands?
14
15    global TUNING_MEMORY=20 #coefficient multiplied by Permute_Instruct
15    ↪  function call limit to give total number of calls remembered by
15    ↪  tuner per function

```

```

16   global CONVERGENCE_MEMORY=500 #number of iterates to display for
    ↳ convergence interval history
17
18   global SRC_PERM_FREQ=.5 #frequency with which random_decorrelate will
    ↳ permute a source
19
20   global PWM_SHIFT_DIST=Weibull(.5,.1) #distribution of weight matrix
    ↳ permutation magnitudes
21   global PWM_SHIFT_FREQ=.2 #proportion of positions in source to
    ↳ permute weight matrix
22   global PWM_LENGTHPERM_FREQ=.2 #proportion of sources to permute
    ↳ length
23   global LENGTHPERM_RANGE=1:3
24
25   global PRIOR_WT=3. #estimate prior dirichlets from product of this
    ↳ constant and sample "mle" wm
26   global PRIOR_LENGTH_MASS=.8
27
28   global EROSION_INFO_THRESH=1.
29
30   global CONSOLIDATE_THRESH=.035
31
32   include("IPM/ICA_PWM_Model.jl")
33   export Model_Record
34   export ICA_PWM_Model
35   include("IPM/IPM_likelihood.jl")
36   include("IPM/IPM_prior_utilities.jl")
37   assemble_source_priors
38   include("ensemble/IPM_Engine.jl")
39   export IPM_Engine, assemble_IPMs
40   include("permutation/permute_utilities.jl")
41   include("permutation/orthogonality_helper.jl")
42   include("permutation/permute_functions.jl")
43   export full_perm_funcvec
44   include("permutation/permute_control.jl")
45   export Permute_Instruct
46   include("permutation/Permute_Tuner.jl")
47   include("ensemble/ensemble_utilities.jl")
48   export ensemble_history, reset_ensemble!, move_ensemble!,
    ↳ copy_ensemble!, rewind_ensemble, complete_evidence, get_model,
    ↳ show_models
49   include("utilities/model_display.jl")
50   include("utilities/worker_diagnostics.jl")

```

```

51 include("utilities/ns_progressmeter.jl")
52 include("utilities/synthetic_genome.jl")
53 include("utilities/worker_sequencer.jl")
54 export synthetic_sample
55 include("nested_sampler/nested_step.jl")
56 include("nested_sampler/converge_ensemble.jl")
57 export converge_ensemble!
58
59 end # module

```

---

### 16.7.2 /src/IPM/ICA\_PWM\_Model.jl

```

1 struct Model_Record #record struct to associate a log_Li with a saved,
2   ↳ calculated model
3   path::String
4   log_Li::AbstractFloat
5 end
6
6 struct ICA_PWM_Model #Independent component analysis position weight
7   ↳ matrix model
8   name::String #designator for saving model to posterior
9   origin::String #functions instantiating IPMs should give an
10    ↳ informative desc of the means by which the model was generated
11   sources::Vector{Tuple{<:AbstractMatrix{<:AbstractFloat},<:Integer}}
12    ↳ #vector of PWM signal sources (LOG PROBABILITY!!!) tupled with an
13    ↳ index denoting the position of the first PWM base on the prior
14    ↳ matrix- allows us to permute length and redraw from the
15    ↳ appropriate prior position
16   source_length_limits::UnitRange{<:Integer} #min/max source lengths
17    ↳ for init and permutation
18   mix_matrix::BitMatrix # obs x sources bool matrix
19   log_Li::AbstractFloat
20   permute_blacklist::Vector{Function} #blacklist of functions that
21    ↳ ought not be used to permute this model (eg. because to do so
22    ↳ would not generate a different model for IPMs produced from
23    ↳ fitting the mix matrix)
24   function ICA_PWM_Model(name, origin, sources, source_length_limits,
25     ↳ mix_matrix, log_Li, permute_blacklist=Vector{Function}())
26     verify_srcs(name, origin, sources)
27     new(name, origin, sources, source_length_limits, mix_matrix,
28       ↳ log_Li, permute_blacklist)
29   end

```

```

18 end
19
20 #ICA_PWM_Model FUNCTIONS
21 ICA_PWM_Model(name::String,
22   ↳ source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat},<:BitMatrix},<:AbstractFloat}}, 
23   ↳ mix_prior::Tuple{BitMatrix,<:AbstractFloat},
24   ↳ bg_scores::AbstractArray{<:AbstractFloat},
25   ↳ observations::AbstractArray{<:Integer},
26   ↳ source_length_limits::UnitRange{<:Integer}) = init_IPM(name,
27   ↳ source_priors,mix_prior,bg_scores,observations,source_length_limits)
28
29 function verify_srcs(name, ori,
30   ↳ sources::Vector{<:Tuple{<:AbstractMatrix{<:AbstractFloat},<:Integer}})
31   for (n,(pwm,pi)) in enumerate(sources)
32     !all(isprobvec(exp.(pwm[pos,:])) for pos in 1:size(pwm,1)) &&
33     ↳ throw(DomainError("Bad probvec in model $name, source $n,
34     ↳ origin $(ori):$(exp.(pwm)))")
35   end
36 end
37
38
39 #MODEL INIT
40 function init_IPM(name::String,
41   ↳ source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat},<:BitMatrix},<:AbstractFloat}}, 
42   ↳ mix_prior::Tuple{BitMatrix,<:AbstractFloat},
43   ↳ bg_scores::AbstractArray{<:AbstractFloat},
44   ↳ observations::AbstractArray{<:Integer},
45   ↳ source_length_limits::UnitRange{<:Integer})
46   assert_obs_bg_compatibility(observations,bg_scores)
47   T,O = size(observations)
48   S=length(source_priors)
49   obs_lengths=[findfirst(iszero,observations[:,o])-1 for o in
50     ↳ 1:size(observations)[2]]
51   sources=init_logPWM_sources(source_priors, source_length_limits)
52   mix=init_mix_matrix(mix_prior,O,S)
53   log_lh = IPM_likelihood(sources, observations, obs_lengths,
54     ↳ bg_scores, mix)
55
56
57   return ICA_PWM_Model(name, "init_IPM", sources, source_length_limits,
58     ↳ mix, log_lh)
59 end
60
61           #init_IPM SUBFUNCS
62           function assert_obs_bg_compatibility(obs, bg_scores)
63             T,O=size(obs)

```

```

44
45     t,o=size(bg_scores)
46     O!=o && throw(DomainError("Background scores and
47         observations must have same number of observation
48         columns!"))
49     T!=t+1 && throw(DomainError("Background score array
50         must have the same observation lengths as
51         observations!"))
52 end
53
54 function
55     → init_logPWM_sources(prior_vector::AbstractVector{<:Union{<:Abstract
56     → source_length_limits::UnitRange{<:Integer}})
57     srcvec = Vector{Tuple{Matrix{Float64},Int64}}()
58     for prior in prior_vector
59         if
60             → typeof(prior)<:AbstractVector{<:Dirichlet{<:AbstractF
61             if length(prior) <
62                 → source_length_limits[1]
63                 length_dist=DiscreteNonParametric(
64                     [source_length_limits ... ],
65
66                         → [PRIOR_LENGTH_MASS,ones(length(source_length
67                         PWM_length=rand(length_dist)
68                         elseif source_length_limits[1] <
69                             length(prior) <
70                             source_length_limits[end]
71                             length_dist=DiscreteNonParametric(
72                                 [source_length_limits ... ],
73
74                                     → [ones(length(prior))-source_length_limit
75                                         → (ones(source_length_limits[end]-length
76                                         ]
77                                         )
78                                         → (ones(source_length_limits[end]-length
79                                         ]
80                                         )
81                                         )
82                                         PWM_length=rand(length_dist)
83                                         else
84                                         PWM_length=rand(source_length_limits)
85                                         end
86
87                                         if PWM_length>length(prior)
88                                             → prior_coord=rand(-(PWM_length-length(prior)):
```

```

73         else
74             ↪ prior_coord=rand(1:length(prior)-PWM_length+1)
75         end
76
77         PWM = zeros(PWM_length,4)
78
79         curr_pos=prior_coord
80         for pos in 1:PWM_length
81             curr_pos < 1 || curr_pos >
82                 ↪ length(prior) ?
83                 ↪ dirichlet=Dirichlet(ones(4)/4) :
84                 ↪ dirichlet=prior[curr_pos]
85             PWM[pos, :] = rand(dirichlet)
86         end
87         push!(srcvec, (log.(PWM), prior_coord))
88             ↪ #push the source PWM to the source
89             ↪ vector with the prior coord idx to
90             ↪ allow drawing from the appropriate
91             ↪ prior dirichlets on permuting source
92             ↪ length
93     elseif prior==false
94         PWM_length=rand(source_length_limits)
95         PWM=zeros(PWM_length,4)
96         dirichlet=Dirichlet(ones(4)/4)
97         for pos in 1:PWM_length
98             PWM[pos,:]=rand(dirichlet)
99         end
100        push!(srcvec, (log.(PWM), 1))
101    else
102        throw(ArgumentError("Bad prior supplied
103                         ↪ for ICA_PWM_Model!"))
104    end
105  end
106  return srcvec
107 end
108
109 function
110     ↪ init_mix_matrix(mix_prior:: Tuple{BitMatrix,<:AbstractFloat},
111     ↪ no_observations:: Integer, no_sources:: Integer)
112     inform, uninform=mix_prior
113     if size(inform,2) > 0

```

```

104     @assert size(inform,1)==no_observations &&
105         ← size(inform,2)≤ no_sources "Bad informative
106         ← mix prior dimensions!"
107     end
108
109     @assert 0.0 ≤ uninform ≤1.0 "Uninformative mix
110         ← prior not between 0.0 and 1.0!"
111     mix_matrix = falses(no_observations, no_sources)
112     if size(inform,2)>0
113         mix_matrix[:,1:size(inform,2)]=inform
114     end
115     for index in
116         ← CartesianIndices((1:no_observations,size(inform,2)+1:no_sources))
117         rand() ≤ uninform && (mix_matrix[index] = true)
118     end
119     return mix_matrix
120 end
121
122 function Base.show(io::IO, m::ICA_PWM_Model;
123     ← nsrc::Integer=length(m.sources), progress=false)
124     nsrc = 0 && (nsrc=length(m.sources))
125     nsrc>length(m.sources) && (nsrc=length(m.sources))
126     nsrc=length(m.sources) ? (srcstr="All") : (srcstr="Top $nsrc")
127
128     printidxs,printsrcs,printfreqs=sort_sources(m,nsrc)
129
130     printstyled(io, "ICA PWM Model $(m.name) w/ logLi $(m.log_Li)\n",
131         ← bold=true)
132     println(io, srcstr*" sources:")
133     for src in 1:nsrc
134         print(io, "S$(printidxs[src]), $(printfreqs[src]*100)%: ")
135         pwmstr_to_io(io, printsrcs[src])
136         println(io)
137     end
138     println(m.origin)
139
140     progress && return(nsrc+4)
141 end
142
143 function sort_sources(m, nsrc)
144     printidxs=Vector{Integer}__()
145     printsrcs=Vector{Matrix{Float64}}__()
146     printfreqs=Vector{Float64}__()

```

```

141     freqs=vec(sum(m.mix_matrix,dims=1)); total=size(m.mix_matrix,1)
142     sortfreqs=sort(freqs,rev=true); sortidxs=sortperm(freqs,rev=true)
143     for srcidx in 1:nsrc
144         push!(printidxs, sortidxs[srcidx])
145         push!(printsracs, m.sources[sortidxs[srcidx]][1])
146         push!(printfreqs, sortfreqs[srcidx]/total)
147     end
148     return printidxs, printsracs, printfreqs
149 end

```

---

### 16.7.3 /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(false(size(observations)[2])))
10        ↳ source_wmls = [size(source[1])[1] for source in sources]
11        ↳ 0 = size(bg_scores)[2]
12        ↳ source_stops = [obsl - wml + 1 for wml in source_wmls, obsl 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
18        ↳ obs_src_idxs = mix_pull_idxs(mix) #get vectors of sources emitting in
19            ↳ each obs
20
21
22        ↳ revcomp ? (srcs = cat(source[1], revcomp_pwm(source[1]), dims=3) for
23            ↳ source in sources; motif_expectations = [( (MOTIF_EXPECT / 2) / obsl)
24            ↳ for obsl in obs_lengths; mat_dim=2) : (srcs = [source[1] for
25            ↳ source in sources]; ; motif_expectations = [(MOTIF_EXPECT / obsl)
26            ↳ for obsl in obs_lengths; mat_dim=1) #setup appropriate reverse
27            ↳ complemented sources if necessary and set
28            ↳ log_motif_expectation - nMica has 0.5 per base for including the
29            ↳ reverse complement, 1 otherwise
30
31        ↳ lme_vec = zeros(length(sources))
32
33

```



```

46             penalty_sum > 1. && (penalty_sum=1.)
47             cardinality_penalty=log(1.0-penalty_sum)
48         else
49             cardinality_penalty=0.0
50         end
51
52         revcomp ? (obs_lhs[t][i]=weave_scores_ds!(weavevec,
53             ↳ lh_vec, obsl, view(bg_scores,:,:,o), score_matrices,
54             ↳ oidxs, mixwmls, log(motif_expectations[o]),
55             ↳ cardinality_penalty, osi_emitting)) :
56             (obs_lhs[t][i]=weave_scores_ss!(lh_vec, obsl,
57                 ↳ view(bg_scores,:,:,o), score_matrices, oidxs,
58                 ↳ mixwmls, log(motif_expectations[o]),
59                 ↳ cardinality_penalty, osi_emitting))
60
61             empty!(osi_emitting)
62
63         end
64     end
65     end
66
67     returncache ? (return lps([lps(obs_lhs[t]) for t in 1:nt]),
68         ↳ vcat(obs_lhs ...)) : (return lps([lps(obs_lhs[t]) for t in 1:nt]))
69 end
70
71 @inline function mix_pull_idxs(A::AbstractArray{Bool})
72     n=count(A)
73     S=[Vector{Int64}() for o in 1:size(A,1)]
74     cnt=1
75     for (i,a) in pairs(A)
76         if a
77             push!(S[i[1]],i[2])
78             cnt+=1
79         end
80     end
81     return S
82 end
83
84
85 @inline function
86     ↳ revcomp_pwm(pwm::AbstractMatrix{<:AbstractFloat}) #in
87     ↳ order to find a motif on the reverse strand, we scan
88     ↳ the forward strand with the reverse complement of the
89     ↳ pwm, reordered 3' to 5', so that eg. an PWM for an
90     ↳ ATG motif would become one for a CAT motif

```

```

77         return pwm[end:-1:1,end:-1:1]
78     end
79
80     @inline function score_sources_ds!(score_mat,
81         ← score_matrices::AbstractVector{<:AbstractMatrix{<:AbstractFloat}}}
82         ← sources, observation::AbstractVector{<:Integer},
83         ← source_stops)
84         for (s,source) in enumerate(sources)
85             for t in 1:source_stops[s]
86                 for position in 1:size(source,1)
87                     score_loc = t+position-1 #score_loc is
88                     ← the position of the obs to be scored
89                     ← by PWM
90                     score_mat[t,1] +=
91                     ← source[position,observation[score_loc],1]
92                     ← #add the appropriate log PWM value
93                     ← from the source to the score
94                     score_mat[t,2] +=
95                     ← source[position,observation[score_loc],2]
96                     ← #add the appropriate log PWM value
97                     ← from the source to the score
98
99             end
100         end
101         score_matrices[s]=score_mat[1:source_stops[s],:]
102         ← #copy score matrix to vector
103         score_mat.=0. #reset score matrix
104     end
105
106     @inline function score_sources_ss!(score_mat,
107         ← score_matrices::AbstractVector{<:AbstractArray{<:AbstractFloat}},}
108         ← sources, observation::AbstractVector{<:Integer},
109         ← source_stops)
110         for (s,source) in enumerate(sources)
111             for t in 1:source_stops[s]
112                 for position in 1:size(source,1)
113                     score_loc = t+position-1 #score_loc is
114                     ← the position of the obs to be scored
115                     ← by PWM

```

```

100                         score_mat[t] +=
101                         ↵ source[position,observation[score_loc]]
102                         ↵ #add the appropriate log PWM value
103                         ↵ from the source to the score
104
105                     end
106                 end
107             end
108
109             @inline function weave_scores_ds!(weavevec, lh_vec,
110                         ↵ obsl::Integer, bg_scores::SubArray,
111                         ↵ score_mat::AbstractVector{<:AbstractMatrix{<:AbstractFloat}},
112                         ↵ obs_source_indices::AbstractVector{<:Integer},
113                         ↵ source_wmls::AbstractVector{<:Integer},
114                         ↵ log_motif_expectation::AbstractFloat,
115                         ↵ cardinality_penalty::AbstractFloat, osi_emitting)
116             for i in 2:obsl+1 #i=1 is ithe lh_vec initializing 0,
117                         ↵ i=2 is the score of the first background position
118                         ↵ (ie t=1)
119                         t=i-1
120                         score = lps(lh_vec[i-1], bg_scores[t],
121                         ↵ cardinality_penalty)
122
123                         #logic: all observations are scored from t=wml to
124                         ↵ the end of the obs-therefore check at each
125                         ↵ position for new sources to add (indexed by
126                         ↵ vector position to retrieve source wml and
127                         ↵ score matrix)
128             if
129                         ↵ length(osi_emitting)<length(obs_source_indices)
130             for n in 1:length(obs_source_indices)
131                         if !(n in osi_emitting)
132                             t ≥ source_wmls[n] &&
133                             ↵ (push!(osi_emitting,n))
134                         end
135                     end
136             end
137
138             for n in osi_emitting

```

```

124             wml = source_wmls[n]
125             from_score = lh_vec[i-wml+1] #score at the
126             ↪ first position of the PWM
127             score_array = score_mat[n] #get the source
128             ↪ score matrix
129             score_idx = t - wml + 1 #translate t to
130             ↪ score_array idx for emission score
131             f_emit_score = score_array[score_idx,1]
132             ↪ #emission score at the last position of
133             ↪ the PWM
134             r_emit_score = score_array[score_idx,2]
135
136             weavevec *= score, lps(from_score,
137             ↪ f_emit_score, log_motif_expectation),
138             ↪ lps(from_score, r_emit_score,
139             ↪ log_motif_expectation)
140
141             score=logsumexp(weavevec)
142         end
143         lh_vec[i] = score
144     end
145     return lh_vec[obs1+1]
146 end
147
148 @inline function weave_scores_ss!(lh_vec, obs1::Integer,
149   ↪ bg_scores::SubArray,
150   ↪ score_mat::AbstractVector{<:AbstractVector{<:AbstractFloat}},
151   ↪ obs_source_indices::AbstractVector{<:Integer},
152   ↪ source_wmls::AbstractVector{<:Integer},
153   ↪ log_motif_expectation::AbstractFloat,
154   ↪ cardinality_penalty::AbstractFloat, osi_emitting)
155   for i in 2:obs1+1 #i=1 is ithe lh_vec initializing 0,
156   ↪ i=2 is the score of the first background position
157   ↪ (ie t=1)
158     t=i-1
159     score = lps(lh_vec[i-1], bg_scores[t],
160     ↪ cardinality_penalty)
161     #logic: all observations are scored from t=wml to
162     ↪ the end of the obs-therefore check at each
163     ↪ position for new sources to add (indexed by
164     ↪ vector position to retrieve source wml and
165     ↪ score matrix)

```

```

145             if
146                 → length(osi_emitting)<length(obs_source_indices)
147                     for n in 1:length(obs_source_indices)
148                         if !(n in osi_emitting)
149                             t ≥ source_wmls[n] &&
150                             → (push!(osi_emitting,n))
151                         end
152                     end
153             end
154
155             for n in osi_emitting
156                 wml = source_wmls[n]
157                 from_score = lh_vec[i-wml+1] #score at the
158                 → first position of the PWM
159                 score_array = score_mat[n] #get the source
160                 → score matrix
161                 score_idx = t - wml + 1 #translate t to
162                 → score_array idx for emission score
163                 f_emit_score = score_array[score_idx]
164                 → #emission score at the last position of
165                 → the PWM
166
167                 score=logaddexp(score, lps(from_score,
168                 → f_emit_score, log_motif_expectation))
169             end
170             lh_vec[i] = score
171         end
172     return lh_vec[obsL+1]
173 end

```

---

#### 16.7.4 /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]]))

```

```

8     prefix != "01" && prefix != "NA" && prefix != "PO" && prefix !=
9         → "://" && (wm=vcat(wm, transpose([parse(Float64,i) for i in
10        → split(line)[2:end]])))
11    prefix = "://" && push!(wms, wm)
12 end
13
14 #wm_samples are in decimal probability space, not log space
15 function assemble_source_priors(no_sources::Integer,
16     → wm_samples::AbstractVector{<:AbstractMatrix{<:AbstractFloat}},
17     → prior_wt::AbstractFloat=PRIOR_WT) #estimate a dirichlet prior on
18     → wm_samples inputs; if the number of samples is lower than the number
19     → of sources, return a false bool for init and permutation functions
20     source_priors = Vector{Union{Vector{Dirichlet{Float64}},Bool}}()
21     for source in 1:no_sources
22         if source ≤ length(wm_samples)
23             push!(source_priors,
24                 → estimate_dirichlet_prior_on_wm(wm_samples[source],
25                 → prior_wt))
26         else
27             push!(source_priors, false)
28         end
29     end
30     return source_priors
31 end

32
33     function
34         → estimate_dirichlet_prior_on_wm(wm::AbstractMatrix{<:AbstractFloat},
35         → wt::AbstractFloat=PRIOR_WT)
36         for i in 1:size(wm)[1]
37             !(isprobvec(wm[i,:])) && throw(DomainError("Bad
38                 → weight vec supplied to
39                 → estimate_dirichlet_prior_on_wm! $(wm[i,:])"))
40         end
41         prior = Vector{Dirichlet{Float64}}()
42         for position in 1:size(wm)[1]
43             normvec=wm[position,:]
44             zero_idxs=findall(isequal(0.),wm[position,:])
45             normvec[zero_idxs].+=10^-99
46             push!(prior, Dirichlet(normvec.*wt))
47         end
48         return prior

```

```

39         end
40
41 function cluster_mix_prior!(df::DataFrame,
42     ↪ wms::AbstractVector{<:AbstractMatrix{<:AbstractFloat}})
43     mix=falses(size(df,1),length(wms))
44     for (o, row) in enumerate(eachrow(df))
45         row.cluster ≠ 0 && (mix[o, row.cluster]=true)
46     end
47
48     represented_sources=unique(df.cluster)
49     return mix[:,represented_sources]
50 end
51
52 function infocenter_wms_trim(wm::AbstractMatrix{<:AbstractFloat},
53     ↪ trimsize::Integer)
54     !(size(wm,2)==4) && throw(DomainError("Bad wm! 2nd dimension should
55     ↪ be size 4"))
56     infovec=get_pwm_info(wm, logsw=false)
57     maxval, maxidx=findmax(infovec)
58     upstream_extension=Int(floor((trimsize-1)/2))
59     downstream_extension=Int(ceil((trimsize-1)/2))
60     1+upstream_extension+downstream_extension > size(wm,1) &&
61     ↪ throw(DomainError("Src too short for trim! $upstream_extension
62     ↪ $downstream_extension"))
63     return
64     ↪ wm[max(1,maxidx-upstream_extension):min(maxidx+downstream_extension,size(wm,1))]
65 end
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 end

```

```

71 function combine_filter_priors(target_src_no::Integer,
72     → target_src_size::Integer,
73     → prior_wms::Tuple{<:AbstractVector{<:AbstractMatrix{<:AbstractFloat}},AbstractVect
74     → prior_mix::Tuple{BitMatrix, BitMatrix})
75     wms=Vector{Matrix{Float64}}(undef, target_src_no)
76     cat_wms=vcat(prior_wms[1],prior_wms[2])
77     first_freq=[sum(prior_mix[1][:,s]) for s in 1:length(prior_wms[1])]
78     second_freq=[sum(prior_mix[2][:,s]) for s in 1:length(prior_wms[2])]
79     freqsort_idxes=sortperm(vcat(first_freq,second_freq))
80     for i in 1:target_src_no
81         target_src_idx=freqsort_idxes[i]
82         wms[i]=infocenter_wms_trim(cat_wms[target_src_idx],
83             → target_src_size)
84     end
85     return wms
86 end

```

---

### 16.7.5 /src/ensemble/IPM\_Engine.jl

```

1 mutable struct IPM_Engine
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 Txo
21    obs_lengths::Vector{Integer}

```



```

48     IPM_likelihood(init_logPWM_sources(source_priors,
49                     ↵   source_length_limits), obs, [findfirst(iszero,obs[:,o])-1 for
50                     ↵   o in 1:size(obs)[2]], bg_scores,
51                     ↵   falses(size(obs)[2],length(source_priors))))
52
53 IPM_Elbow(worker_pool::AbstractVector{<:Integer}, path::String,
54             ↵   no_models::Integer,
55             ↵   source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFlo
56             ↵   mix_prior::Tuple{BitMatrix,<:AbstractFloat},
57             ↵   bg_scores::AbstractMatrix{<:AbstractFloat}, obs::Array{<:Integer},
58             ↵   source_length_limits; posterior_switch::Bool=false} =
59 IPM_Elbow(
60             ↵   path,
61             ↵   distributed_IPM_assembly(worker_pool, path, no_models,
62                     ↵   source_priors, mix_prior, bg_scores, obs,
63                     ↵   source_length_limits)... ,
64                     [-Inf], #L0 = 0
65                     [0], #ie exp(0) = all of the prior is covered
66                     [-Inf], #w0 = 0
67                     [-Inf], #Liwi0 = 0
68                     [-1e300], #Z0 = 0
69                     [0], #H0 = 0,
70                     ↵   obs,
71                     ↵   [findfirst(iszero,obs[:,o])-1 for o in 1:size(obs)[2]],
72                     ↵   source_priors,
73                     ↵   mix_prior,
74                     ↵   bg_scores, #precalculated background score
75                     ↵   posterior_switch,
76                     ↵   Vector{String}(),
77                     ↵   no_models+1,
78                     ↵   IPM_likelihood(init_logPWM_sources(source_priors,
79                         ↵   source_length_limits), obs, [findfirst(iszero,obs[:,o])-1 for
80                             ↵   o in 1:size(obs)[2]], bg_scores,
81                             ↵   falses(size(obs)[2],length(source_priors))))
82
83 function assemble_IPMs(path::String, no_models::Integer,
84             ↵   source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFlo
85             ↵   mix_prior::Tuple{BitMatrix,<:AbstractFloat},
86             ↵   bg_scores::AbstractArray{<:AbstractFloat},
87             ↵   obs::AbstractArray{<:Integer},
88             ↵   source_length_limits::UnitRange{<:Integer})
89             ↵   ensemble_records = Vector{Model_Record}()
90             ↵   !isdir(path) && mkpath(path)

```

```

73
74     @assert size(obs)[2]==size(bg_scores)[2]
75
76     @showprogress 1 "Assembling IPM ensemble ... " for model_no in
77         1:no_models
78         model_path = string(path,'/',model_no)
79         if !isfile(model_path)
80             model = ICA_PWM_Model(string(model_no),
81             source_priors, mix_prior, bg_scores, obs,
82             source_length_limits)
83             serialize(model_path, model) #save the model to
84             the ensemble directory
85             push!(ensemble_records,
86                 Model_Record(model_path,model.log_Li))
87         else #interrupted assembly pick up from where we left off
88             model = deserialize(model_path)
89             push!(ensemble_records,
90                 Model_Record(model_path,model.log_Li))
91     end
92
93     return ensemble_records, minimum([record.log_Li for record in
94         ensemble_records])
95 end
96
97 function distributed_IPM_assembly(worker_pool::Vector{Int64},
98     path::String, no_models::Integer,
99     source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat}},<:BitMatrix,<:AbstractFloat}},
100    mix_prior::Tuple{BitMatrix,<:AbstractFloat},
101    bg_scores::AbstractArray{<:AbstractFloat},
102    obs ::AbstractArray{<:Integer},
103    source_length_limits ::UnitRange{<:Integer})
104     ensemble_records = Vector{Model_Record}()
105     !isdir(path) && mkpath(path)
106
107     @assert size(obs)[2]==size(bg_scores)[2]
108
109     model_chan=
110         RemoteChannel(()→Channel{ICA_PWM_Model}(length(worker_pool)))
111     job_chan = RemoteChannel(()→Channel{Union{Tuple,Nothing}}(1))
112     put!(job_chan,(source_priors, mix_prior, bg_scores, obs,
113         source_length_limits))

```



```

132                                     return counter
133                                         end
134                                         end
135                                         return counter
136                                         end
137
138                                         function
139                                         → worker_assemble(job_chan::RemoteChannel,
140                                         → models_chan::RemoteChannel,
141                                         → comms_chan::RemoteChannel)
142                                         put!(comms_chan,myid())
143                                         wait(job_chan)
144                                         params=fetch(job_chan)
145                                         while !(fetch(job_chan) ===
146                                         → nothing)
147                                         → model=ICA_PWM_Model(string(myid()),pa
148                                         → put!(models_chan,model)
149                                         end
150                                         end
151
152                                         function Base.show(io::IO, e::IPM_E
153                                         livec=[model.log_Li for model in e.models]
154                                         maxLH=maximum(livec)
155                                         printstyled(io, "ICA PWM Model Ensemble @ $(e.path)\n",
156                                         → bold=true)
157                                         msg = @sprintf "Contour: %3.6e MaxLH:%3.3e Max/Naive:%3.3e log
158                                         → Evidence:%3.6e" e.contour maxLH (maxLH-e.naive_lh)
159                                         → e.log_Zi[end]
160                                         println(io, msg)
161                                         hist=UnicodePlots.histogram(livec, title="Ense
162                                         → mbles Likelihood
163                                         → Distribution")
164                                         show(io, hist)
165                                         println()
166                                         progress && return(nrows(hist.graphics)+6)
167
168                                         end

```

---

### 16.7.6 /src/ensemble/ensemble\_utilities.jl

---

```

1 function ensemble_history(e::IPM_E
2   !e.sample_posterior && throw(ArgumentError("This ensemble has no
→ posterior samples to show a history for!"))

```

```

3    livec=vcat([model.log_Li for model in e.models],[model.log_Li for
4      ↵ model in e.posterior_samples])
5    show(histogram(livec, nbins=bins))
6
7 function e_backup(e::IPM_Ensemble, tuner::Permute_Tuner)
8     serialize(string(e.path,'/',"ens"), e)
9     serialize(string(e.path,'/',"tuner"), tuner)
10 end
11
12 function clean_ensemble_dir(e::IPM_Ensemble, model_pad::Integer;
13   ↵ 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   for file in readdir(e.path)
18     !(file in vcat([basename(model.path) for model in
19       ↵ e.models],"ens",[string(number) for number in
20         ↵ e.model_counter-length(e.models)-model_pad:e.model_counter-1]))
21     && rm(e.path*'*file)
22   end
23 end
24
25 function complete_evidence(e::IPM_Ensemble)
26   return final_logZ = logaddexp(e.log_Zi[end], (logsumexp([model.log_Li
27     ↵ for model in e.models]) + e.log_Xi[length(e.log_Li)] -
28     ↵ log(length(e.models))))
29 end
30
31 function reset_ensemble!(e::IPM_Ensemble)
32   new_e=deepcopy(e)
33   for i in 1:length(e.models)
34     if string(i) in [basename(record.path) for record in e.models]
35       new_e.models[i]=e.models[findfirst(isequal(string(i)),
36         ↵ [basename(record.path) for record in e.models])]
37     else
38
39       ↵ new_e.models[i]=e.posterior_samples[findfirst(isequal(string(i)),
40         ↵ [basename(record.path) for record in
41           ↵ e.posterior_samples])]
42     end
43   end
44 end

```

```

33     new_e.contour=minimum([record.log_Li for record in new_e.models])
34
35     new_e.log_Li=[new_e.log_Li[1]]
36     new_e.log_Xi=[new_e.log_Xi[1]]
37     new_e.log_wi=[new_e.log_wi[1]]
38     new_e.log_Liwi=[new_e.log_Liwi[1]]
39     new_e.log_Zi=[new_e.log_Zi[1]]
40     new_e.Hi=[new_e.Hi[1]]
41
42     new_e.posterior_samples=Vector{Model_Record}()
43
44     new_e.model_counter=length(new_e.models)+1
45
46     clean_ensemble_dir(new_e, 0; ignore_warn=true)
47     isfile(e.path*/"tuner") && rm(e.path*/"tuner")
48     serialize(e.path*/"ens", new_e)
49
50     return new_e
51 end
52
53 function move_ensemble!(e::IPM_Ensemble,path::String)
54     !isdir(path) && mkdir(path)
55     for file in readdir(e.path)
56         mv(e.path*/'*file,path*/'*file)
57     end
58
59     for (n,model) in enumerate(e.models)
60         e.models[n]=Model_Record(path*/'*basename(model.path),
61             ↪ model.log_Li)
62     end
63     if e.sample_posterior
64         for (n,model) in enumerate(e.posterior_samples)
65             ↪ e.posterior_samples[n]=Model_Record(path*/'*basename(model.path),
66             ↪ model.log_Li)
67         end
68     end
69     rm(e.path)
70     e.path=path
71     serialize(e.path*/"ens",e)
72     return e
73 end

```



```

110     push!(new_e.models, pop!(new_e.posterior_samples))
111   end
112   new_e.contour=new_e.log_Li[rewind_idx]
113   new_e.log_Li=new_e.log_Li[1:rewind_idx]
114   new_e.log_Xi=new_e.log_Xi[1:rewind_idx]
115   new_e.log_wi=new_e.log_wi[1:rewind_idx]
116   new_e.log_Liwi=new_e.log_Liwi[1:rewind_idx]
117   new_e.log_Zi=new_e.log_Zi[1:rewind_idx]
118   new_e.Hi=new_e.Hi[1:rewind_idx]
119
120   new_e.model_counter=length(new_e.models)+rewind_idx
121
122   return new_e
123 end
124
125 function show_models(e::IPM_Ensemble,idxs)
126   liperm=sortperm([model.log_Li for model in e.models],rev=true)
127   for idx in idxs
128     m=deserialize(e.models[liperm[idx]].path)
129     show(m)
130   end
131 end
132
133 function get_model(e::IPM_Ensemble,no)
134   return deserialize(e.path*"/"*string(no))
135 end

```

---

### 16.7.7 /src/nested\_sampler/converge\_ensemble.jl

```

1 function converge_ensemble!(e::IPM_Ensemble,
2   instruction::Permute_Instruct; max_iterates=typemax(Int64),
2   backup::Tuple{Bool, Integer}=(false,0),
2   clean::Tuple{Bool, Integer, Integer}=(false,0,0), verbose::Bool=false,
2   converge_criterion::String="standard",
2   converge_factor::AbstractFloat=.001, progargs ... )
2   N = length(e.models); curr_it=length(e.log_Li)
2
3   curr_it>1 && isfile(e.path*"/tuner") ?
3     (tuner=deserialize(e.path*"/tuner")) : (tuner =
3       Permute_Tuner(instruction)) #restore tuner from saved if any
5   wk_mon = Worker_Monitor([1])

```

```

6     meter = ProgressNS(e, wk_mon, tuner, 0.; start_it=curr_it,
7                           ↵ progargs ... )
8
9     converge_check = get_convfunc(converge_criterion)
10    while !converge_check(e, converge_factor) && (curr_it ≤
11        ↵ max_iterates)
12        warn,step_report = nested_step!(e, instruction) #step the
13        ↵ ensemble
14        warn == 1 && #"1" passed for warn code means no workers persist;
15        ↵ all have hit the permute limit
16        (@error "Failed to find new models, aborting at current
17        ↵ iterate."; return e) #if there is a warning, just
18        ↵ return the ensemble and print info
19        curr_it += 1
20
21        tune_weights!(tuner, step_report)
22        instruction = tuner.inst
23
24        backup[1] && curr_it%backup[2] == 0 && e_backup(e,tuner) #every
25        ↵ backup interval, serialise the ensemble and instruction
26        clean[1] && !e.sample_posterior && curr_it%clean[2] == 0 &&
27        ↵ clean_ensemble_dir(e,clean[3]) #every clean interval, remove
28        ↵ old discarded models
29
30        update!(meter, converge_check(e,converge_factor,vals=true) ... )
31    end
32
33    if converge_check(e,converge_factor)
34        final_logZ = complete_evidence(e)
35        @info "Job done, sampled to convergence. Final logZ $final_logZ"
36
37        e_backup(e,tuner)
38        clean[1] && !e.sample_posterior && clean_ensemble_dir(e,0) #final
39        ↵ clean
40        return final_logZ
41    elseif curr_it==max_iterates
42        @info "Job done, sampled to maximum iterate $max_iterates.
43        ↵ Convergence criterion not obtained."
44
45        e_backup(e,tuner)
46        clean[1] && !e.sample_posterior && clean_ensemble_dir(e,0) #final
47        ↵ clean
48        return e.log_Zi[end]

```

```

37     end
38 end
39
40 function converge_ensemble!(e::IPM_Engsemble,
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)
65     wk_mon=Worker_Monitor(wk_pool)
66     curr_it>1 && isfile(e.path*"/tuner") ?
67         → (tuner=deserialize(e.path*"/tuner")) : (tuner =
68             → Permute_Tuner(instruction)) #restore tuner from saved if any
69     meter = ProgressNS(e, wk_mon, tuner, 0.; start_it=curr_it,
70         → progargs ... )
71
72     while !converge_check(e, converge_factor) && (curr_it ≤
73         → max_iterates)
74         warn, step_report = nested_step!(e, model_chan, wk_mon) #step the
75             → ensemble
76         warn = 1 && #"1" passed for warn code means no workers persist;
77             → all have hit the permute limit

```

```

60             (@error "All workers failed to find new models, aborting
61             ↵ at current iterate."; return e) #if there is a
62             ↵ warning, just return the ensemble and print info
63 curr_it += 1
64 tune_weights!(tuner, step_report)
65 instruction = tuner.inst
66 take!(job_chan); put!(job_chan,(e.models,e.contour,instruction))
67 backup[1] && curr_it%backup[2] == 0 && e_backup(e,tuner) #every
68             ↵ backup interval, serialise the ensemble and instruction
69 clean[1] && !e.sample_posterior && curr_it%clean[2] == 0 &&
70             ↵ clean_ensemble_dir(e,clean[3]) #every clean interval, remove
71             ↵ old discarded models
72
73 update!(meter, converge_check(e,converge_factor,vals=true) ... )
74 end
75
76 take!(job_chan); put!(job_chan, (e.models, e.contour, "stop")) #stop
77             ↵ instruction terminates worker functions
78
79 if converge_check(e,converge_factor)
80     final_logZ = complete_evidence(e)
81     @info "Job done, sampled to convergence. Final logZ $final_logZ"
82
83 e_backup(e,tuner)
84 clean[1] && !e.sample_posterior && clean_ensemble_dir(e,0) #final
85             ↵ clean
86     return final_logZ
87 elseif curr_it==max_iterates
88     @info "Job done, sampled to maximum iterate $max_iterates.
89             ↵ Convergence criterion not obtained."
90
91     e_backup(e,tuner)
92     clean[1] && !e.sample_posterior && clean_ensemble_dir(e,0) #final
93             ↵ clean
94     return e.log_Zi[end]
95 end
96
97 end
98
99         function evidence_converge(e, evidence_fraction;
100             ↵ vals=false)
101             val=lps(findmax([model.log_Li for model in
102                             ↵ e.models])[1], e.log_Xi[end])
103             thresh=lps(log(evidence_fraction),e.log_Zi[end])

```

```

92             vals ? (return val, thresh) : (return val


---



```

### 16.7.8 /src/nested\_sampler/nested\_step.jl

```

1 ##### IMPLEMENTATION OF JEFF 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
4         ↳ posterior surface
5     i = length(e.log_Li) #iterate number, index for last values
6     j = i+1 #index for newly pushed values
7
8     e.contour, least_likely_idx = findmin([model.log_Li for model in
9         ↳ e.models])
10    Li_model = e.models[least_likely_idx]
11
12    #SELECT NEW MODEL, SAVE TO ENSEMBLE DIRECTORY, CREATE RECORD AND PUSH
13        ↳ TO ENSEMBLE
14    model_selected=false; step_report=0
15    while !model_selected
16        candidate,step_report=permute_IPM(e, instruction)
17        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),
20                 → candidate.log_Li);
21             e.sample_posterior && push!(e.posterior_samples,
22                 → Li_model)#if sampling posterior, push the model
23                 → record to the ensemble's posterior samples vector
24             deleteat!(e.models, least_likely_idx)
25             push!(e.models, new_model_record);
26
27             final_model=ICA_PWM_Model(string(e.model_counter),
28                 → candidate.origin, candidate.sources,
29                 → candidate.source_length_limits, candidate.mix_matrix,
30                 → candidate.log_Li, candidate.permute_blacklist)
31             serialize(new_model_record.path, final_model)
32
33             e.model_counter +=1
34         end
35     else
36         push!(e.models, Li_model)
37         return 1, step_report
38     end
39 end
40
41 #UPDATE ENSEMBLE QUANTITIES
42 push!(e.log_Li, minimum([model.log_Li for model in e.models])) #log
43             → likelihood of the least likely model - the current ensemble ll
44             → contour at Xi
45 push!(e.log_Xi, -i/N) #log Xi - crude estimate of the iterate's
46             → enclosed prior mass
47 push!(e.log_wi, lps(e.log_Xi[i], -((j+1)/N)-log(2))) #log width of
48             → prior mass spanned by the last step-trapezoidal approx
49 push!(e.log_Liwi, lps(e.log_Li[j],e.log_wi[j])) #log likelihood + log
50             → width = increment of evidence spanned by iterate
51 push!(e.log_Zi, logaddexp(e.log_Zi[i],e.log_Liwi[j]))      #log
52             → evidence
53 #information- dimensionless quantity
54 push!(e.Hi, lps(
55             (exp(lps(e.log_Liwi[j],-e.log_Zi[j])) * e.log_Li[j]), #term1
56             (exp(lps(e.log_Zi[i],-e.log_Zi[j])) *
57             → lps(e.Hi[i],e.log_Zi[i])), #term2

```

```

44         -e.log_Zi[j])) #term3
45
46     return 0, step_report
47 end
48
49 function nested_step!(e::IPM_Ensemble, model_chan::RemoteChannel,
50   ↳ wk_mon::Worker_Monitor)
51   N = length(e.models)+1 #number of sample models/particles on the
52   ↳ posterior surface- +1 as one has been removed in the distributed
53   ↳ dispatch for converge_ensemble
54   i = length(e.log_Li) #iterate number, index for last values
55   j = i+1 #index for newly pushed values
56
57   e.contour, least_likely_idx = findmin([model.log_Li for model in
58   ↳ e.models])
59   Li_model = e.models[least_likely_idx]
60
61   #SELECT NEW MODEL, SAVE TO ENSEMBLE DIRECTORY, CREATE RECORD AND
62   ↳ REPLACE LEAST LIKELY MODEL
63   model_selected=false;wk=0;step_report=0
64   while !model_selected
65     @async wait(model_chan)
66     candidate,wk,step_report = take!(model_chan)
67     if !(candidate=="quit")
68       if (candidate.log_Li > e.contour) && !(candidate.log_Li in
69         ↳ [m.log_Li for m in e.models])
70         model_selected=true
71
72         new_model_record =
73           Model_Record(string(e.path,'/',e.model_counter),
74             ↳ candidate.log_Li);
75         e.sample_posterior && push!(e.posterior_samples,
76           ↳ Li_model)#if sampling posterior, push the model
77           ↳ record to the ensemble's posterior samples vector
78         deleteat!(e.models, least_likely_idx)
79         push!(e.models, new_model_record);
80
81         final_model=ICA_PWM_Model(string(e.model_counter),
82           ↳ candidate.origin, candidate.sources,
83           ↳ candidate.source_length_limits, candidate.mix_matrix,
84           ↳ candidate.log_Li, candidate.permute_blacklist)
85         serialize(new_model_record.path, final_model)
86
87
88
89
90
91
92
93

```

```

74         e.model_counter +=1
75     end
76     update_worker_monitor!(wk_mon,wk,true)
77   else
78     update_worker_monitor!(wk_mon,wk,false)
79     !any(wk_mon.persist) && (return 1,step_report)
80   end
81 end
82
83 #UPDATE ENSEMBLE QUANTITIES
84 push!(e.log_Li, minimum([model.log_Li for model in e.models])) #log
85   ↳ likelihood of the least likely model - the current ensemble ll
86   ↳ contour at Xi
85 push!(e.log_Xi, -i/N) #log Xi - crude estimate of the iterate's
86   ↳ enclosed prior mass
86 push!(e.log_wi, lps(e.log_Xi[i], -((j+1)/N)-log(2))) #log width of
87   ↳ prior mass spanned by the last step-trapezoidal approx
87 push!(e.log_Liwi, lps(e.log_Li[j],e.log_wi[j])) #log likelihood + log
88   ↳ width = increment of evidence spanned by iterate
88 push!(e.log_Zi, logaddexp(e.log_Zi[i],e.log_Liwi[j]))    #log
89   ↳ evidence
90
90 #information- dimensionless quantity
91 push!(e.Hi, lps(
92   ↳ (exp(lps(e.log_Liwi[j],-e.log_Zi[j])) * e.log_Li[j]), #term1
93   ↳ (exp(lps(e.log_Zi[i],-e.log_Zi[j])) *
94     ↳ lps(e.Hi[i],e.log_Zi[i])), #term2
94   ↳ -e.log_Zi[j])) #term3
95
96   return 0, step_report
97 end

```

---

### 16.7.9 /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
26     tabular_display=DataFrame("Function"⇒funcnames,
27     → "Succeed"⇒zeros(Int64,nfuncs), "Fail"⇒zeros(Int64,
28     → nfuncs), "Velocity"⇒ones(nfuncs), "Weights"⇒instruction.weights)
29
30     instruction.override_time>0. ? (override=true) : (override=false)
31     return
32         → Permute_Tuner(instruction,vels,succs,tabular_display,zeros(CONVERGENCE_MEMORY))
33 end
34
35 """
36     tune_weights!(tuner, call_report)
37
38 Given a call_report from permute_IPM(), adjust tuner's Permute_Instruct
39     → weights for function success rate and likelihood surface velocity.
40 """
41
42 function tune_weights!(tuner::Permute_Tuner,
43     → call_report::Vector{Tuple{Int64,Float64,Float64}})
44     for call in call_report
45         funcidx,time,distance=call
46         distance!==-Inf &&
47             (tuner.velocities[:,funcidx]=update_velocity!(tuner.velocities[:,funcidx])
48             → #do not push velocity to array if it has -Inf probability
49             → (usu no new model found))
50         if call==call_report[end]
51             → tuner.successes[:,funcidx]=update_sucvec!(tuner.successes[:,funcidx],)

```

```

42         tuner.tabular_display[funcidx, "Succeed"]+=1
43     else
44
45         → tuner.successes[:,funcidx]=update_sucvec!(tuner.successes[:,funcidx],
46             tuner.tabular_display[funcidx, "Fail"]+=1
47     end
48 end
49 tuner.override && mean(tuner.time_history)>tuner.inst.override_time ?
50     → (tuner.inst.weights=tuner.inst.override_weights;
51     → tuner.tabular_display[!, "Weights"]=tuner.inst.override_weights) :
52         update_weights!(tuner)
53 end
54
55 function update_velocity!(velvec,time,distance)
56     popfirst!(velvec) #remove first value
57     vel = distance - log(time)
58     push!(velvec,distance-log(time))
59 end
60
61 function update_sucvec!(sucvec, bool)
62     popfirst!(sucvec)
63     push!(sucvec, bool)
64 end
65
66 function update_weights!(t::Permute_Tuner)
67     mvels=[mean(t.velocities[:,n]) for n in 1:length(t.inst.funcs)]
68     t.tabular_display[!, "Velocity"]=copy(mvels)
69
70     any(i→i<0,mvels) && (mvels.=minimum(mvels)+1.) #to calculate
71         → weights, scale negative values into >1.
72     pvec=[mvels[n]*(sum(t.successes[:,n])/length(t.successes[:,n])) for n
73         in 1:length(t.inst.funcs)]
74     pvec./=sum(pvec)
75     (any(pvec.<t.inst.min_clmps) || any(pvec.>t.inst.max_clmps)) &&
76         → clamp_pvec!(pvec,t.inst.min_clmps,t.inst.max_clmps)
77
78     @assert isprobvec(pvec)
79     t.inst.weights=pvec; t.tabular_display[!, "Weights"]=pvec
80 end
81
82 """
83     clamp_pvec(pvec, tuner)
84 """

```

```

79 Clamp the values of a probability vector between the minimums and
  ↳ maximums provided by a Permute_Tuner.
80 """
81     function clamp_pvec!(pvec, min_clmps, max_clmps)
82         #logic- first accumulate on low values, then distribute
83         ↳ excess from high values
84         any(pvec.<min_clmps) ? (low_clamped=false) :
85             ↳ (low_clamped=true)
86         while !low_clamped
87             vals_to_accumulate=pvec.<min_clmps
88             vals_to_deplete=pvec.>min_clmps
89
90             ↳ depletion=sum(min_clmps[vals_to_accumulate]--pvec[vals_to_accumulate])
91             ↳ pvec[vals_to_accumulate].=min_clmps[vals_to_accumulate]
92             ↳ pvec[vals_to_deplete].-=depletion/sum(vals_to_deplete)
93
94             !any(pvec.<min_clmps)&&(low_clamped=true)
95         end
96
97         any(pvec.>max_clmps) ? (high_clamped=false) :
98             ↳ (high_clamped=true)
99         while !high_clamped
100            vals_to_deplete=pvec.>max_clmps
101            vals_to_accumulate=pvec.<max_clmps
102
103            ↳ depletion=sum(pvec[vals_to_deplete]--max_clmps[vals_to_deplete])
104            pvec[vals_to_deplete].=max_clmps[vals_to_deplete]
105
106            ↳ pvec[vals_to_accumulate].+=depletion/sum(vals_to_accumulate)
107
108        !any(pvec.>max_clmps)&&(high_clamped=true)
109    end
110
111 end

```

---

### 16.7.10 /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))
18
19        for host_src in filter(!in(vcat(values(con_idxs) ... )),
20            ↳ keys(con_idxs)) #copy mix information to the source to be
21            ↳ consolidated on as host
22            for cons_src in con_idxs[host_src]
23                new_mix[:,host_src]=[new_mix[o,host_src] ||
24                    ↳ new_mix[o,cons_src] for o in 1:size(new_mix,1)]
25            end
26        end
27
28        remote ? (merger_m = deserialize(rand(models).path)) : (merger_m
29            ↳ = remotecall_fetch(deserialize, 1, rand(models).path))
30            ↳ #randomly select a model to merge
31        used_m_srcs=Vector{Int64}()
32
33        for src in unique(vcat(values(con_idxs) ... )) #replace all
34            ↳ non-host sources with sources from a merger model unlike the
35            ↳ one being removed
36            distvec=[pwm_distance(m.sources[src][1],m_src[1]) for m_src
37                ↳ in merger_m.sources]
38            m_src=findmax(distvec)[2]
39            while m_src in used_m_srcs
40                distvec[m_src]=0.; m_src=findmax(distvec)[2]
41            end
42        end
43    end
44
```

```

26         length(used_m_srcs)==length(merger_m.sources) &&
27             → break; break
28     end
29
30     clean[new_mix[:,src]]*=false #mark dirty any obs that start
31         → with the source
32     new_sources[src]=merger_m.sources[m_src]
33     new_mix[:,src]=merger_m.mix_matrix[:,m_src]
34     clean[new_mix[:,src]]*=false #mark dirty any obs that end
35         → with the source
36     push!(used_m_srcs, m_src)
37 end
38
39 if consolidate_check(new_sources)[1] #if the new sources pass the
40     → consolidate check
41     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
42         → obs_lengths, bg_scores, new_mix, true, true, cache,
43         → clean) #assess likelihood
44 end
45
46 iterate += 1
47 end
48
49 return ICA_PWM_Model("candidate", "consolidated $(m.origin)",
50     → new_sources, m.source_length_limits, new_mix, new_log_Li)
51 end
52
53 function
54     → consolidate_check(sources :: AbstractVector{<: Tuple{<: AbstractMatrix{<: AbstractFloat}}}
55     → thresh=CONSOLIDATE_THRESH, revcomp=REVCOMP)
56     pass=true
57     lengthδmat=[size(src1[1],1) - size(src2[1],1) for src1 in sources,
58         → src2 in sources]
59     cons_idxs=Dict{Integer, Vector{Integer}}()
60     for src1 in 1:length(sources), src2 in src1+1:length(sources)
61         if lengthδmat[src1,src2]==0
62             revcomp ? (info_condition =
63                 → (pwm_distance(sources[src1][1],sources[src2][1]) < thresh
64                 ||
65                 → pwm_distance(sources[src1][1],revcomp_pwm(sources[src2][1]))
66                 < thresh)) : (info_condition =
67                 → (pwm_distance(sources[src1][1],sources[src2][1]) <
68                 thresh))

```

```

53         if info_condition
54             if !in(src1,keys(cons_idxs))
55                 cons_idxs[src1]=[src2]; pass=false
56             else
57                 push!(cons_idxs[src1], src2)
58             end
59         end
60     end
61
62     return pass, cons_idxs
63 end
64
65
66     function pwm_distance(pwm1,pwm2)
67         minwml=min(size(pwm1,1),size(pwm2,1))
68         return sum([euclidean(exp.(pwm1[pos,:]),
69             → exp.(pwm2[pos,:])) for pos in 1:minwml])/minwml
70     end

```

---

### 16.7.11 /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
20     1:length(funcs))=assert_permute_instruct(funcs,weights,args,
21     &&
22     new(funcs,weights,args,model_limit,func_limit,min_clmps,max_c
23
24 end
25
26 function
27   ← assert_permute_instruct(funcs,weights,args,model_limit,func_limit,min_clmps,
28   ← max_clmps, override_time, override_weights)
29
30   ← !(length(funcs)==length(args)==length(weights)==length(min_clmps)==length(max_
31   ← && throw(ArgumentError("A valid Permute_Instruct must have as
32   ← many tuning weights and argument vectors as functions!")))
33   model_limit<1 && throw(ArgumentError("Permute_Instruct limit on
34   ← models to permute must be positive Integer!"))
35   func_limit<1 && throw(ArgumentError("Permute_Instruct limit on
36   ← fuction calls per model permuted must be positive Integer!"))
37   any(min_clmps.≥max_clmps) && throw(ArgumentError("Permute_Instruct
38   ← max_clmps must all be greater than corresponding min_clmps!"))
39   sum(min_clmps)>1. && throw(ArgumentError("Sum of minimum clamps must
40   ← be <1.0 to maintain a valid probability vector!"))
41   any(max_clmps.>1.) && throw(ArgumentError("Permute_Tuner maximum
42   ← clamps cannot be >1.0!"))
43   override_time<0. && throw(ArgumentError("Permute_Instruct
44   ← override_time must be 0 (disabled) or positive float!"))
45   override_time>0. && !isprobvec(override_weights) &&
46   ← throw(ArgumentError("Permute_Instruct override_weights must be
47   ← valid probvec!"))
48
49   return true
50
51 end
52
53
54 function permute_IPM(e::IPM_Ensemble, instruction::Permute_Instruct)
55   call_report=Vector{Tuple{Int64,Float64,Float64}}(){}
56   for model = 1:instruction.model_limit
57     m_record = rand(e.models)
58     m = deserialize(m_record.path)
59
60     model_blacklist=copy(m.permute_blacklist)
61     filteridxs, filtered_funcs, filtered_weights, filtered_args =
62     ← filter_permutes(instruction, model_blacklist)
63
64     for call in 1:instruction.func_limit
65       start=time()

```

```

45         funcidx=rand(filtered_weights)
46         permute_func=filtered_funcs[funcidx]
47
48             ↳ pos_args,kw_args=get_permfunc_args(permute_func,e,m,filtered_args[fun
49             new_m=permute_func(pos_args ... ;kw_args ... )
50
51             ↳ push!(call_report,(filteridxs[funcidx],time()-start,new_m.log_Li
52             ↳ - e.contour))
53
54     if length(new_m.permute_blacklist) > 0 && new_m.log_Li ==
55         ↳ -Inf #in this case the blacklisted function will not work
56         ↳ on this model at all; it should be removed from the
57         ↳ functions to use
58         vcat(model_blacklist,new_m.permute_blacklist)
59         filteridxs, filtered_funcs, filtered_weights,
60         ↳ filtered_args = filter_permutes(instruction,
61         ↳ model_blacklist)
62     end
63
64         dupecheck(new_m,m) && new_m.log_Li > e.contour &&
65             ↳ return new_m, call_report
66         end
67     end
68     return nothing, call_report
69 end
70
71 function permute_IPM(e::IPM_Ensemble, job_chan::RemoteChannel,
72     ↳ models_chan::RemoteChannel, comms_chan::RemoteChannel)
73     ↳ #ensemble.models is partially updated on the worker to populate
74     ↳ arguments for permute funcs
75     persist=true
76     id=myid()
77     put!(comms_chan,id)
78     while persist
79         wait(job_chan)
80         start=time()
81         e.models, e.contour, instruction = fetch(job_chan)
82         instruction = "stop" && (persist=false; break)
83
84         call_report=Vector{Tuple{Int64,Float64,Float64}}()
85         for model=1:instruction.model_limit
86             found::Bool=false
87             m_record = rand(e.models)

```

```

76
77     remotecall_fetch(isfile,1,m_record.path) ? m =
    →   (remotecall_fetch(deserialize,1,m_record.path)) : (break)
78
79     model_blacklist=copy(m.permute_blacklist)
80     filteridxs, filtered_funcs, filtered_weights, filtered_args =
    →   filter_permutes(instruction, model_blacklist)
81
82     for call in 1:instruction.func_limit
83         start=time()
84         funcidx=rand(filtered_weights)
85         permute_func=filtered_funcs[funcidx]
86
    →   pos_args,kw_args=get_permfunc_args(permute_func,e,m,filtered_args)
87         new_m=permute_func(pos_args ... ;kw_args ... )
88
    →   push!(call_report,(filteridxs[funcidx],time()-start,new_m.log_Li
    →   - e.contour))
89
90     if length(new_m.permute_blacklist) > 0 && new_m.log_Li ==
    →   -Inf #in this case the blacklisted function will not
    →   work on this model at all; it should be removed from
    →   the functions to use
91         vcat(model_blacklist,new_m.permute_blacklist)
92         filteridxs, filtered_funcs, filtered_weights,
    →   filtered_args = filter_permutes(instruction,
    →   model_blacklist)
93     end
94
95     dupecheck(new_m,m) && new_m.log_Li > e.contour &&
    →   ((put!(models_chan, (new_m ,id, call_report)));
    →   found=true; break)
96
97     end
98     found=true && break;
99     wait(job_chan)
100    fetch(job_chan)≠e.models && (break) #if the ensemble has
    →   changed during the search, update it
101    model=instruction.model_limit &&
    →   (put!(models_chan, ("quit", id,
    →   call_report));persist=false)#worker to put
    →   "quit" on channel if it fails to find a model
    →   more likely than contour

```



```

136         end
137     end
138
139     kw_args = NamedTuple()
140     if length(args) > 0 #if there are any keyword
141         ← arguments to pass
142         sym=Vector{Symbol}(){}
143         val=Vector{Any}(){}
144         for arg in args
145             push!(sym, arg[1]); push!(val, arg[2])
146         end
147         kw_args = (;zip(sym,val) ... )
148     end
149
150     return pos_args, kw_args
151
152     function dupecheck(new_model, model)
153         (new_model.sources==model.sources &&
154         ← new_model.mix_matrix==model.mix_matrix) ? (return
155         ← false) : (return true)
156     end

```

---

### 16.7.12 /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{<:AbstractFloat}},<:UnitRange{<:Integer}}}},
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     O = size(obs_array,2);S = length(m.sources)
6     new_sources=deepcopy(m.sources);

```

```

7     a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
8         ↵ bg_scores, m.mix_matrix, true, true)
9
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))
15        clean[m.mix_matrix[:,s]]=false #all obs with source are dirty
16
17        weight_shift_freq > 0 &&
18            ↵ (new_sources[s]=permute_source_weights(new_sources[s],
19            ↵ weight_shift_freq, weight_shift_dist))
20        rand() < length_change_freq &&
21            ↵ (new_sources[s]=permute_source_length(new_sources[s],
22            ↵ source_priors[s], m.source_length_limits, length_perm_range))
23
24        new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
25            ↵ obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
26            ↵ clean)
27        iterate += 1
28    end
29
30    cons_check, cons_idxs = consolidate_check(new_sources)
31    cons_check ? (return ICA_PWM_Model("candidate", "PS from $(m.name)", 
32        ↵ new_sources, m.source_length_limits, m.mix_matrix, new_log_Li)) :
33        (return consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate",
34            ↵ "PS from $(m.name)", new_sources, m.source_length_limits,
35            ↵ m.mix_matrix, new_log_Li), obs_array, obs_lengths, bg_scores,
36            ↵ contour, models; remote=remote))
37 end
38
39 function permute_mix(m::ICA_PWM_Model,
40     ↵ obs_array::AbstractMatrix{<:Integer},
41     ↵ obs_lengths::AbstractVector{<:Integer},
42     ↵ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
43     ↵ iterates::Integer=10,
44     ↵ mix_move_range::UnitRange=1:length(m.mix_matrix), remote=false)
45     new_log_Li=-Inf; iterate = 1; O = size(obs_array,2);
46     new_mix=false(size(m.mix_matrix))
47
48     a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
49         ↵ bg_scores, m.mix_matrix, true, true)
50
51     while new_log_Li ≤ contour && iterate ≤ iterates #until we produce
52         ↵ a model more likely than the lh contour or exceed iterates
53         new_sources=deepcopy(m.sources);
54         s = rand(1:S)
55         clean=Vector{Bool}(trues(0))
56         clean[m.mix_matrix[:,s]]=false #all obs with source are dirty
57
58         weight_shift_freq > 0 &&
59             ↵ (new_sources[s]=permute_source_weights(new_sources[s],
60             ↵ weight_shift_freq, weight_shift_dist))
61         rand() < length_change_freq &&
62             ↵ (new_sources[s]=permute_source_length(new_sources[s],
63             ↵ source_priors[s], m.source_length_limits, length_perm_range))
64
65         new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
66             ↵ obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
67             ↵ clean)
68         iterate += 1
69     end
70
71     cons_check, cons_idxs = consolidate_check(new_sources)
72     cons_check ? (return ICA_PWM_Model("candidate", "PS from $(m.name)", 
73         ↵ new_sources, m.source_length_limits, m.mix_matrix, new_log_Li)) :
74         (return consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate",
75             ↵ "PS from $(m.name)", new_sources, m.source_length_limits,
76             ↵ m.mix_matrix, new_log_Li), obs_array, obs_lengths, bg_scores,
77             ↵ contour, models; remote=remote))
78 end
79
80 function IPM_likelihood(sources, obs_array, obs_lengths,
81     ↵ bg_scores, mix_matrix, true, true, cache)
82     if cache
83         return cache
84     end
85
86     N = size(obs_array, 1)
87     S = size(sources, 1)
88
89     log_Li = zeros(N)
90
91     for i in 1:N
92         log_Li[i] = log.(bg_scores[i])
93         for j in 1:S
94             log_Li[i] += log.(mix_matrix[j, i])
95         end
96     end
97
98     log_Li
99 end
100
```

```

31     a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
32         ↵ bg_scores, m.mix_matrix, true, true)
33     dirty=false
34
35     while (new_log_Li <= contour || !dirty) && iterate <= iterates #until
36         ↵ we produce a model more likely than the lh contour or exceed
37         ↵ iterates
38         mix_moves=rand(mix_move_range)
39         mix_moves > length(m.mix_matrix) && (mix_moves =
40             ↵ length(m.mix_matrix))
41
42         new_mix, clean = mix_matrix_decorrelate(m.mix_matrix, mix_moves)
43             ↵ #generate a decorrelated candidate mix
44         c_log_li, c_cache = IPM_likelihood(m.sources, obs_array,
45             ↵ obs_lengths, bg_scores, new_mix, true, true, cache, clean)
46             ↵ #calculate the model with the candidate mix
47         positive_indices=c_cache.>(cache) #obtain any obs indices that
48             ↵ have greater probability than we started with
49         clean=Vector{Bool}(trues(0)-positive_indices)
50         if any(positive_indices) #if there are any such indices
51             new_log_Li, cache = IPM_likelihood(m.sources, obs_array,
52                 ↵ obs_lengths, bg_scores, new_mix, true, true, cache,
53                 ↵ clean) #calculate the new model
54             dirty=true
55         end
56         iterate += 1
57     end
58
59     return ICA_PWM_Model("candidate","PM from $(m.name)", m.sources,
60         ↵ m.source_length_limits, new_mix, new_log_Li, Vector{Function}())
61         ↵ #no consolidate check is necessary as sources havent changed
62 end
63
64 function perm_src_fit_mix(m::ICA_PWM_Model,
65     ↵ models::Vector{Model_Record}, obs_array::AbstractMatrix{<:Integer},
66     ↵ obs_lengths::AbstractVector{<:Integer},
67     ↵ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat,
68     ↵ source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFloat}},  

69     ↵ iterates::Integer=length(m.sources)*2,
70     ↵ weight_shift_freq::AbstractFloat=PWM_SHIFT_FREQ,
71     ↵ length_change_freq::AbstractFloat=PWM_LENGTHPERM_FREQ,
72     ↵ length_perm_range::UnitRange{<:Integer}=LENGTHPERM_RANGE, weight_shift_dist::Distr
73     ↵ remote=false)
74 end

```

```

53     new_log_Li=-Inf; iterate = 1
54     O = size(obs_array,2); S = length(m.sources)
55     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
56     fit_mix in m.permute_blacklist ? (new_bl=m.permute_blacklist) :
57       (new_bl=Vector{Function}())
58
58     while new_log_Li <= contour && iterate <= iterates #until we produce
59       a model more likely than the lh contour or exceed iterates
60       new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix);
61       tm_one=deepcopy(m.mix_matrix);
62       clean=Vector{Bool}(trues(0))
63       s = rand(1:S);
64       clean[new_mix[:,s]]*=false #all obs starting with source are
65         dirty
66
66     new_mix[:,s]*=false;tm_one[:,s]*=true
67
67     weight_shift_freq > 0 &&
68       (new_sources[s]=permute_source_weights(new_sources[s],
69         weight_shift_freq, weight_shift_dist))
70     rand() < length_change_freq &&
71       (new_sources[s]=permute_source_length(new_sources[s],
72         source_priors[s], m.source_length_limits, length_perm_range))
73
73     l,zero_cache=IPM_likelihood(new_sources, obs_array, obs_lengths,
74       bg_scores, new_mix, true, true)
75     l,one_cache=IPM_likelihood(new_sources, obs_array, obs_lengths,
76       bg_scores, tm_one, true, true)
77     fit_mix=one_cache. ≥ zero_cache
78
78     if REVCOMP #if we're looking at reverse strands, we want to see
79       if sources fit better on the reverse strand, and if so switch
80       over to the revcomp source and use its fitted mix
81       revsrc=(revcomp_pwm(new_sources[s][1]),new_sources[s][2])
82       revsrcs=deepcopy(new_sources)
83       revsrcs[s]=revsrc
84       l, revsrc_cache=IPM_likelihood(revsrcs, obs_array,
85         obs_lengths, bg_scores, tm_one, true, true)
86       rfit_mix=revsrc_cache. ≥ zero_cache
87       lps(revsrc_cache[rfit_mix])>lps(one_cache[fit_mix]) &&
88         (new_sources=revsrcs;fit_mix=rfit_mix) #if the total
89         likelihood contribution for the revsource fit is greater
90         than the source fit, take the revsource

```

```

80     end
81
82     new_mix[:,s]=fit_mix
83
84     clean[fit_mix]=false #all obs ending with source are dirty
85
86     new_log_Li = IPM_likelihood(new_sources, obs_array, obs_lengths,
87         ↪ bg_scores, new_mix, true, false, zero_cache, clean)
87     iterate += 1
88
89     end
90
91     cons_check, cons_idxs = consolidate_check(new_sources)
92     cons_check ? (return ICA_PWM_Model("candidate","PSFM from
93         ↪ $(m.name)",new_sources, m.source_length_limits, new_mix,
94         ↪ new_log_Li, new_bl)) : (return consolidate_srcs(cons_idxs,
95         ↪ ICA_PWM_Model("candidate","PSFM from $(m.name)",new_sources,
96         ↪ m.source_length_limits, new_mix, new_log_Li, new_bl), obs_array,
97         ↪ obs_lengths, bg_scores, contour, models; remote=remote))
98   end
99
100
101 function fit_mix(m::ICA_PWM_Model, models::Vector{Model_Record},
102     ↪ obs_array::AbstractMatrix{<:Integer},
103     ↪ obs_lengths::AbstractVector{<:Integer},
104     ↪ bg_scores::AbstractMatrix{<:AbstractFloat}; remote=false)
105     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix);
106     ↪ test_mix=falses(size(m.mix_matrix))
107     new_bl=[fit_mix]
108
109
110     l,zero_cache=IPM_likelihood(m.sources, obs_array, obs_lengths,
111         ↪ bg_scores, test_mix, true, true)
112     for (s,source) in enumerate(m.sources)
113         test_mix=falses(size(m.mix_matrix))
114         test_mix[:,s]=true
115         l,src_cache=IPM_likelihood(m.sources, obs_array, obs_lengths,
116             ↪ bg_scores, test_mix, true, true)
117         fit_mix=src_cache.≥ zero_cache
118
119
120         if REVCOMP #if we're looking at reverse strands, we want to see
121             ↪ if sources fit better on the reverse strand, and if so switch
122             ↪ over to the revcomp source and use its fitted mix
123             revsrc=(revcomp_pwm(source[1]),source[2])
124             revsrcs=deepcopy(new_sources)
125             revsrcs[s]=revsrc

```

```

109     l, revsrc_cache=IPM_likelihood(revsrcts, obs_array,
110         ↵ obs_lengths, bg_scores, test_mix, true, true)
111     rfit_mix=revsrc_cache.≥zero_cache
112     lps(revsrc_cache[rfit_mix])>lps(src_cache[fit_mix]) &&
113         ↵ (new_sources=revsrcts;fit_mix=rfit_mix) #if the total
114         ↵ likelihood contribution for the revsource fit is greater
115         ↵ than the source fit, take the revsource
116     end
117
118     new_mix[:,s]=fit_mix
119 end
120 new_mix=m.mix_matrix ? (new_log_Li=-Inf) : (new_log_Li =
121     ↵ IPM_likelihood(m.sources, obs_array, obs_lengths, bg_scores,
122     ↵ new_mix, true)) #bail if the mix matrix doesnt change
123 new_log_Li in [model.log_Li for model in models] && (new_log_Li=-Inf)
124     ↵ #an existing identical log_Li is almost certainly a fitted model
125     ↵ from another origin with the same sources, in this case abort to
126     ↵ prevent dupes
127
128 return ICA_PWM_Model("candidate","FM from $(m.name)",new_sources,
129     ↵ m.source_length_limits, new_mix, new_log_Li, new_bl) #no
130     ↵ consolidate check necessary as no change to sources
131 end
132
133 function random_decorrelate(m::ICA_PWM_Model,
134     ↵ models::Vector{Model_Record}, obs_array::AbstractMatrix{<:Integer},
135     ↵ obs_lengths::AbstractVector{<:Integer},
136     ↵ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat,
137     ↵ source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFlo
138     ↵ iterates::Integer=length(m.sources)*2,
139     ↵ source_permute_freq::AbstractFloat=SRC_PERM_FREQ,
140     ↵ weight_shift_freq::AbstractFloat=PWM_SHIFT_FREQ,
141     ↵ length_change_freq::AbstractFloat=PWM_LENGTHPERM_FREQ,
142     ↵ weight_shift_dist::Distributions.ContinuousUnivariateDistribution=PWM_SHIFT_DIST,
143     ↵ mix_move_range::UnitRange=1:size(m.mix_matrix,1), remote=false)
144     ↵ new_log_Li=-Inf; iterate = 1
145     ↵ O = size(obs_array,2); S = length(m.sources)
146     ↵ new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
147
148 a, cache = IPM_likelihood(new_sources, obs_array, obs_lengths,
149     ↵ bg_scores, new_mix, true, true)

```

```

129   while new_log_Li ≤ contour && iterate ≤ iterates #until we produce
130     ↵ a model more likely than the lh contour or exceed iterates
131     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
132     clean=Vector{Bool}(trues(0))
133     s = rand(1:S)
134     clean[new_mix[:,s]]*=false #all obs starting with source are
135     ↵ dirty
136     rand() < source_permute_freq &&
137       ↵ (new_sources[s]=permute_source_weights(new_sources[s],
138         ↵ weight_shift_freq, weight_shift_dist))
139     rand() < length_change_freq &&
140       ↵ (new_sources[s]=permute_source_length(new_sources[s],
141         ↵ source_priors[s], m.source_length_limits))
142
143     ↵ new_mix[:,s]=mixvec_decorrelate(new_mix[:,s],rand(mix_move_range))
144     clean[new_mix[:,s]]*=false #all obs ending with source are dirty
145     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
146       ↵ obs_lengths, bg_scores, new_mix, true, true, cache, clean)
147     iterate += 1
148
149 end
150
151
152 function shuffle_sources(m::ICA_PWM_Model,
153   ↵ models :: AbstractVector{<:Model_Record},
154   ↵ obs_array :: AbstractMatrix{<:Integer},
155   ↵ obs_lengths :: AbstractVector{<:Integer},
156   ↵ bg_scores :: AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
157   ↵ remote=false)
158   new_log_Li=-Inf; 0 = size(obs_array,2); S = length(m.sources)
159   new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
160
161   a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
162     ↵ bg_scores, m.mix_matrix, true, true)

```

```

152     remote ? (merger_m = deserialize(rand(models).path)) : (merger_m =
153         ← remotecall_fetch(deserialize, 1, rand(models).path))#randomly
154         ← select a model to merge
155
156     svec=[1:S ... ]
157
158     while new_log_Li ≤ contour && length(svec)>0 #until we produce a
159         ← model more likely than the lh contour or no more sources to
160         ← attempt merger
161         new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
162         clean=Vector{Bool}(trues(0))
163
164         s = popat!(svec,rand(1:length(svec))) #randomly select a source
165             ← to merge
166
167         new_sources[s]=merger_m.sources[s];
168             ← new_mix[:,s] .= merger_m.mix_matrix[:,s] #shuffle in the merger
169             ← model source from this index
170
171         clean[m.mix_matrix[:,s]] .= false #mark dirty any obs that have the
172             ← source
173         new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
174             ← obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
175             ← clean) #assess likelihood
176     end
177
178     cons_check, cons_idxs = consolidate_check(new_sources)
179     cons_check ? (return ICA_PWM_Model("candidate","SS from
180         ← $(m.name)",new_sources, m.source_length_limits, new_mix,
181         ← new_log_Li,Vector{Function}())) : (return
182         ← consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate","SS from
183         ← $(m.name)",new_sources, m.source_length_limits, new_mix,
184         ← new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;
185         ← remote=remote))
186 end
187
188 function accumulate_mix(m::ICA_PWM_Model,
189     ← models::AbstractVector{<:Model_Record},
190     ← obs_array::AbstractMatrix{<:Integer},
191     ← obs_lengths::AbstractVector{<:Integer},
192     ← bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
193     ← remote=false)
194     new_log_Li=-Inf; 0 = size(obs_array,2); S = length(m.sources)

```

```

174     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
175     new_b1=Vector{Function}()
176
177     a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
178     ↪ bg_scores, m.mix_matrix, true, true)
179
180     remote ? (merger_m = deserialize(rand(models).path)) : (merger_m =
181     ↪ remotecall_fetch(deserialize, 1, rand(models).path))#randomly
182     ↪ select a model to merge
183
184     svec=[1:S ... ]
185
186     while new_log_Li ≤ contour && length(svec)>0 #until we produce a
187     ↪ model more likely than the lh contour or no more sources to
188     ↪ attempt merger
189     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
190     clean=Vector{Bool}(trues(0))
191
192     s = popat!(svec,rand(1:length(svec))) #randomly select a source
193     ↪ to merge
194     distvec=[pwm_distance(src[1],m_src[1]) for src in m.sources,
195     ↪ m_src in merger_m.sources]
196     S > 1 ? merge_s=findmin(distvec)[2][2] :
197     ↪ merge_s=findmin(distvec)[2]
198
199     new_mix[:,s]=[new_mix[o,s] || merger_m.mix_matrix[o,merge_s] for
200     ↪ o in 1:size(new_mix,1)] #accumulate the mix vector for this
201     ↪ source
202
203     clean[m.mix_matrix[:,s]]*=false #mark dirty any obs that have the
204     ↪ source
205     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
206     ↪ obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
207     ↪ clean) #assess likelihood
208
209     if new_log_Li ≤ contour #if accumulating on the host source
210     ↪ doesnt work, try copying over the merger source
211     new_sources[s]=merger_m.sources[merge_s]
212     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
213     ↪ obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
214     ↪ clean) #assess likelihood
215
216   end
217
218 end

```

```

201 (new_log_Li = m.log_Li || new_log_Li in [model.log_Li for model in
202   ↵ models]) && (new_log_Li=-Inf) #accumulate can sometimes duplicate
203   ↵ models if source mix is identical
204
205 cons_check, cons_idxs = consolidate_check(new_sources)
206 cons_check ? (return ICA_PWM_Model("candidate","AM from
207   ↵ $(m.name)",new_sources, m.source_length_limits, new_mix,
208   ↵ new_log_Li)) : (return consolidate_srcs(cons_idxs,
209   ↵ ICA_PWM_Model("candidate","AM from $(m.name)",new_sources,
210   ↵ m.source_length_limits, new_mix, new_log_Li), obs_array,
211   ↵ obs_lengths, bg_scores, contour, models; remote=remote))
212 end
213
214 function distance_merge(m::ICA_PWM_Model,
215   ↵ models::AbstractVector{<:Model_Record},
216   ↵ obs_array::AbstractMatrix{<:Integer},
217   ↵ obs_lengths::AbstractVector{<:Integer},
218   ↵ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
219   ↵ remote=false)
220   new_log_Li=-Inf; O = size(obs_array,2); S = length(m.sources)
221   new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
222
223   a, cache = IPM_likelihood(new_sources, obs_array, obs_lengths,
224     ↵ bg_scores, new_mix, true, true)
225
226   remote ? (merger_m = deserialize(rand(models).path)) : (merger_m =
227     ↵ remotecall_fetch(deserialize, 1, rand(models).path)) #randomly
228     ↵ select a model to merge
229
230   svec=[1:S ... ]
231
232   while new_log_Li ≤ contour && length(svec)>0 #until we produce a
233     ↵ model more likely than the lh contour or no more sources to
234     ↵ attempt merger
235     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
236     clean=Vector{Bool}(trues(O))
237
238     s = popat!(svec,rand(1:length(svec))) #randomly select a source
239       ↵ to merge
240     merge_s=most_dissimilar(new_mix[:,s],merger_m.mix_matrix) #find
241       ↵ the source in the merger model whose mixvec is most
242       ↵ dissimilar to the one selected

```

```

224
225     clean[new_mix[:,s]] = false #mark dirty any obs that start with
226     ↪   the source
227     new_sources[s] = merger_m.sources[merge_s] #copy the source
228     new_mix[:,s] = merger_m.mix_matrix[:,merge_s] #copy the mixvector
229     ↪   (without which the source will likely be highly improbable)
230     clean[new_mix[:,s]] = false #mark dirty any obs that end with the
231     ↪   source
232
233     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
234     ↪   obs_lengths, bg_scores, new_mix, true, true, cache, clean)
235     ↪   #assess likelihood
236   end
237
238   cons_check, cons_idxs = consolidate_check(new_sources)
239   cons_check ? (return ICA_PWM_Model("candidate", "DM from
240   ↪   $(m.name)", new_sources, m.source_length_limits, new_mix,
241   ↪   new_log_Li, Vector{Function}())) : (return
242   ↪   consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate", "DM from
243   ↪   $(m.name)", new_sources, m.source_length_limits, new_mix,
244   ↪   new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;
245   ↪   remote=remote))
246 end
247
248 function similarity_merge(m::ICA_PWM_Model,
249   ↪   models::AbstractVector{<:Model_Record},
250   ↪   obs_array::AbstractMatrix{<:Integer},
251   ↪   obs_lengths::AbstractVector{<:Integer},
252   ↪   bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
253   ↪   remote=false)
254   new_log_Li=-Inf; O = size(obs_array,2); S = length(m.sources)
255   new_sources=deepcopy(m.sources)
256
257   a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
258   ↪   bg_scores, m.mix_matrix, true, true)
259
260   remote ? (merger_m = deserialize(rand(models).path)) : (merger_m =
261   ↪   remotecall_fetch(deserialize, 1, rand(models).path))#randomly
262   ↪   select a model to merge
263
264   svec=[1:S ... ]
265
266

```

```

247   while new_log_Li ≤ contour && length(svec)>0 #until we produce a
248     ← model more likely than the lh contour or no more sources to
249     ← attempt merger
250
251     new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
252     clean=Vector{Bool}(trues(0))
253
254     s = popat!(svec,rand(1:length(svec))) #randomly select a source
255     ← to merge
256     merge_s=most_similar(m.mix_matrix[:,s],merger_m.mix_matrix)
257     ← #obtain the source in the merger model whose mixvec is most
258     ← similar to the one in the original
259
260     clean[m.mix_matrix[:,s]]*=false #mark dirty any obs that have the
261     ← source
262     new_sources[s] = merger_m.sources[merge_s] #copy the source, but
263     ← dont copy the mixvector
264     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
265     ← obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
266     ← clean) #assess likelihood
267   end
268
269   cons_check, cons_idxs = consolidate_check(new_sources)
270   cons_check ? (return ICA_PWM_Model("candidate","SM from
271     ← $(m.name)",new_sources, m.source_length_limits, m.mix_matrix,
272     ← new_log_Li,Vector{Function}())) : (return
273     ← consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate","SM from
274     ← $(m.name)",new_sources, m.source_length_limits, m.mix_matrix,
275     ← new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;
276     ← remote=remote))
277 end
278
279 function reinit_src(m::ICA_PWM_Model,
280   ← models::AbstractVector{<:Model_Record},
281   ← obs_array::AbstractMatrix{<:Integer},
282   ← obs_lengths::AbstractVector{<:Integer},
283   ← bg_scores::AbstractMatrix{<:AbstractFloat},contour::AbstractFloat,
284   ← source_priors::AbstractVector{<:Union{<:AbstractVector{<:Dirichlet{<:AbstractFlo
285   ← iterates::Integer=length(m.sources)*2, remote=false}
286   new_log_Li=-Inf; iterate = 1
287   O = size(obs_array,2); S = length(m.sources)
288   new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix)
289
290

```

```

268   while new_log_Li ≤ contour && iterate ≤ iterates #until we produce
269     ↵ a model more likely than the lh contour or exceed iterates
270   new_sources=deepcopy(m.sources); new_mix=deepcopy(m.mix_matrix);
271   ↵ tm_one=deepcopy(m.mix_matrix);
272   clean=Vector{Bool}(trues(0))
273   s = rand(1:S);
274   clean[new_mix[:,s]]*=false #all obs starting with source are
275   ↵ dirty
276
277   new_mix[:,s]*=false;tm_one[:,s]*=true
278
279   new_sources[s] = init_logPWM_sources([source_priors[s]],
280   ↵ m.source_length_limits)[1] #reinitialize the source
281
282   l,zero_cache=IPM_likelihood(new_sources, obs_array, obs_lengths,
283   ↵ bg_scores, new_mix, true, true)
284   l,one_cache=IPM_likelihood(new_sources, obs_array, obs_lengths,
285   ↵ bg_scores, tm_one, true, true)
286
287   fit_mix=one_cache. ≥ zero_cache
288
289   new_mix[:,s]=fit_mix
290
291   clean[fit_mix]*=false #all obs ending with source are dirty
292
293   new_log_Li = IPM_likelihood(new_sources, obs_array, obs_lengths,
294   ↵ bg_scores, new_mix, true, false, zero_cache, clean)
295   iterate += 1
296 end
297
298 cons_check, cons_idxs = consolidate_check(new_sources)
299 cons_check ? (return ICA_PWM_Model("candidate","RS from $(m.name)",
300   ↵ new_sources, m.source_length_limits, new_mix,
301   ↵ new_log_Li,Vector{Function}())) : (return
302   ↵ consolidate_srcs(cons_idxs, ICA_PWM_Model("candidate","RS from
303   ↵ $(m.name)",new_sources, m.source_length_limits, new_mix,
304   ↵ new_log_Li), obs_array, obs_lengths, bg_scores, contour, models;
305   ↵ remote=remote))
306 end
307
308

```

```

295 function erode_model(m::ICA_PWM_Model,
296   → models :: AbstractVector{<:Model_Record},
297   → obs_array :: AbstractMatrix{<:Integer},
298   → obs_lengths :: AbstractVector{<:Integer},
299   → bg_scores :: AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
300   → info_thresh::AbstractFloat=EROSION_INFO_THRESH, remote=false)
301   new_log_Li=-Inf; 0 = size(obs_array,2);
302   new_sources=deepcopy(m.sources)

303
304   erosion_sources=Vector{Integer}()
305   for (s,src) in enumerate(m.sources)
306     pwm,pi=src
307     if size(pwm,1)>m.source_length_limits[1] #do not consider
308       → eroding srcs at min length limit
309       infovec=get_pwm_info(pwm)
310       any(info-><(info, info_thresh),infovec) &&
311       → push!(erosion_sources,s)
312   end
313 end

314
315
316
317
318
319
320

```

length(erosion\_sources)==0 && return ICA\_PWM\_Model("candidate","EM  
   → from \$(m.name)",new\_sources, m.source\_length\_limits,  
   → m.mix\_matrix, new\_log\_Li,[erode\_model])#if we got a model we cant  
   → erode bail out with a model marked -Inf lh and with EM  
   → blacklisted

a, cache = IPM\_likelihood(m.sources, obs\_array, obs\_lengths,  
   → bg\_scores, m.mix\_matrix, true, true)

while new\_log\_Li ≤ contour && length(erosion\_sources) > 0 #until we  
   → produce a model more likely than the lh contour or there are no  
   → more sources to erode  
   clean=Vector{Bool}(trues(0))  
   s=popat!(erosion\_sources,rand(1:length(erosion\_sources)))

new\_sources[s]=erode\_source(new\_sources[s],  
   → m.source\_length\_limits, info\_thresh)  
   clean[m.mix\_matrix[:,s]]\*=false

new\_log\_Li, cache = IPM\_likelihood(new\_sources, obs\_array,  
   → obs\_lengths, bg\_scores, m.mix\_matrix, true, true, cache,  
   → clean) #assess likelihood

end

```

321 new_log_Li <= contour ? (blacklist=[erode_model]) :
322   ↵ (blacklist=Vector{Function}())
323
324 cons_check, cons_idxs = consolidate_check(new_sources)
325 cons_check ? (return ICA_PWM_Model("candidate", "EM from
326   ↵ $(m.name)", new_sources, m.source_length_limits, m.mix_matrix,
327   ↵ new_log_Li, blacklist)) : (return consolidate_srcs(cons_idxs,
328   ↵ ICA_PWM_Model("candidate", "EM from $(m.name)", new_sources,
329   ↵ m.source_length_limits, m.mix_matrix, new_log_Li, blacklist),
330   ↵ obs_array, obs_lengths, bg_scores, contour, models;
331   ↵ remote=remote))
332 end
333
334 function info_fill(m::ICA_PWM_Model,
335   ↵ models::AbstractVector{<:Model_Record},
336   ↵ obs_array::AbstractMatrix{<:Integer},
337   ↵ obs_lengths::AbstractVector{<:Integer},
338   ↵ bg_scores::AbstractMatrix{<:AbstractFloat}, contour::AbstractFloat;
339   ↵ remote=false)
340   new_log_Li=-Inf; O = size(obs_array,2); S = length(m.sources);
341   ↵ iterate=1
342   new_sources=deepcopy(m.sources)
343
344   a, cache = IPM_likelihood(m.sources, obs_array, obs_lengths,
345     ↵ bg_scores, m.mix_matrix, true, true)
346
347   svec=[1:S ... ]
348
349   while new_log_Li <= contour && length(svec)>0 #until we produce a
350     ↵ model more likely than the lh contour or no more sources to
351     ↵ attempt infofill
352       new_sources=deepcopy(m.sources)
353       clean=Vector{Bool}(trues(O))
354       s = popat!(svec,rand(1:length(svec)))
355       pwm=new_sources[s][1]
356       fill_idx=findmin(get_pwm_info(pwm))[2]
357       fill_bases=[1,2,3,4]
358       fill_candidate=findmax(pwm[fill_idx,:])[2]
359       deleteat!(fill_bases, findfirst(b→b==fill_candidate,fill_bases))
360       new_sources[s][1][fill_idx,:]==-Inf;
361       ↵ new_sources[s][1][fill_idx,fill_candidate]=0.
362       clean[m.mix_matrix[:,s]]-=false

```

```

347     new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
348         ↵ obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
349         ↵ clean) #assess likelihood

350     while new_log_Li <= contour && length(fill_bases)>0
351         fill_candidate=popat!(fill_bases,rand(1:length(fill_bases)))
352         new_sources[s][1][fill_idx,:]==-Inf;
353             ↵ new_sources[s][1][fill_idx,fill_candidate]=0.
354         new_log_Li, cache = IPM_likelihood(new_sources, obs_array,
355             ↵ obs_lengths, bg_scores, m.mix_matrix, true, true, cache,
356             ↵ clean) #assess likelihood
357     end
358 end
359
360 new_log_Li <= contour ? (blacklist=[info_fill]) :
361     ↵ (blacklist=Vector{Function}())
362
363 cons_check, cons_idxs = consolidate_check(new_sources)
364 cons_check ? (return ICA_PWM_Model("candidate","IF from
365     ↵ $(m.name)",new_sources, m.source_length_limits, m.mix_matrix,
366     ↵ new_log_Li, blacklist)) : (return consolidate_srcs(cons_idxs,
367     ↵ ICA_PWM_Model("candidate","IF from $(m.name)",new_sources,
368     ↵ m.source_length_limits, m.mix_matrix, new_log_Li, blacklist),
369     ↵ obs_array, obs_lengths, bg_scores, contour, models;
370     ↵ remote=remote))
371
372 full_perm_funcvec=[permute_source, permute_mix, perm_src_fit_mix,
373     ↵ fit_mix, random_decorrelate, shuffle_sources, accumulate_mix,
374     ↵ distance_merge, similarity_merge, reinit_src, erode_model, info_fill]

```

### 16.7.13 /src/permuation/permute\_utilities.jl

```
1 ##BASIC UTILITY FUNCTIONS
2 #SOURCE PERMUTATION
3 function
4   ↵  permute_source_weights(source::Tuple{<:AbstractMatrix{<:AbstractFloat},<:Integer})
5   ↵  shift_freq::AbstractFloat,
6   ↵  PWM_shift_dist::Distribution{Univariate,Continuous})
7   ↵  dirty=false; source_length=size(source[1],1)
8   ↵  new_source=deepcopy(source)
9
```

```

7   for source_pos in 1:source_length
8     if rand() ≤ shift_freq
9       pos_WM = exp.(source[1][source_pos,:]) #leaving logspace, get
10      ↳ the wm at that position
11      new_source[1][source_pos,:] = log.(wm_shift(pos_WM,
12        ↳ PWM_shift_dist)) #accumulate probility at a randomly
13        ↳ selected base, reassign in logspace and carry on
14        !dirty && (dirty=true)
15      end
16    end
17
18    if !dirty #if no positions were shifted, pick one and shift
19      rand_pos=rand(1:source_length)
20      pos_WM = exp.(source[1][rand_pos,:])
21      new_source[1][rand_pos,:]=log.(wm_shift(pos_WM, PWM_shift_dist))
22    end
23
24    return new_source
25  end
26
27  function
28    ↳ wm_shift(pos_WM::AbstractVector{<:AbstractFloat},
29    ↳ PWM_shift_dist::Distribution{Univariate,Continuous})
30    base_to_shift = rand(1:4) #pick a base to accumulate
31    ↳ probability
32    permute_sign = rand(-1:2:1)
33    shift_size = rand(PWM_shift_dist)
34    new_wm=zeros(4)
35
36    for base in 1:4 #ACGT
37      if base == base_to_shift
38        new_wm[base] =
39        clamp(0, #no lower than 0 prob
40          (pos_WM[base] #selected PWM posn
41           + permute_sign * shift_size), #randomly
42           ↳ permuted by size param
43           1) #no higher than prob 1
44      else
45        size_frac = shift_size / 3 #other bases
46        ↳ shifted in the opposite direction by 1/3
47        ↳ the shift accumulated at the base to
48        ↳ permute
49        new_wm[base] =
50      end
51    end
52  end
53
```

```

40             clamp(0,
41             (pos_WM[base]
42             - permute_sign * size_frac),
43             1)
44         end
45     end
46     new_wm = new_wm ./ sum(new_wm) #renormalise to sum 1
47     ↪ - necessary in case of clamping at 0 or 1
48     !isprobvec(new_wm) && throw(DomainError(new_wm, "Bad
49     ↪ weight vector generated in wm_shift!")) #throw
50     ↪ assertion exception if the position WM is invalid
51
52     return new_wm
53 end
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72

```

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)

permute\_sign, permute\_length = get\_length\_params(source\_length,  
length\_limits, permute\_range)

permute\_sign==1 ? permute\_pos = rand(1:source\_length+1) :  
permute\_pos=rand(1:source\_length-permute\_length)

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)))  
↪ :

```

73         ins_WM[pos,:] =
    ↳ log.(transpose(rand(prior[prior_position])))
74     end
75 end
76 upstream_source=source_PWM[1:permute_pos-1,:]
77 downstream_source=source_PWM[permute_pos:end,:]
78 source_PWM=vcat(upstream_source,ins_WM,downstream_source)
79 permute_pos==1 && (prior_idx-=permute_length)
80 else #if we're to remove positions
81     upstream_source=source_PWM[1:permute_pos-1,:]
82     downstream_source=source_PWM[permute_pos+permute_length:end,:]
83     source_PWM=vcat(upstream_source,downstream_source)
84     permute_pos==1 && (prior_idx+=permute_length)
85 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
96     if extendable == 0 && contractable > 0
97         permute_sign=-1
98     elseif contractable == 0 && extendable > 0
99         permute_sign=1
100    else
101        permute_sign = rand(-1:2:1)
102    end
103
104    permute_sign==1 && extendable<permute_range[end] &&
    ↳ (permute_range=permute_range[1]:extendable)
105    permute_sign== -1 && contractable<permute_range[end]
    ↳ && (permute_range=permute_range[1]:contractable)
106    permute_length = rand(permute_range)
107
108    return permute_sign, permute_length
109 end
110
111 function
    ↳ erode_source(source::Tuple{<:AbstractMatrix{<:AbstractFloat},<:Integer},length_li

```

```

110     pwm,prior_idx=source
111     infovec=get_pwm_info(pwm)
112     start_idx,end_idx=get_erosion_idxs(infovec, info_thresh,
113                                         ↪ length_limits)
114
115     return new_source=(pwm[start_idx:end_idx,:], prior_idx+start_idx-1)
116 end
117
118 function get_pwm_info(pwm::AbstractMatrix{<:AbstractFloat};
119                         ↪ logsw::Bool=true)
120     wml=size(pwm,1)
121     infovec=zeros(wml)
122     for pos in 1:wml
123         logsw ? wvec=deepcopy(exp.(pwm[pos,:])) :
124             ↪ wvec=deepcopy(pwm[pos,:])
125         !isprobvec(wvec) && throw(DomainError(wvec, "Bad wvec in
126                                         ↪ get_pwm_info -Original sources must be in logspace!!"))
127         wvec.=+10^-99
128         infscore = (2.0 + sum([x*log(2,x) for x in wvec]))
129         infovec[pos]=infscore
130     end
131     return infovec
132 end
133
134 function get_erosion_idxs(infovec::AbstractVector{<:AbstractFloat},
135                           ↪ info_thresh::AbstractFloat, length_limits::UnitRange{<:Integer})
136     srcl=length(infovec)
137     contractable = srcl-length_limits[1]
138     contractable ≤ 0 && throw(DomainError(contractable, "erode_source
139                                         ↪ passed a source at its lower length limit!"))
140     centeridx=findmax(infovec)[2]
141
142     start_idx=findprev(info→<(info,info_thresh),infovec,centeridx)
143     start_idx==nothing ? (start_idx=1) : (start_idx+=1)
144     end_idx=findnext(info→<(info, info_thresh),infovec,centeridx)
145     end_idx==nothing ? (end_idx=srcl) : (end_idx-=1)
146
147     pos_to_eroде=srcl-(end_idx-start_idx)
148     if pos_to_eroде > contractable
149         pos_to_restore = pos_to_eroде-contractable
150         while pos_to_restore>0
151             end_die=rand()
152             if end_die ≤ .5

```

```

147             start_idx>1 && (pos_to_restore-=1; start_idx-=1)
148         else
149             end_idx<srcl && (pos_to_restore-=1; end_idx+=1)
150         end
151     end
152 end
153
154     return start_idx, end_idx
155 end
156
157 #MIX MATRIX FUNCTIONS
158 function mixvec_decorrelate(mix::BitVector, moves::Integer)
159     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
191     merge_s=findmin(src_sim)[2] #source from merger model will be the one
192                 with the highest equality comparison score
193
194 end
195
196
197 function most_similar(src_mixvec, target_mixmat)
198     src_sim = [sum(src_mixvec.=target_mixmat[:,s]) for s in
199                 1:size(target_mixmat,2)] #compose array of elementwise equality
200                 comparisons between mixvectors and sum to score
201     merge_s=findmax(src_sim)[2] #source from merger model will be the one
202                 with the highest equality comparison score
203
204 end

```

---

### 16.7.14 /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
16     ↪ 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
34   ↵  diagrams
34 function
35   ↵  logo_from_model(model::ICA_PWM_Model,svg_output::String;freq_sort::Bool=false)
35   source_tups = Vector{Tuple{AbstractFloat, Integer,
36     ↵  Matrix{AbstractFloat}}}{}) #(%of sequences w/ source, prior index,
36     ↵  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],
39         ↵  prior, exp.(source[1])))
40   end
41 freq_sort && sort(source_tups)
42
43 file = open(svg_output, "w")
44 write(file,
45   ↵  svg_header(xpad+xpixels_per_position*maximum([length(source[1])
45     ↵  for source in
45     ↵  model.sources]),ypixels_per_source*length(model.sources)+ypad))
46
46 curry = 0
47 for (frequency, index, source) in source_tups
48   curry += yscale
49   font1y = curry-190
50   ndig = ndigits(index+1)
51   write(file,
52     ↵  pwm_to_logo(source,source_left_x,curry,xlen,yscale*scale_factor))
52   write(file, "<text x=\"10\" y=\"$font1y\""
53     ↵  font-family=\"Helvetica\" font-size=\"$fontsize1\""
53     ↵  font-weight=\"bold\" >$index</text>")

```

```

53     write(file, "<text x=\"$((20+fontsize1*0.6*ndig))\""
54     ↪   y=\"$((font1y-fontsize1+1.5*fontsize2))\""
55     ↪   font-family=\"Helvetica\" font-size=\"$fontsize2\""
56     ↪   font-weight=\"bold\" >$((frequency*100)) % of
57     ↪   sequences</text>\n")
58 end
59
60
61 function svg_header(canvas_size_x, canvas_size_y)
62     return """
63     <?xml version="1.0" standalone="no"?>
64     <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
65     "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
66     <svg width=\"$canvas_size_x\" height=\"$canvas_size_y"
67     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

```

```

89     wvec.+=10^-99
90     infoscore=(2.0 + sum([x*log(2,x) for x in wvec]))
91     infovec = [x*infoscore for x in wvec]
92     val,idx=findmax(infovec)
93     return char,color=get_char_by_thresh(idx,val)
94 end
95     function get_char_by_thresh(idx,val)
96         char = '?'; seen_thresh=0.
97         for (thresh,threshchar) in thresh_dict
98             val ≥ thresh && thresh ≥ seen_thresh &&
99                 (char=threshchar; seen_thresh=thresh)
100            end
101            char=='?' && println(val)
102            char=="lwcs" && (char=lwcs_vec[idx])
103            char=="upcs" && (char=upcs_vec[idx])
104            color=cs_vec[idx]
105
106        return char,color
107    end
108
109
110
111
112 function print_weightvec_at_x_y(wvec, xpos, ypos, xlen, yscale)
113     # xlen is the length occupied by that column
114     # yscale is the total height for 2 information bits i.e. the
115     # maximum height available for a "perfect" nucleotide
116     outstr = ""
117     basestr = "ACGT"
118     wvec.+=10^-99 #prevent log(0) = -Inf
119     wvec = [x/sum(wvec) for x in wvec] #renorm
120     infscore = (2.0 + sum([x*log(2,x) for x in wvec]))
121     if infscore==0.0
122         return ""
123     end
124     wvec = [x*infscore*yscale/2.0 for x in wvec]
125     # at this point, the sum of all wvec is a maximum of yscale
126     wveclist = [(wvec[n],basestr[n]) for n in 1:4]
127     wveclist = sort(wveclist)
128     curr_ypos = ypos
129     for n in 1:4
130         curr_ypos -= wveclist[n][1]

```

```

131         outstr *=
132             ↪ print_char_in_rect(wveclist[n][2],xpos,curr_ypos,xlen,wveclist[n][1])
133     end
134   return outstr
135 end
136 blo = 1
137
138 function print_char_in_rect(c,x,y,width,height)
139     raw_x, raw_y, raw_fontsize, raw_xsclae = charvals_dict[c]
140     raw_x = (raw_x*raw_xsclae-100)/raw_xsclae
141     raw_y = raw_y-100
142
143     xscale = width/100.0 * raw_xsclae
144     yscale = height/100.0
145
146     scaled_x = x/xscale + raw_x
147     scaled_y = y/yscale + raw_y
148
149     return "<text x=\"$scaled_x\" y=\"$scaled_y\""
150         ↪ font-size=\"$raw_fontsize\" font-weight=\"bold\""
151         ↪ font-family=\"Helvetica\" fill=\"$rgb*colour_dict[c]*\""
152         ↪ transform=\"scale($xscale,$yscale)\">$c</text>\n"
153 end
154
155 function svg_footer()
156     return "</svg>"
157 end

```

---

### 16.7.15 /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

```

```

1    printed::Bool          # true if we have issued at least one status
2      ↵ update
3
4    desc::AbstractString # prefix to the percentage, e.g. "Computing ... "
5    color::Symbol         # default to green
6    output::IO             # output stream into which the progress is
7      ↵ written
8    numprintedvalues::Integer # num values printed below progress in
9      ↵ last iteration
10   offset::Integer        # position offset of progress bar
11     ↵ (default is 0)
12
13
14
15
16
17
18   e::IPM_Ensemble
19   wm::Worker_Monitor
20   tuner::Permute_Tuner
21   top_m::ICA_PWM_Model
22
23   mean_stp_time::AbstractFloat
24
25   wk_disp::Bool
26   tuning_disp::Bool
27   conv_plot::Bool
28   ens_disp::Bool
29   lh_disp::Bool
30   liwi_disp::Bool
31   src_disp::Bool
32   no_displayed_srcs::Integer
33
34   disp_rotate_inst::Vector{Any}
35
36   convergence_history::Vector{Float64}
37
38   function ProgressNS{T}(
39     e::IPM_Ensemble,
40     top_m::ICA_PWM_Model,
41     wm::Worker_Monitor,
42     tuner::Permute_Tuner,
43     interval::T;
44     dt::Real=0.1,
45     desc::AbstractString="Nested Sampling :: ",
46     color::Symbol=:green,
47     output::IO=stdout,
48     offset::Int=0,
49     start_it::Int=1,
50     wk_disp::Bool=false,
51     )
52     ...
53   end

```

```

50          tuning_disp::Bool=false,
51          conv_plot::Bool=true,
52          ens_disp::Bool=false,
53          lh_disp::Bool=false,
54          liwi_disp::Bool=false,
55          src_disp::Bool=true,
56          nsrcts::Integer=0,
57
58          ↳ disp_rotate_inst=[false,0,0,Vector{Vector{Symbol}}]
59          ↳ where T
60
61      tfirst = tlast = time()
62      printed = false
63      new{T}(interval,
64          dt,
65          start_it,
66          start_it,
67          false,
68          tfirst,
69          tlast,
70          0.,
71          printed,
72          desc,
73          color,
74          output,
75          0,
76          offset,
77          e,
78          wM,
79          tuner,
80          top_m,
81          0.,
82          wk_disp,
83          tuning_disp,
84          conv_plot,
85          ens_disp,
86          lh_disp,
87          liwi_disp,
88          src_disp,
89          nsrcts,
90          disp_rotate_inst,
91          zeros(CONVERGENCE_MEMORY))
92
93      end
94  end

```



```

117     t = time()
118     p.disp_rotate_inst[1] && p.counter%p.disp_rotate_inst[2] == 0 &&
119         → rotate_displays(p)
120
121     if p.interval ≤ 0 && !p.triggered
122         p.triggered = true
123         if p.printed
124             p.triggered = true
125             dur = durationstring(t-p.tfirst)
126             msg = @sprintf "%s Converged. Time: %s (%d iterations). logZ:
127             → %s\n" p.desc dur p.counter p.e.log_Zi[end]
128
129             print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
130             move_cursor_up_while_clearing_lines(p.output,
131                 → p.numprintedvalues)
132             upper_lines=display_upper_dash(p)
133             printover(p.output, msg, :magenta)
134             lower_lines=display_lower_dash(p)
135
136             p.numprintedvalues=upper_lines + lower_lines + 1
137
138             if keep
139                 println(p.output)
140             else
141                 print(p.output, "\r\u1b[A" ^ (p.offset +
142                 → p.numprintedvalues))
143             end
144         end
145         return
146     end
147
148     if t > p.tlast+p.dt && !p.triggered
149         p.counter < CONVERGENCE_MEMORY ?
150             mean_step_time=mean(p.tuner.time_history[end-(p.counter-1):end])
151             :
152             → mean_step_time=mean(p.tuner.time_hist
153
154             msg = @sprintf "%s Iterate: %s Recent step time μ: %s Convergence
155             → Interval: %g\n" p.desc p.counter hmss(mean_step_time)
156             → p.interval
157
158             print(p.output, "\n" ^ (p.offset + p.numprintedvalues))
159             move_cursor_up_while_clearing_lines(p.output, p.numprintedvalues)

```

```

151     upper_lines=display_upper_dash(p)
152     printover(p.output, msg, p.color)
153     lower_lines=display_lower_dash(p)
154
155     p.numprintedvalues=upper_lines + lower_lines + 1
156     print(p.output, "\r\u1b[A" ^ (p.offset + p.numprintedvalues))
157
158     # Compensate for any overhead of printing. This can be
159     # especially important if you're running over a slow network
160     # connection.
161     p.tlast = t + 2*(time()-t)
162     p.printed = true
163   end
164 end
165
166 function rotate_displays(p)
167   curr_inst=p.disp_rotate_inst[3]
168   curr_inst+1>length(p.disp_rotate_inst[4]) ?
169     ↪ next_inst=1 : next_inst=curr_inst+1
170   next_disps=p.disp_rotate_inst[4][next_inst]
171   for disp in
172     ↪ [:wk_disp,:tuning_disp,:conv_plot,:ens_disp,:lh_disp,:liwi_di
173       disp in next_disps ? setproperty!(p, disp, true)
174         ↪ : setproperty!(p, disp, false)
175   end
176   p.disp_rotate_inst[3]=next_inst
177 end
178
179 function hmss(dt)
180   dt<0 ? (dt=-dt; prfx="-") : (prfx="")
181   isnan(dt) && return "NaN"
182   (h,r) = divrem(dt,60*60)
183   (m,r) = divrem(r, 60)
184   (isnan(h)||isnan(m)||isnan(r)) && return "NaN"
185   string(prfx,Int(h),":",Int(m),":",Int(ceil(r)))
186 end
187
188 function display_upper_dash(p::ProgressNS)
189   wklines = tunelines = cilines = 0
190   p.wk_disp && (wklines=show(p.output, p.wm,
191     ↪ progress=true);println())
192   p.tuning_disp && (tunelines=show(p.output, p.tuner,
193     ↪ progress=true);println())
194   if p.conv_plot

```

```

189          ↵ ciplot=lineplot([p.counter-(length(p.convergence_history)
190          ↵ p.convergence_history, title="Convergence
191          ↵ Interval Recent History",
192          ↵ xlabel="Iterate",ylabel="CI", color=:yellow)
193          cilines=nrows(ciplot.graphics)+5
194          show(p.output, ciplot); println()
195      end
196      return wklines + tunelines + cilines
197  end
198
199  function display_lower_dash(p::ProgressNS)
200      lhlines = liwilines = ensemblelines = srclines = 0
201      if p.lh_disp
202          lhplot=lineplot(p.e.log_Li[2:end], title="Contour
203          ↵ History", xlabel="Iterate", color=:magenta,
204          ↵ name="Ensemble logLH")
205          lineplot!(lhplot, [p.e.naive_lh for it in
206          ↵ 1:length(p.e.log_Li[2:end])], name="Naive
207          ↵ logLH")
208          lhlines=nrows(lhplot.graphics)+5
209          show(p.output, lhplot); println()
210      end
211      if p.liwi_disp && p.counter>2
212          ↵ liwiplot=lineplot([max(2,p.counter-(CONVERGENCE_MEMORY-1)
213          ↵ title="Recent iterate evidentiary weight",
214          ↵ xlabel="Iterate", name="Ensemble log Liwi",
215          ↵ color=:cyan)
216          liwilines=nrows(liwiplot.graphics)+5
217          show(p.output, liwiplot); println()
218      end
219      p.ens_disp && (ensemblelines=show(p.output, p.e,
220          ↵ progress=true))
221      p.src_disp && (println("MLE Model
222          ↵ Sources:");srclines=show(p.output, p.top_m,
223          ↵ nsr=p.no_displayed_srcs, progress=true))
224  return lhlines + liwilines + ensemblelines + srclines
225 end

```

---

### 16.7.16 /src/utilities/synthetic\_genome.jl

```

1 function synthetic_sample(no_obs::Integer, obsl,
2   ↵ bhmm_vec::AbstractVector{<:BHMM}, bhmm_dist::Categorical,
3   ↵ spikes::AbstractVector{<:AbstractMatrix{<:AbstractFloat}}},
4   ↵ spike_instruct::AbstractVector{<:Tuple{<:Bool, <:Tuple}})
5   !(typeof(obsl) <:UnitRange || typeof(obsl) <:Integer) &&
6     ↵ throw(ArgumentError("obsl must be Integer or UnitRange"))
7   length(spike_instruct)≠length(spikes) &&
8     ↵ throw(ArgumentError("spike_instruct must be as long as spikes"))
9
10
11  obs, hmm_truth=obs_array_from_bhmms(no_obs, obsl, bhmm_vec, bhmm_dist)
12
13
14  spike_truth = spike_obs!(obs, spikes, spike_instruct)
15
16
17  bg_scores = score_synthetic(obs, bhmm_vec, hmm_truth)
18
19
20  return obs, bg_scores, hmm_truth, spike_truth
21
22 end
23
24
25 function obs_array_from_bhmms(no_obs, obsl, bhmm_vec, bhmm_dist)
26   obss=zeros(UInt8,max(obsl...)+1, no_obs)
27   truthvec=zeros(UInt8,no_obs)
28   for o in 1:no_obs
29     hmm_idx=rand(bhmm_dist)
30     truthvec[o]=hmm_idx
31     bhmm=bhmm_vec[hmm_idx]
32     typeof(obsl)<:UnitRange ? (l=rand(obsl)) : (l=obsl)
33     obs[1:l,o]=rand(bhmm, l)
34   end
35   return obs, truthvec
36
37 end
38
39 function spike_obs!(obs, spikes, spike_instruct)
40   truthmat=false.(size(obs,2),length(spikes))
41   for (s,spike) in enumerate(spikes)
42     structural,ins=spike_instruct[s]
43     structural ? (truthmat[:,s] = spike_struct!(obs, spike, ins...)) :
44       ↵ (truthmat[:,s] = spike_irreg!(obs, spike, ins...))
45   end
46   return truthmat
47
48 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     end
77     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,:]))
86     end
87     return lh_mat
88 end

```

---

### 16.7.17 /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()

```

```

28     show(ls)
29     progress && return nrows(pers.graphics)+nrows(ls.graphics)+7
30 end
31
32     function persistcolor(z, zmin, zmax)
33         z=1. && return 154
34         z=0. && return 160
35     end

```

---

### 16.7.18 /src/utilities/worker\_sequencer.jl

```

1 #waits for one worker to begin before calling the next so that network is
  ↵ not slammed trying to send huge arrays to many workers simultaneously
2
3 function sequence_workers(wk_pool, func, args ... )
4     comms_chan = RemoteChannel(()→Channel{Integer}(length(wk_pool)))
5
6     for worker in wk_pool
7         remote_do(func, worker, args ... , comms_chan)
8         wait(comms_chan); report=take!(comms_chan)
9         @assert worker==report
10    end
11 end

```

---

### 16.7.19 /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}}("TTACGCACAGATGTTAC")]
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     consolidate_two =
34     ↳ [(log.(src_ATG),0),(log.(src_ATG),0),(log.(src_ATG),0)]
35     cons_two_mix = BitMatrix([true false false
36                               false true false
37                               false false true])

38     distance_model =
39     ↳ [(log.(src_TTAC),0),(log.(src_CAG),0),(log.(src_GCA),0)]

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     dpath=randstring()
49     serialize(dpath, dmodel)
50     drec=Model_Record(dpath,dmodel.log_Li)

#check distance calculation
51     pwmtest_1=zeros(1,4);pwmtest_1[1]=1.
52     pwmtest_2=zeros(1,4);pwmtest_2[2]=1.

```

```

51 @test pwm_distance(log.(pwmtest_1),log.(pwmtest_2)) =
52   ↳ euclidean(pwmtest_1,pwmtest_2) = 1.4142135623730951
53
54 #test consolidate check
55 @test consolidate_check(distance_model) =
56   ↳ (true,Dict{Integer,Vector{Integer}}())
57 @test consolidate_check(consolidate_one) = (false,
58   ↳ Dict{Integer,Vector{Integer}}(1⇒[3]))
59 @test consolidate_check(consolidate_two) = (false,
60   ↳ Dict{Integer,Vector{Integer}}(1⇒[2,3], 2⇒[3]))
61
62 #test overall consolidate function
63 _,con_idxs=consolidate_check(consolidate_one)
64 c1consmod=consolidate_srcs(con_idxs, c1model, obs, obsl, bg_scores,
65   ↳ drec.log_Li, [drec])
66
67 @test consolidate_check(c1consmod.sources)[1]
68 @test c1consmod.log_Li > dmodel.log_Li
69 @test all(c1consmod.mix_matrix[:,1])
70 @test c1consmod.sources[1][1]=log.(src_ATG)
71 @test c1consmod.sources[3][1]≠log.(src_ATG)
72 @test c1consmod.origin=="consolidated c1"
73
74 _,con_idxs=consolidate_check(consolidate_two)
75 c2consmod=consolidate_srcs(con_idxs, c2model, obs, obsl, bg_scores,
76   ↳ drec.log_Li, [drec])
77
78 @test consolidate_check(c2consmod.sources)[1]
79 @test c2consmod.log_Li > dmodel.log_Li
80 @test all(c2consmod.mix_matrix[:,1])
81 @test c2consmod.sources[1][1]=log.(src_ATG)
82 @test c2consmod.sources[2][1]≠log.(src_ATG)
83 @test c2consmod.sources[3][1]≠log.(src_ATG)
84 @test c2consmod.origin=="consolidated c2"
85
86 rm(dpath)
87 end

```

---

### 16.7.20 /test/ensemble\_tests.jl

---

```

1 @testset "Ensemble assembly and nested sampling functions" begin
2     ensambledir = randstring()

```

```

3 spensemblendir = 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])
mix_prior=.5
19
20 bg_scores = log.(fill(.1, (30,4)))
21
22   ↵ obs=[BioSequences.LongSequence{DNAAlphabet{2}}("CCGTTGACGATGTGATGAATAATGAAAGA")
23
24   ↵ BioSequences.LongSequence{DNAAlphabet{2}}("CCCCGATGATGACCGTTGACCAGATGGATG")
25
26   ↵ BioSequences.LongSequence{DNAAlphabet{2}}("CCCCGATGATGACCCGATTTGAAAAAAA")
27
28   ↵ BioSequences.LongSequence{DNAAlphabet{2}}("TCATCATGCTGATGATGAATCAGATGAAAG")
29 ]
30
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 ensemble = IPM_Engine(ensembedir, 150, source_priors,
36   ↵ (falses(0,0),mix_prior), bg_scores, obs, src_length_limits)
37 ensemble = IPM_Engine(ensembedir, 200, source_priors,
38   ↵ (falses(0,0),mix_prior), bg_scores, obs, src_length_limits) #test
39   ↵ resumption
40
41 sp_ensemble = IPM_Engine(spensemblendir, 200, source_priors,
42   ↵ (falses(0,0),mix_prior), bg_scores, obs, src_length_limits,
43   ↵ posterior_switch=true)
44
45

```

```

36     @test length(ensemble.models) == 200
37     for model in ensemble.models
38         @test -350 < model.log_Li < -150
39     end
40
41     @test length(sp_ensemble.models) == 200
42     for model in sp_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_Engsemble(assembler, distdir, 150, source_priors,
51     ↪ (falses(0,0),mix_prior), bg_scores, obs, src_length_limits)
52     dist_ensemble=IPM_Engsemble(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
60
61     rmprocs(assembler)
62     rm(distdir, recursive=true)
63
64     models_to_permute = 600
65     funclimit=200
66     funcvec=full_perm_funcvec
67
68     instruct = Permute_Instruct(funcvec,
69     ↪ ones(length(funcvec))./length(funcvec),models_to_permute,200,
70     ↪ min_clmps=fill(.02,length(funcvec)))
71
72     @info "Testing convergence displays..."
73     sp_logZ = converge_ensemble!(sp_ensemble, instruct,
74     ↪ converge_factor=500., wk_disp=true, tuning_disp=true,
75     ↪ ens_disp=true, conv_plot=true, src_disp=true, lh_disp=true,
76     ↪ liwi_disp=true, max_iterates=50)
77
78     sp_ensemble=reset_ensemble!(sp_ensemble)

```

```

71
72     @info "Testing threaded convergence ... "
73     sp_logZ = converge_ensemble!(sp_ensemble, instruct,
74         → converge_factor=500., wk_disp=false, tuning_disp=false,
75         → ens_disp=false, conv_plot=false, src_disp=false, lh_disp=false,
76         → liwi_disp=false, backup=(true, 150), max_iterates=300)
77
78     @info "Testing resumption ... "
79     sp_logZ = converge_ensemble!(sp_ensemble, instruct,
80         → converge_factor=500., wk_disp=false, tuning_disp=false,
81         → ens_disp=false, conv_plot=false, src_disp=false, lh_disp=false,
82         → liwi_disp=false, backup=(true, 150))
83     @test length(sp_ensemble.models) == 200
84     @test length(sp_ensemble.log_Li) == length(sp_ensemble.log_Xi) ==
85         → length(sp_ensemble.log_wi) == length(sp_ensemble.log_Liwi) ==
86         → length(sp_ensemble.log_Zi) == length(sp_ensemble.Hi) ==
87         → sp_ensemble.model_counter-200
88     for i in 1:length(sp_ensemble.log_Li)-1
89         @test sp_ensemble.log_Li[i] ≤ sp_ensemble.log_Li[i+1]
90     end
91     for i in 1:length(sp_ensemble.log_Zi)-1
92         @test sp_ensemble.log_Zi[i] ≤ sp_ensemble.log_Zi[i+1]
93     end
94     @test sp_logZ > -1500.0
95
96
97     @info "Testing multiprocess convergence ... "
98     @info "Spawning worker pool..."
99     worker_pool=addprocs(2, topology=:master_worker)
100    @everywhere using BioMotifInference
101
102    #####CONVERGE#####
103    final_logZ = converge_ensemble!(ensemble, instruct, worker_pool,
104        → converge_factor=500., wk_disp=false, tuning_disp=false,
105        → ens_disp=false, conv_plot=false, src_disp=false, lh_disp=false,
106        → liwi_disp=false, backup=(true,500), clean=(true, 500, 1000))
107
108    convits=length(ensemble.log_Li)
109
110    #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     @test length(ensemble.models) == 200
109     @test length(ensemble.log_Li) == length(ensemble.log_Xi) ==
110         ← length(ensemble.log_wi) == length(ensemble.log_Liwi) ==
111         ← length(ensemble.log_Zi) == length(ensemble.Hi) ==
112         ← ensemble.model_counter-200
113     for i in 1:length(ensemble.log_Li)-1
114         @test ensemble.log_Li[i] ≤ ensemble.log_Li[i+1]
115     end
116     for i in 1:length(ensemble.log_Zi)-1
117         @test ensemble.log_Zi[i] ≤ ensemble.log_Zi[i+1]
118     end
119     @test typeof(final_logZ) == Float64
120     @test final_logZ > -1500.0
121
122     rm(ensembedir, recursive=true)
123     rm(spensembedir, recursive=true)
124
125 end

```

---

### 16.7.21 /test/lh\_perf.jl

---

```

1 using BenchmarkTools, BioMotifInference, BioBackgroundModels,
2     ← BioSequences
3 println(Threads.nthreads())
4 source_pwm = [.7 .1 .1 .1
5             .1 .1 .1 .7
6             .1 .1 .7 .1]
7 source_pwm_2 = [.6 .1 .1 .2
8                 .2 .1 .1 .6
9                 .1 .2 .6 .1]
10 sources = [(log.(source_pwm), 1),(log.(source_pwm_2), 1)]
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))

```

---

### 16.7.22 /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],
    → source_stop)
23    @test score_matrices_ds[1] = [0.0 -Inf; 0.0 -Inf; 0.0 -Inf; 0.0
    → -Inf; 0.0 -Inf]

24

```

```

25 score_sources_ss!(score_mat_ss, score_matrices_ss, ss_srcs, obs[:,1],
26   ↵ source_stop)
27
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]
46 target_s2 = [.6^3 (.2*.1^2); (.2*.1^2) (.2*.1^2); (.2*.1^2)
47   ↵ (.2*.6^2); .6^3 (.2*.1^2); (.2*.1^2) (.2*.1^2); (.2*.1^2)
48   ↵ (.2*.6^2); .6^3 (.2*.1^2); (.2*.1^2) (.2*.1^2); (.2*.1^2)
49   ↵ (.2*.6^2); .6^3 (.2*.1^2)]
50 log_pwms = [log.(source_pwm), log.(source_pwm_2)]
51
52 source_start = 1
53 source_stop = 10
54
55 #score preallocates
56 ss_srcs = log_pwms
57 ds_srcs=[cat(pwm,revcomp_pwm(pwm),dims=3) for pwm in log_pwms]
58 score_mat_ds=zeros(source_stop,2)
59 score_mat_ss=zeros(source_stop,1)
60
61 score_sources_ds!(score_mat_ds, score_matrices_ds, ds_srcs, obs[:,1],
62   ↵ [source_stop, source_stop])
63 @test isapprox(score_matrices_ds[1], log.(target_s1))
64 @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
65     #test score weaving and IPM likelihood calculations
66     #trivial example using fake cardinality_penalty
67     target=logaddexp(log(.25), 0.)
68     lh_vec=zeros(2)
69     obsl=1
70     bg_scores=view([log(.25)],:,:)
71     score_mat=[[0.]]
72     osi=[1]
73     source_wmls=[1]
74     lme=0.
75     cardinality_penalty=0. #not correct but for test purposes
76     osi_emit=Vector{Int64}()
77
78     @test weave_scores_ss!(lh_vec, obsl, bg_scores, score_mat, osi,
79     ↳ source_wmls, lme, cardinality_penalty, osi_emit) = target
80
81     target2=logsumexp([log(.25), log(.5), log(.5)])
82     lh_vec=zeros(2)
83     weavevec=zeros(3)
84     score_mat=[zeros(1,2)]
85     empty!(osi_emit)
86     lme=log(.5)
87
88     @test weave_scores_ds!(weavevec, lh_vec, obsl, bg_scores, score_mat,
89     ↳ osi, source_wmls, lme, cardinality_penalty, osi_emit) = target2
90
91     #test more complicated weaves
92     obs=[BioSequences.LongSequence{DNAAlphabet{2}}("ATGATGATGATG")
93           BioSequences.LongSequence{DNAAlphabet{2}}("TGATGATGATGA")]
94     order_seqs = BioBackgroundModels.get_order_n_seqs(obs, 0)
95     coded_seqs = BioBackgroundModels.code_seqs(order_seqs)
96     obs=Array(transpose(coded_seqs))
97
98     sources = [(log.(source_pwm), 1),(log.(source_pwm_2), 1)]
99     dbl1_sources=[(log.(source_pwm), 1),(log.(source_pwm), 1)]
100
101    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
100      ↵  dbl1_sources]

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; .1^3
104    ↵  (.1*.7^2); .7^3 .1^3; .1^3 .1^3; .1^3 (.1*.7^2); .7^3 .1^3; .1^3
105    ↵  .1^3]
106
107  target_o2_s2 = [( .2*.1^2) (.2*.1^2); ( .2*.1^2) (.2*.6^2); .6^3
108    ↵  ( .2*.1^2); ( .2*.1^2) (.2*.1^2); ( .2*.1^2) (.2*.6^2); .6^3
109    ↵  ( .2*.1^2); ( .2*.1^2) (.2*.1^2); ( .2*.1^2) (.2*.6^2); .6^3
110    ↵  ( .2*.1^2); ( .2*.1^2) (.2*.1^2)]  

111
112  score_mat_ds=zeros(source_stop,2)
113  score_matrices_ds=Vector{Matrix{Float64}}(undef, 2)

114  score_sources_ds!(score_mat_ds, score_matrices_ds, ds_srcs, obs[:,1],
115    ↵  source_stops)

116  o1_mats = copy(score_matrices_ds)

117  @test isapprox(o1_mats, [log.(target_s1), log.(target_s2)])

118  score_sources_ds!(score_mat_ds, score_matrices_ds, ds_srcs, obs[:,2],
119    ↵  source_stops)

120  o2_mats = copy(score_matrices_ds)

121  @test isapprox(o2_mats, [log.(target_o2_s1), log.(target_o2_s2)])

122  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     empty!(osi_emit)

140
141     o2_lh = weave_scores_ds!(weavevec, lh_vec, 12, view(bg_scores,:,:2),
142                               ↳ o2_mats, osi, source_wmls, log_motif_expectation,
143                               ↳ cardinality_penalty, osi_emit)

144     lh,cache = IPM_likelihood(sources, obs, [12,12], bg_scores,
145                               ↳ trues(2,2), true,true)
146     @test isapprox(o1_lh,cache[1])
147     @test isapprox(o2_lh,cache[2])
148     @test isapprox(lps(o1_lh,o2_lh),lh)

149     #test source penalization
150     score_sources_ds!(score_mat_ds, score_matrices_ds, dbl1_srcs,
151                       ↳ obs[:,1], source_stops)

152
153     dbl_score_mat=copy(score_matrices_ds)

154
155     empty!(osi_emit)

156
157     naive=weave_scores_ds!(weavevec, lh_vec, 12,
158                               ↳ view(bg_scores,:,:1),Vector{Matrix{Float64}}(), Vector{Int64}(),
159                               ↳ Vector{Int64}(), log_motif_expectation, cardinality_penalty,
160                               ↳ osi_emit)

161
162     empty!(osi_emit)

163
164     single=weave_scores_ds!(weavevec, lh_vec, 12, view(bg_scores,:,:1),
165                               ↳ [dbl_score_mat[1]], [1], [3], log_motif_expectation,
166                               ↳ cardinality_penalty, osi_emit)

167
168     empty!(osi_emit)

169
170     double=weave_scores_ds!(weavevec, lh_vec, 12, view(bg_scores,:,:1),
171                               ↳ dbl_score_mat, [1,2], [3,3], log_motif_expectation,
172                               ↳ cardinality_penalty, osi_emit)

173
174     empty!(osi_emit)

```

```

164
165     triple=weave_scores_ds!(weavevec, lh_vec, 12, view(bg_scores,:,:1),
166     ↳ [dbl_score_mat[1] for i in 1:3], [1,2,3], [3,3,3],
167     ↳ log_motif_expectation, cardinality_penalty, osi_emit)
168
169     @test (single-naive) > (double-single) > (triple-double)
170
171     naive_target=-16.635532333438686
172     naive = IPM_likelihood(sources, obs, [findfirst(iszero,obs[:,o])-1
173     ↳ for o in 1:size(obs)[2]],
174     ↳ bg_scores,falses(size(obs)[2],length(sources)))
175     @test naive==naive_target
176
177     #test IPM_likelihood clean vector and cache calculations
178     mix_matrix=trueones(2,2)
179     baselh,basecache = IPM_likelihood(sources, obs, [12,12], bg_scores,
180     ↳ mix_matrix, true,true)
181
182     clean=[true,false]
183
184     unchangedlh,unchangedcache=IPM_likelihood(sources, obs, [12,12],
185     ↳ bg_scores, mix_matrix, true, true, basecache, clean)
186
187     changed_lh,changedcache=IPM_likelihood(sources, obs, [12,12],
188     ↳ bg_scores, BitMatrix([true true; false false]), true, true,
189     ↳ basecache, clean)
190
191     indep_lh, indepcache=IPM_likelihood(sources, obs,[12,12], bg_scores,
192     ↳ BitMatrix([true true; false false]), true, true)
193
194     @test baselh==unchangedlh!=changed_lh==indep_lh
195     @test basecache==unchangedcache!=changedcache==indepcache
196
197     #check that IPM_likelihood works in single stranded operation
198     IPM_likelihood(sources, obs,[12,12], bg_scores, BitMatrix([true true;
199     ↳ false false]), false)
200 end

```

---

### 16.7.23 /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=falses(0)
36  src_mixvec[Int(ceil(0/2)):end]=true
37  @test most_similar(src_mixvec,compare_mix)=test_idx
38 end

```

---

### 16.7.24 /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        ~ (falses(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
36    @test ps_model.sources != test_model.sources
37    @test ps_model.mix_matrix == test_model.mix_matrix
38    @test ps_model.origin == "PS from test"
39
40    pm_model= permute_mix(test_model, obs, obsl, bg_scores,
41        ~ test_model.log_Li, iterates=1000)
42    @test pm_model.log_Li > test_model.log_Li
43    @test pm_model.sources == test_model.sources
44    @test pm_model.mix_matrix != test_model.mix_matrix
45    @test "PM from test" == pm_model.origin

```

```

43 psfm_model=perm_src_fit_mix(test_model, Vector{Model_Record}(),obs,
44   ↪ obsl, bg_scores, test_model.log_Li, source_priors,
45   ↪ iterates=1000)
46 @test psfm_model.log_Li > test_model.log_Li
47 @test psfm_model.sources ≠ test_model.sources
48 @test psfm_model.mix_matrix ≠ test_model.mix_matrix
49 @test "PSFM from test" = psfm_model.origin

50 fm_model=fit_mix(test_model, Vector{Model_Record}(),
51   ↪ obs,obsl, bg_scores)
52 @test fm_model.log_Li > test_model.log_Li
53 @test fm_model.sources = test_model.sources
54 @test fm_model.mix_matrix ≠ test_model.mix_matrix
55 @test "FM from test" = fm_model.origin
56 @test fit_mix in fm_model.permute_blacklist

57 post_fm_psfm=perm_src_fit_mix(fm_model, Vector{Model_Record}(),obs,
58   ↪ obsl, bg_scores, test_model.log_Li, source_priors, iterates=1000)
59 @test "PSFM from candidate" = post_fm_psfm.origin
60 @test fit_mix in post_fm_psfm.permute_blacklist

61 rd_model=random_decorrelate(test_model, Vector{Model_Record}(),obs,
62   ↪ obsl, bg_scores, test_model.log_Li, source_priors, iterates=1000)
63 @test rd_model.log_Li > test_model.log_Li
64 @test rd_model.sources ≠ test_model.sources
65 @test rd_model.mix_matrix ≠ test_model.mix_matrix
66 @test "RD from test" = rd_model.origin

67 rs_model=reinit_src(test_model, Vector{Model_Record}(),obs, obsl,
68   ↪ bg_scores, test_model.log_Li, source_priors, iterates=1000)
69 @test rs_model.log_Li > test_model.log_Li
70 @test rs_model.sources ≠ test_model.sources
71 @test rs_model.mix_matrix ≠ test_model.mix_matrix
72 @test "RS from test" = rs_model.origin

73
74 → erosion_sources=[(log.(source_pwm),1),(log.(source_pwm_2),1),(log.(pwm_to_ero
75
76 eroded_mix=true(2,3)

77 erosion_lh=IPM_likelihood(erosion_sources,obs,obsl, bg_scores,
    ↪ eroded_mix)

```

```

78     erosion_model=ICA_PWM_Model("erode", "", erosion_sources,
    ↳ test_model.source_length_limits, eroded_mix, erosion_lh)

79
80     eroded_model=erode_model(erosion_model, Vector{Model_Record}(), obs,
    ↳ 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,
    ↳ 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
    ↳ 0.02 0.94 0.02]), 1),
112     (log.([0.94 0.02 0.02 0.02; 0.02 0.07 0.07 0.84; 0.02 0.02 0.94
    ↳ 0.02]), 1),
113     (log.([0.95 0.01 0.02 0.02; 0.02 0.02 0.02 0.94; 0.02 0.02 0.94
    ↳ 0.02]), 1)]

```

```

115     merger_mix = BitMatrix([true false false
116                             true true false])
117
118     accurate_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
147     @test all([src=merger_base.sources[n] ||
148                → src=merger_target.sources[n] for (n,src) in
149                → enumerate(sm_model.sources)])
150     @test "SM from merge" == sm_model.origin
151
152     shuffled_model=shuffle_sources(merger_base, [test_record], obs, obsl,
153                               → bg_scores, merger_base.log_Li)
154     @test
155         → sum(shuffled_model.sources.=merger_base.sources)=length(shuffled_model.sources)
156     for (s,src) in enumerate(shuffled_model.sources)

```

```

145     @test src==merger_base.sources[s] ||
146         → src==merger_target.sources[s]
147     if src == merger_base.sources[s]
148         @test
149             → shuffled_model.mix_matrix[:,s]==merger_base.mix_matrix[:,s]
150     else
151         @test
152             → shuffled_model.mix_matrix[:,s]==merger_target.mix_matrix[:,s]
153     end
154 end
155
156 acc_base_mix=false(2,1);acc_base_mix[1,1]=true
157 acc_merge_mix=false(2,1);acc_merge_mix[2,1]=true
158
159 acc_base=ICA_PWM_Model("accbase", "",
160     → [merger_srcs[1]],src_length_limits, acc_base_mix,
161     → IPM_likelihood([merger_srcs[1]],obs,obsl, bg_scores,
162     → acc_base_mix))
163 acc_merge=ICA_PWM_Model("accmerge", "",
164     → [accurate_srcs[1]],src_length_limits, acc_merge_mix,
165     → IPM_likelihood([accurate_srcs[1]],obs,obsl, bg_scores,
166     → acc_merge_mix))
167
168 accpath=randstring()
169 acc_record = Model_Record(accpath, acc_merge.log_Li)
170 serialize(accpath, acc_merge)
171
172 acc_model=accumulate_mix(acc_base, [acc_record], obs, obsl,
173     → bg_scores, acc_merge.log_Li)
174 @test acc_model.mix_matrix=true(2,1)
175 @test acc_model.sources == acc_merge.sources
176 @test "AM from accbase" == acc_model.origin
177
178 testwk=addprocs(1)[1]
179 @everywhere import BioMotifInference
180
181 ddm_model=remotecall_fetch(distance_merge, testwk, merger_base,
182     → [test_record], obs, obsl, bg_scores, merger_base.log_Li,
183     → remote=true)
184 @test ddm_model.log_Li > merger_base.log_Li
185 @test ddm_model.sources ≠ merger_base.sources

```

```

175     @test all([src=merger_base.sources[n] ||
176             src=merger_target.sources[distance_dict[n]] for (n,src) in
177             enumerate(ddm_model.sources)])
178     @test "DM from merge" == ddm_model.origin
179
180
181     dsm_model=remotecall_fetch(similarity_merge, testwk, merger_base,
182         [test_record], obs, obsl, bg_scores, merger_base.log_Li,
183         remote=true)
184     @test dsm_model.log_Li > merger_base.log_Li
185     @test dsm_model.sources != merger_base.sources
186     @test all([src=merger_base.sources[n] ||
187             src=merger_target.sources[n] for (n,src) in
188             enumerate(dsm_model.sources)])
189     @test "SM from merge" == dsm_model.origin
190
191
192     dshuffled_model=remotecall_fetch(shuffle_sources, testwk,
193         merger_base, [test_record], obs, obsl, bg_scores,
194         merger_base.log_Li, remote=true)
195     @test
196         sum(dshuffled_model.sources.=merger_base.sources)=length(dshuffled_model.sou
197     for (s,src) in enumerate(dshuffled_model.sources)
198         @test src=merger_base.sources[s] ||
199             src=merger_target.sources[s]
200         if src == merger_base.sources[s]
201             @test
202                 dshuffled_model.mix_matrix[:,s]=merger_base.mix_matrix[:,s]
203         else
204             @test
205                 dshuffled_model.mix_matrix[:,s]=merger_target.mix_matrix[:,s]
206         end
207     end
208     @test "SS from merge" == dshuffled_model.origin
209
210
211     dacc_model=remotecall_fetch(accumulate_mix, testwk, acc_base,
212         [acc_record], obs, obsl, bg_scores, acc_merge.log_Li)
213     @test dacc_model.mix_matrix=trues(2,1)
214     @test dacc_model.sources == acc_merge.sources
215     @test "AM from accbase" == dacc_model.origin
216
217
218     info_base=ICA_PWM_Model("info_base", "", [accurate_srcs[1]],
219         src_length_limits, accurate_mix[:,1:1],
220         IPM_likelihood([accurate_srcs[1]],obs,obsl,bg_scores,accurate_mix[:,1:1]))

```

```

202     → info_model=info_fill(info_base,Vector{Model_Record}(),obs,obsl,bg_scores,info)
203     @test info_model.sources[1][1][:] = [0., -Inf, -Inf, -Inf]
204     @test info_model.log_Li > info_base.log_Li
205     @test "IF from info_base" = info_model.origin
206
207     rmprocs(testwk)
208     rm(path)
209     rm(accpath)
210 end

```

---

### 16.7.25 /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]
20
21    #test override
22    instruct=Permute_Instruct(funcvec,
23        → [.5,.5],100,100,override_time=2.,override_weights=[.75,.25])
24    tuner=Permute_Tuner(instruct)
25    tune_weights!(tuner, [(1,1.,1.)])
26    @test tuner.inst.weights=[.5,.5] #no override
27    tuner.time_history=fill(3.,CONVERGENCE_MEMORY)
28    tune_weights!(tuner, [(1,1.,1.)])
29    @test tuner.inst.weights=[.75,.25]

```

```

27
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
38
39         ↪ clamp_pvec!(testvec,fill(.02,length(testvec)),fill(1.,length(testvec)))
40 @test isprobvec(testvec)
41 @test testvec==target
42
43
44     #high clamp
45     testvec=[.9,.08,.02]
46     clamp_pvec!(testvec, fill(.05,length(testvec)), [.45,.45,.45])
47     @test testvec=[.45, .2825, .2675]
48 end

```

---

### 16.7.26 /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)
10    end
11
12    bad_input = wm_input .* 2
13    @test_throws DomainError estimate_dirichlet_prior_on_wm(bad_input)
14
15    wm_input = [.1 .2 .3 .4; .1 .2 .3 .4]
16    est_dirichlet_vec = estimate_dirichlet_prior_on_wm(wm_input)
17    @test typeof(est_dirichlet_vec) == Vector{Dirichlet{Float64}}
18    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
50     permuted_weight_sources[1]=permute_source_weights(permuted_weight_sources[1],
51
52     permuted_weight_sources[2]=permute_source_weights(permuted_weight_sources[2],
53 @test permuted_weight_sources != test_sources
54 for (s,source) in enumerate(permuted_weight_sources)
55     for pos in 1:size(source[1],1)
56         @test isprobvec(exp.(source[1][pos,:]))
57         @test source[1][pos,:] != test_sources[s][1][pos,:]
58     end
59 end
60

```



```

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.]
121
122 end

```

---

### 16.7.27 /test/runtests.jl

---

```

1 @info "Loading test packages ... "
2
3 using BioMotifInference, BioBackgroundModels, BioSequences,
4   ↳ 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!"

```

---

### 16.7.28 /test/spike\_recovery.jl

---

```

1 #Testbed for synthetic spike recovery from example background
2
3 using nnlearn, BGHMM, HMMBase, Distributions, Random, Serialization,
→   Distributions
4
5 Random.seed!(786)
6
7 #CONSTANTS
8 no_obs=1000
9 obsl=140
10

```

```

11 hmm=HMM{Univariate,Float64}([0.4016518533961019, 0.2724399569450827,
→ 0.3138638675018568, 0.012044322156962559], [3.016523036789942e-9
→ 2.860631288328858e-6 0.2299906524188302 0.7700064839333549;
→ 3.0102278323431375e-11 0.7494895424906354 0.23378615437778671
→ 0.016724303101477486; 8.665894321573098e-17 0.2789950381410553
→ 0.7141355461949104 0.006869415664033568; 0.006872526597796038
→ 0.016052425322133648 0.017255179541768192 0.9598198685383041],
→ [Categorical([0.1582723599684065, 0.031949729618356536,
→ 0.653113286526948, 0.15666462388628763]),
→ Categorical([0.4610499748798372, 0.2613013005680122,
→ 0.15161801872560146, 0.12603070582654768]),
→ Categorical([0.08601960130086236, 0.13869192872427524,
→ 0.26945182329686523, 0.5058366466779973]),
→ Categorical([0.3787366527613563, 0.11034604356978756,
→ 0.1119586976058099, 0.3989586060630392])))

12
13 struc_sig=[.1 .7 .1 .1
14           .1 .1 .1 .7
15           .1 .7 .1 .1]
16 periodicity=8
17 struc_frac_obs=.35
18
19 tata_box=[.05 .05 .05 .85
20           .85 .05 .05 .05
21           .05 .05 .05 .85
22           .85 .05 .05 .05
23           .425 .075 .075 .425
24           .85 .05 .05 .05
25           .425 .075 .075 .425]
26 tata_frac_obs=.7
27 tata_recur_range=1:4
28
29 combined_ensemble = "/bench/PhD/NGS_binaries/nnlearn/combined_ensemble"
30
31 #JOB CONSTANTS
32 const position_size = 141
33 const ensemble_size = 100
34 const no_sources = 3
35 const source_min_bases = 3
36 const source_max_bases = 12
37 @assert source_min_bases < source_max_bases
38 const source_length_range= source_min_bases:source_max_bases
39 const mixing_prior = .3

```

```

40 @assert mixing_prior ≥ 0 & mixing_prior ≤ 1
41 const models_to_permute = ensemble_size * 3
42
43 job_sets=[[
44 ([[
45     ("PS", (no_sources)),
46     ("PM", (no_sources)),
47     ("PSFM", (no_sources)),
48     ("PSFM", (no_sources*10, .8, 1.)),
49     ("FM", (())),
50     ("DM", (no_sources)),
51     ("SM", (no_sources)),
52     ("RD", (no_sources)),
53     ("RI", (no_sources)),
54     ("EM", (no_sources))
55 ], [.025, .025, .025, .025, .775, .025, .025, .025, .025, .025]),,
56 ([[
57     ("PS", (no_sources)),
58     ("PM", (no_sources)),
59     ("PSFM", (no_sources)),
60     ("PSFM", (no_sources, 0., 1.)),
61     ("PSFM", (no_sources, 0.8, .0)),
62     ("FM", (())),
63     ("DM", (no_sources)),
64     ("SM", (no_sources)),
65     ("RD", (no_sources)),
66     ("RI", (no_sources*10)),
67     ("EM", (no_sources))
68 ], [.05, .15, .15, .10, .05, .20, 0.0, 0.05, 0.15, 0.10, .0]),,
69 ]
70 job_limit=4
71
72 const prior_wt=1.1
73
74 #FUNCTIONS
75 function setup_obs(hmm, no_obs, obsl)
76     obs=vec(Int64.(rand(hmm,obsl)[2]))
77     for o in 2:no_obs
78         obs=hcat(obs, vec(Int64.(rand(hmm,obsl)[2])))
79     end
80     obs=vcat(obs,zeros(Int64,1,no_obs))
81     return obs
82 end

```

```

83
84 function spike_irreg!(obs, source, frac_obs, recur)
85     truth=Vector{Int64}()
86     for o in 1:size(obs,2)
87         if rand()<frac_obs
88             push!(truth, o)
89             for r in rand(recur)
90                 rand()<.5 && (source=nnlearn.revcomp_pwm(source))
91                 pos=rand(1:size(obs,1)-1)
92                 pwm_ctr=1
93                 while pos ≤ size(obs,1)-1&& pwm_ctr ≤ size(source,1)
94                     obs[pos,o]=rand(Categorical(source[pwm_ctr,:]))
95                     pos+=1
96                     pwm_ctr+=1
97                 end
98             end
99         end
100    return truth
101 end
102
103
104 function spike_struc!(obs, source, frac_obs, periodicity)
105     truth=Vector{Int64}()
106     truthpos=Vector{Vector{Int64}}()
107     for o in 1:size(obs,2)
108         if rand()<frac_obs
109             push!(truth, o)
110             rand()<.5 && (source=nnlearn.revcomp_pwm(source))
111             pos=rand(1:periodicity)
112             posvec=Vector{Int64}()
113             while pos ≤ size(obs,1)
114                 pos_ctrl=pos
115                 pwm_ctrl=1
116                 while pos_ctrl ≤ size(obs,1)-1&&pwm_ctrl ≤ size(source,1)
117                     push!(posvec,pos_ctrl)
118                     base=rand(Categorical(source[pwm_ctrl,:]))
119                     obs[pos_ctrl,o]=base
120                     pos_ctrl+=1
121                     pwm_ctrl+=1
122                 end
123                 pos+=periodicity+pwm_ctrl
124             end
125             push!(truthpos,posvec)

```

```

126     end
127   end
128   return truth,truthpos
129 end
130
131 function get_BGHMM_lhs(obs,hmm)
132   lh_mat=zeros(size(obs,1)-1,size(obs,2))
133   for o in 1:size(obs,2)
134     obso=zeros(Int64,size(obs,1),1)
135     obso[1:end,1] = obs[:,o]
136
137     → lh_mat[:,o]=BGHMM.get_BGHMM_symbol_lh(Matrix(transpose(obso)),hmm)
138   end
139   return lh_mat
140 end
141 @info "Setting up synthetic observation set ... "
142 obs=setup_obs(hmm, no_obs, obsl)
143 struc_truth,struc_postruth=spike_struc!(obs, struc_sig, struc_frac_obs,
144   → periodicity)
145 irreg_truth=spike_irreg!(obs, tata_box, tata_frac_obs, tata_recur_range)
146
147 @info "Calculating background likelihood matrix ... "
148 bg_lhs=get_BGHMM_lhs(obs,hmm)
149
150 @info "Assembling source priors ... "
151 #prior_array= [struc_sig, tata_box]
152 prior_array= Vector{Matrix{Float64}}(){}
153 source_priors = nnlearn.assemble_source_priors(no_sources, prior_array,
154   → prior_wt, source_length_range)
155
156 @info "Assembling ensemble ... "
157 path=randstring()
158 if isfile(string(path,'/',"ens"))
159   ens = deserialize(string(path,'/',"ens"))
160   job_set_thresh=[-Inf,ens.naive_lh]
161   param_set=(job_sets,job_set_thresh,job_limit)
162
163 @info "Converging ensemble ... "

```

```

164 nnlearn.ns_converge!(ens, param_set, models_to_permute, .0001,
    ↳ model_display=no_sources, backup=(true,5))
165
166 rm(path,recursive=true)
167
168 #811973

```

---

## 16.8 AWSWrangler

### 16.8.1 /src/AWSWrangler.jl

```

1 module AWSWrangler
2
3 using AWSSCore, AWSSDK, Dates, Sockets
4
5 function spot_wrangle(no_instances::Integer, spot_price::AbstractFloat,
    ↳ sgrp_name::String, sgrp_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     sgrp=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  end
38
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

```

---

## 16.9 Analyses

### 16.9.1 /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 pne_pth="/bench/PhD/NGS_binaries/BSS/A10/pop_norm"
8 plne_pth="/bench/PhD/NGS_binaries/BSS/A10/pop_lognorm"
9 vne_pth="/bench/PhD/NGS_binaries/BSS/A10/vol_norm"
10 vlne_pth="/bench/PhD/NGS_binaries/BSS/A10/vol_lognorm"
11
12 a10df=DataFrame(CSV.read(a10pth))
13 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
   ↳ eachrow(a10df)]
14
15 X=Vector{Float64}()
16 measure_dict=Dict{String,Vector{Vector{Float64}}}()
17
18 measure_dict["PopEst"]=Vector{Vector{Float64}}()
19 measure_dict["VolEst"]=Vector{Vector{Float64}}()
20
21 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
22 for t_df in groupby(a10df, "Time point (d)")
23     push!(X,Float64(unique(t_df."Time point (d)")[1]))
24     pevec,tvec,dvec,vvec,vevec, rtvec, rplvec, rptvec, ldvec, ondvec,
   ↳ onlvec, oplvec, inlvec, iplvec, gclvec = [zeros(0) for i in 1:15]
25
26     for i_df in groupby(t_df,"BSR")

```

```

27     length([skipmissing(i_df."Lens diameter") ... ])>0 &&
28         → push!(ldvec,mean(skipmissing(i_df."Lens diameter")))
29 end
30
31 for i_df in groupby(t_df,"BSR")
32     if length([skipmissing(i_df."CMZ Sum") ... ])>0
33         push!(tvec,mean(skipmissing(i_df."CMZ Sum")))
34         if length([skipmissing(i_df."Lens diameter") ... ])>0
35             push!(pevec,mean(skipmissing(i_df."CMZ
36                 → Sum"))*mean(skipmissing(i_df."Lens diameter"))/14.)
37         else
38             push!(pevec,mean(skipmissing(i_df."CMZ
39                 → Sum"))*mean(ldvec)/14.)
40         end
41     end
42     if length([skipmissing(i_df."Retina thickness") ... ])>0 &&
43         length([skipmissing(i_df."IPL") ... ])>0 &&
44         length([skipmissing(i_df."OPL") ... ])>0 &&
45         length([skipmissing(i_df."RPE length") ... ])>0
46
47         rthi=mean(skipmissing(i_df."Retina
48             → thickness"))-mean(skipmissing(i_df."OPL"))-mean(skipmissing(i_df."OPL"))
49         rpel=mean(skipmissing(i_df."RPE length"))
50         if length([skipmissing(i_df."Lens diameter") ... ])>0
51             rcirc=rpel+mean(skipmissing(i_df."Lens diameter"))
52         else
53             rcirc=rpel+mean(ldvec)
54         end
55
56         or=rcirc/2π
57         ir=or-.5*rthi
58
59         ov=(4/3)*π*(or^3)
60         iv=(4/3)*π*(ir^3)
61
62         push!(vevec,(ov-iv)*(4/5))
63     end
64 end
65
66 push!(measure_dict["PopEst"],filter(p→p>0,pevec))
67 push!(measure_dict["VolEst"],vevec)
68 end

```

```

66 x=measure_dict["PopEst"][1]
67 uncorrX3d=ones(length(x),1)
68 corrX3d=hcat(ones(length(x)),x)
69 y=measure_dict["VolEst"][1]
70 uncorr=BayesianLinearRegression.fit!(BayesianLinReg(uncorrX3d,y))
71 corr=BayesianLinearRegression.fit!(BayesianLinReg(corrX3d,y))

72
73 function blr_plt(x,y,Xs,blrs,colors,labels,q=.975)
74
    ↪ plt=scatter(x,y,marker=:diamond,markersize=8,markerstrokestyle=:bold,markerco
    ↪ "retinal volume", xlabel="Estimated CMZ Annulus
    ↪ Population", legend=:bottomleft)
75   for (X,blr,color,label) in zip(Xs, blrs,colors,labels)
76     line=zeros(size(X,1))
77     ribbon=zeros(size(X,1))
78     predictions=BayesianLinearRegression.predict(blr,X)
79     for (n,p) in enumerate(predictions)
80       normal=Normal(value(p),uncertainty(p))
81       ribbon[n]=quantile(normal,q)-mean(normal)
82       line[n]=mean((value(p)))
83     end
84     plot!(plt,x,line,ribbon=ribbon,color=color,label=label)
85   end
86   return plt
87 end
88
89 p=blr_plt(x,y,[corrX3d,uncorrX3d],[corr,uncorr],[:darkmagenta,:green],[ "Correlated
    ↪ Model", "Uncorrelated Model"])
90
91 savefig(p, "/bench/PhD/Thesis/images/cmz/3dcorr.png")
92 @info "Saved fig."
93
94 println("\\begin{tabular}{|l|l|l|l|}")
95 println("\\hline")
96 println("{\\bf Age (dpf)} & {\\bf Uncorrelated logZ} & {\\bf Correlated
    ↪ logZ} & {\\bf logZR}\\\\ \\hline")
97
98 for (x,xs,ys) in zip(X,measure_dict["PopEst"],measure_dict["VolEst"])
99   uncorrX=ones(length(xs),1)
100  corrX=hcat(ones(length(xs)),xs)
101  uncorr=BayesianLinearRegression.fit!(BayesianLinReg(uncorrX,ys))
102  corr=BayesianLinearRegression.fit!(BayesianLinReg(corrX,ys))

```

```
103     println("$x & {\bf $(round(logEvidence(uncorr),digits=3))} &
           ← $(round(logEvidence(corr),digits=3)) &
           ← $(round(logEvidence(uncorr)-logEvidence(corr),digits=3))\\\\\\\
           ← \hline")
104 end
105
106 println("\end{tabular}")
107
108
109
```

## 16.9.2 /cmz/a10dvratio.jl

```

1 using
   ↳ BayesianLinearRegression,Csv,DataFrames,Distributions,BioSimpleStochastic,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 &&
21             ↳ mean(skipmissing(i_df."Dorsal CMZ (#)")) >0. &&
22             ↳ push!(dvec,mean(skipmissing(i_df."Dorsal CMZ (#)")))
23         length([skipmissing(i_df."Ventral CMZ (#)") ... ])>0 &&
24             ↳ mean(skipmissing(i_df."Ventral CMZ (#)")) >0. &&
25             ↳ push!(vvec,mean(skipmissing(i df."Ventral CMZ (#)")))

```

```

22     length(skipmissing(i_df."Dorsal CMZ (#)") ... ])>0 &&
23     mean(skipmissing(i_df."Dorsal CMZ (#)")) >0. &&
24     length(skipmissing(i_df."Ventral CMZ (#)") ... ])>0 &&
25     mean(skipmissing(i_df."Ventral CMZ (#)")) >0. &&
26     push!(rvec,mean(skipmissing(i_df."Dorsal CMZ
27     (#)"))/mean(skipmissing(i_df."Ventral CMZ (#)")))
28 end
29
30 d_logn_mts=[fit(MarginalTDist,log.(measure_dict["Dorsal CMZ (#)"][n]))
31   ↪   for n in 1:length(X)]
32 d_mean=[exp(mean(mt)) for mt in d_logn_mts]
33 d_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in d_logn_mts]
34 d_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in d_logn_mts]
35 v_logn_mts=[fit(MarginalTDist,log.(measure_dict["Ventral CMZ (#)"][n]))
36   ↪   for n in 1:length(X)]
37 v_mean=[exp(mean(mt)) for mt in v_logn_mts]
38 v_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in v_logn_mts]
39 v_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in v_logn_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 dv_chart=Plots.plot(X, d_mean, ribbon=(d_lower,d_upper), color=:green,
48   ↪   label="Dorsal mean", ylabel="CMZ sectional population size",
49   ↪   showaxis=:y, xticks=plotticks, xformatter=_→"")
50 plot!(X, v_mean, ribbon=(v_lower,v_upper), color=:darkmagenta,
51   ↪   label="Ventral mean")
52 scatter!(vcat([[X[n] for i in 1:length(measure_dict["Dorsal CMZ
53   ↪   (#)"])[n]] for n in 1:length(X)] ... ),vcat(measure_dict["Dorsal CMZ
54   ↪   (#)"] ... ), marker=:utriangle, color=:green, markersize=3,
55   ↪   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}
61             b]
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")

```

---

### 16.9.3 /cmz/a10dvslice.jl

---

```

1 using
    ↵ BayesianLinearRegression, CSV, DataFrames, Distributions, BioSimpleStochastic, GMC_NS,
    ↵ Serialization, Plots,
    ↵ KernelDensityEstimate, KernelDensityEstimatePlotting, Gadfly
2 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
3
4 edpath="/bench/PhD/NGS_binaries/BSS/A10/ed"
5 evpath="/bench/PhD/NGS_binaries/BSS/A10/ev"
6 edvpath="/bench/PhD/NGS_binaries/BSS/A10/edv"
7 paths=[edpath, evpath, edvpath]
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["CMZ Sum"]=Vector{Vector{Float64}}()
18 measure_dict["Dorsal CMZ (#)"]=Vector{Vector{Float64}}()
19 measure_dict["Ventral CMZ (#)"]=Vector{Vector{Float64}}()
20 measure_dict["Lens circumference"]=Vector{Vector{Float64}}()
21
22 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
23 for t_df in groupby(a10df, "Time point (d)")
24   if !(Float64(unique(t_df."Time point (d)")[1]) in [180., 360.])
25
26     push!(X,Float64(unique(t_df."Time point (d)")[1]))
27     tvec,dvec,vvec,lcirc = [zeros(0) for i in 1:4]
28
29     for i_df in groupby(t_df,"BSR")
30       length([skipmissing(i_df."Lens diameter")...])>0 &&
31         → push!(lcirc,mean(skipmissing(i_df."Lens diameter"))*π)
32   end
33
34     for i_df in groupby(t_df,"BSR")
35       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       length([skipmissing(i_df."Dorsal CMZ (#)")...])>0 &&
39         → mean(skipmissing(i_df."Dorsal CMZ (#)")) >0. &&
40         → push!(dvec,mean(skipmissing(i_df."Dorsal CMZ (#)")))
41       length([skipmissing(i_df."Ventral CMZ (#)")...])>0 &&
42         → mean(skipmissing(i_df."Ventral CMZ (#)")) >0. &&
43         → push!(vvec,mean(skipmissing(i_df."Ventral CMZ (#)")))
44   end
45 end

```

```

45
46 obsset=[measure_dict["Dorsal CMZ (#)"],measure_dict["Ventral CMZ
   ↵ (#)"],measure_dict["CMZ Sum"]]
47
48 dpopdist=fit(LogNormal,measure_dict["Dorsal CMZ (#)"][1])
49 vpopdist=fit(LogNormal,measure_dict["Ventral CMZ (#)"][1])
50 dvpopdist=fit(LogNormal,measure_dict["CMZ Sum"][1])
51
52 logLC=vcat([log.(measure_dict["Lens circumference"])[n]) for n in
   ↵ 1:length(X)]...)
53 logX=vcat([[log.(X[n]) for ms in measure_dict["Lens circumference"]][n]
   ↵ for n in 1:length(X)]...)
54 logX=hcat(ones(length(logX)),logX)
55
56 lm=BayesianLinearRegression.fit!(BayesianLinReg(logX,logLC))
57 w1,w2=posteriorWeights(lm)
58 factor=10^w1.val
59 power=w2.val
60
61 lm1=Lens_Model(factor,power,14.)
62 lm2=Lens_Model(factor,power,28.)
63
64 mc_its=Int64(1e6)
65 base_scale=[144.,5.]
66 base_min=[10.,0.]
67 time_scale=90
68 time_min=4.
69
70 cycle_prior=[LogNormal(log(20),log(2)),LogNormal(log(.9),log(1.6))]
71 end_prior=[Uniform(4,90)]
72
73 function compose_priors(phases)
74     prs=Vector{Vector{Distribution}} }()
75     bxes=Vector{Matrix{Float64}} }()
76     for p in phases
77         pr=[repeat(cycle_prior,p)...,repeat(end_prior,p-1)...]
78         push!(prs,pr)
79         bx=hcat([repeat(base_min,p)...,fill(time_min,p-1)...],
80                 [repeat(base_scale,p)...,fill(time_scale,p-1)...])
81         push!(bxes,GMC_NS.to_unit_ball.(bx,pr))
82     end
83     return prs,bxes
84 end

```

```

85
86 prior,box=compose_priors(2)
87
88 d_constants=[X,dpopdist,lm1,mc_its,2]
89 v_constants=[X,vpopdist,lm1,mc_its,2]
90 dv_constants=[X,dvpopdist,lm2,mc_its,2]
91 constants=[d_constants,v_constants,dv_constants]
92
93 uds=Vector{Vector{Function}}([[],[liwi_display],[convergence_display]])
94 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display],[ensemble_displa
95
96 for (pth,constants,obs) in zip(paths,constants,obsset)
97     if isfile(pth*/ens")
98         e=deserialize(pth*/ens")
99     else
100        e=Slice_Ensemble(pth,3000,obs, prior..., constants, box..., 
101                           ← GMC_DEFAULTS)
102    end
103
104 converge_ensemble!(e,backup=true,50),upper_displays=uds,
105   ← lower_displays=lds, disp_rot_its=100, mc_noise=.3,
106   ← converge_criterion="compression", converge_factor=1.)
107
108 ed=deserialize(edpath*/ens")
109 ev=deserialize(epath*/ens")
110 edv=deserialize(edvpath*/ens")
111
112 edev=measure_evidence(ed)
113 evev=measure_evidence(ev)
114 edvev=measure_evidence(edv)
115
116 evidence_ratio=edvev-(edev+evev)
117
118 println("\\begin{tabular}{|l|l|l|l|l|}")
119 println("\\hline")
120 println("{\\bf Total logZ} & {\\bf Dorsal logZ} & {\\bf Ventral logZ} &
121   ← {\\bf logZR} & {\\bf \$\sigma\$ Significance}\\\"\\\"\\\"\\hline")
122 println("\\textbf{($round(edvev,digits=3))} & $(round(edev,digits=3)) &
123   ← $(round(eevev,digits=3)) & $(round(evidence_ratio,digits=3)) &
124   ← $(round(evidence_ratio.val/evidence_ratio.err,digits=3))\\\"\\\"\\\"\\\"\\hline")
125 println("\\end{tabular}")

```

```

121
122 mapd=deserialize(ed.models[findmax([m.log_Li for m in
123   ↵ ed.models])[2]].path)
123 mapv=deserialize(ev.models[findmax([m.log_Li for m in
124   ↵ ev.models])[2]].path)
124 mapdv=deserialize(edv.models[findmax([m.log_Li for m in
125   ↵ edv.models])[2]].path)

125
126 println("\\begin{tabular}{l|l|l|l}")
127 println("\\hline")
128 println("{\\bf Parameter} & {\\bf Total MAP} & {\\bf Dorsal MAP} & {\\bf
129   ↵ Ventral MAP}\\\"\\hline")
130 println("Phase 1 \$CT\$ (h) & $(round(mapdv.θ[1], digits=1)) &
131   ↵ $(round(mapd.θ[1], digits=1)) & $(round(mapv.θ[1], digits=1))\\\\\\
132   ↵ \\hline")
133 println("Phase 1 \$\\epsilon\$ & $(round(mapdv.θ[2], digits=2)) &
134   ↵ $(round(mapd.θ[2], digits=2)) & $(round(mapv.θ[2], digits=2))\\\\\\
135   ↵ \\hline")
136 println("Phase 2 \$CT\$ (h) & $(round(mapdv.θ[3], digits=1)) &
137   ↵ $(round(mapd.θ[3], digits=1)) & $(round(mapv.θ[3], digits=1))\\\\\\
138   ↵ \\hline")
139 println("Transition age & $(round(mapdv.θ[5], digits=1)) &
140   ↵ $(round(mapd.θ[5], digits=1)) & $(round(mapv.θ[5], digits=1))\\\\\\
141   ↵ \\hline")
142 println("\\end{tabular}")

143
144 X=ed.constants[1]
145
146 plotticks=[10*i for i in 1:9]

147 catdobs=vcat([ed.obs[t] for t in 1:length(X)] ... )
148 catvobs=vcat([ev.obs[t] for t in 1:length(X)] ... )
149 catdvobs=vcat([edv.obs[t] for t in 1:length(X)] ... )

150 dXs=vcat([[X[n] for i in 1:length(ed.obs[n])] for n in 1:length(X)] ... )
151 vXs=vcat([[X[n] for i in 1:length(ev.obs[n])] for n in 1:length(X)] ... )
152 dvXs=vcat([[X[n] for i in 1:length(edv.obs[n])] for n in 1:length(X)] ... )

153
154 d_mean=mapd.disp_mat[:,2]
155 d_upper=mapd.disp_mat[:,3]-mapd.disp_mat[:,2]

```

```

150 d_lower=mapd.disp_mat[:,2]--mapd.disp_mat[:,1]
151
152 v_mean=mapv.disp_mat[:,2]
153 v_upper=mapv.disp_mat[:,3]--mapv.disp_mat[:,2]
154 v_lower=mapv.disp_mat[:,2]--mapv.disp_mat[:,1]
155
156 dv_mean=mapdv.disp_mat[:,2]
157 dv_upper=mapdv.disp_mat[:,3]--mapdv.disp_mat[:,2]
158 dv_lower=mapdv.disp_mat[:,2]--mapdv.disp_mat[:,1]
159
160 mapd_plt=scatter(dxS,catdobs, marker=:cross, color=:black, markersize=3,
    ↵ label="Dorsal CMZ population data", showaxis=:y, xticks=plotticks,
    ↵ xformatter=_→ "", ylabel="Population")
161 plot!(mapd_plt, X, d_mean, ribbon=(d_lower,d_upper), color=:green,
    ↵ label="Dorsal model population")
162 annotate!([(8,150,Plots.text("B",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=plotticks)
165 plot!(mapv_plt, X, v_mean, ribbon=(v_lower,v_upper), color=:darkmagenta,
    ↵ label="Ventral model population")
166 annotate!([(8,175,Plots.text("C",18))])
167
168 mapdv_plt=scatter(dvxs,catdvobs, marker=:cross, color=:black,
    ↵ markersize=3, label="Total CMZ population data", showaxis=:y,
    ↵ xticks=plotticks, xformatter=_→ "", ylabel="Population")
169 plot!(mapdv_plt, X, dv_mean, ribbon=(dv_lower,dv_upper),
    ↵ color=:darkorange, label="Total model population")
170 annotate!([(8,300,Plots.text("A",18))])
171
172 combined_map=Plots.plot(mapdv_plt, mapd_plt, mapv_plt, layout=grid(3,1),
    ↵ size=(800,900), link=:x)
173
174 savefig(combined_map, "/bench/PhD/Thesis/images/cmz/a10dvMAP.png")
175
176 kde=posterior_kde(edv)
177
178 ph1marg=marginal(kde,[1;2])
179 ph2marg=marginal(kde,[3;4])
180 ph12marg=marginal(kde,[1;3])
181 transmarg=marginal(kde,[5])
182

```

```

183 ph1mplt=KernelDensityEstimatePlotting.plot(ph1marg;dimLbls=[ "Phase 1 CT
    ↵ (hr)", "Phase 1 ∈ rate"], axis=[0. 144.; 0. 4.])
184 ph2mplt=KernelDensityEstimatePlotting.plot(ph2marg;dimLbls=[ "Phase 2 CT
    ↵ (hr)", "Phase 2 ∈ rate"], axis=[0. 144.; 0. 4.])
185 ph12mplt=KernelDensityEstimatePlotting.plot(ph12marg; dimLbls=[ "Phase 1
    ↵ CT (hr)", "Phase 2 CT (hr)"],axis=[0. 144.; 0. 144.])
186 tmplt=KernelDensityEstimatePlotting.plotKDE(transmarg,xlbl="Phase
    ↵ transition age (dpf)", points=false, c=["green"])
187
188 combined_marg=Gadfly.gridstack([ph1mplt ph12mplt; ph2mplt tmplt])
189 img=SVG( "/bench/PhD/Thesis/images/cmz/a10dvmarginals.svg",24cm,24cm)
190 draw(img,combined_marg)

```

---

#### 16.9.4 /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/BSS/A10/pop_norm"
5 plne_pth="/bench/PhD/NGS_binaries/BSS/A10/pop_lognorm"
6 vne_pth="/bench/PhD/NGS_binaries/BSS/A10/vol_norm"
7 vlne_pth="/bench/PhD/NGS_binaries/BSS/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 &&
25     → push!(ldvec,mean(skipmissing(i_df."Lens diameter")))
26 end
27
28 for i_df in groupby(t_df,"BSR")
29     if length(skipmissing(i_df."CMZ Sum") ... ])>0 &&
30         → mean(skipmissing(i_df."CMZ Sum")) > 0
31         if length(skipmissing(i_df."Lens diameter") ... ])>0
32             push!(pevec,mean(skipmissing(i_df."CMZ
33             → Sum"))*mean(skipmissing(i_df."Lens diameter"))/14.)
34         else
35             push!(pevec,mean(skipmissing(i_df."CMZ
36             → Sum"))*mean(ldvec)/14.)
37         end
38     end
39
40     if length(skipmissing(i_df."Retina thickness") ... ])>0 &&
41         length(skipmissing(i_df."IPL") ... ])>0 &&
42         length(skipmissing(i_df."OPL") ... ])>0 &&
43         length(skipmissing(i_df."RPE length") ... ])>0
44
45         rthi=mean(skipmissing(i_df."Retina
46         → thickness"))-mean(skipmissing(i_df."OPL"))-mean(skipmissing(i_df."IPL"))
47         rpel=mean(skipmissing(i_df."RPE length"))
48         if length(skipmissing(i_df."Lens diameter") ... ])>0
49             rcirc=rpel+mean(skipmissing(i_df."Lens diameter"))
50         else
51             rcirc=rpel+mean(ldvec)
52         end
53
54         or=rcirc/2π
55         ir=or-.5*rthi
56
57         ov=(4/3)*π*(or^3)
58         iv=(4/3)*π*(ir^3)
59
60         push!(spvec,(ov-iv)*(4/5))
61     end
62 end
63
64 push!(measure_dict["PopEst"],pevec)
65 push!(measure_dict["VolEst"],spvec)
66 end

```

```

62 gmc=GMC_DEFAULTS
63 gmc[1]=5
64 gmc[2]=1.e-15
65
66 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]])
67
68 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
69
70 evdict=Dict{String,Measurement}()
71
72 pmax_μ=2e5
73 pmin_μ=1
74 pmax_λ=1e-3
75 pmin_λ=1e-10
76 vmax_μ=1e10
77 vmin_μ=100
78 vmax_λ=1e-10
79 vmin_λ=1e-20
80
81 pn_priors=[Uniform(pmin_μ,pmax_μ),Uniform(pmin_λ,pmax_λ)]
82 pn_box=[pmin_μ pmax_μ;pmin_λ pmax_λ]
83 vn_priors=[Uniform(vmin_μ,vmax_μ),Uniform(vmin_λ,vmax_λ)]
84 vn_box=[vmin_μ vmax_μ;vmin_λ vmax_λ]
85
86 n_models=50
87
88 for (pth,prior,box,eststring) in zip([pne_pth, vne_pth],[pn_priors,
    ↪ 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),
    ↪ measure_dict[eststring][nx]), prior, box, gmc ... )
95     end
96

```

```

97     pth in keys(evdict) ? (evdict[pth] +=
  ↵   converge_ensemble!(e,backup=true,10000),upper_displays=uds,
  ↵   lower_displays=lds, disp_rot_its=10000,
  ↵   converge_criterion="compression", converge_factor=1e-6)) :
  ↵   (evdict[pth] =
  ↵   converge_ensemble!(e,backup=true,10000),upper_displays=uds,
  ↵   lower_displays=lds, disp_rot_its=10000,
  ↵   converge_criterion="compression", converge_factor=1e-6))
98 end
99 end
100
101 ln_pmax_mu=log(pmax_mu^2/sqrt(pmax_mu^2 + inv(pmax_lambda)))
102 ln_pmin_mu=log(pmin_mu^2/sqrt(pmin_mu^2 + inv(pmin_lambda)))
103 ln_pmax_lambda=1/log(1+(inv(pmax_lambda)/pmax_mu^2))
104 ln_pmin_lambda=1/log(1+(inv(pmin_lambda)/pmin_mu^2))
105
106 ln_vmax_mu=log(vmax_mu^2/sqrt(vmax_mu^2 + inv(vmax_lambda)))
107 ln_vmin_mu=log(vmin_mu^2/sqrt(vmin_mu^2 + inv(vmin_lambda)))
108 ln_vmax_lambda=1/log(1+(inv(vmax_lambda)/vmax_mu^2))
109 ln_vmin_lambda=1/log(1+(inv(vmin_lambda)/vmin_mu^2))
110
111 pln_priors=[Uniform(ln_pmin_mu,ln_pmax_mu),Uniform(ln_pmin_lambda,ln_pmax_lambda)]
112 pln_box=[ln_pmin_mu ln_pmax_mu;ln_pmin_lambda ln_pmax_lambda]
113 vln_priors=[Uniform(ln_vmin_mu,ln_vmax_mu),Uniform(ln_vmin_lambda,ln_vmax_lambda)]
114 vln_box=[ln_vmin_mu ln_vmax_mu;ln_vmin_lambda ln_vmax_lambda]
115
116 for (pth,prior,box,eststring) in zip([plne_pth, vln_pth],[pn_priors,
  ↵  vn_priors],[pn_box,vn_box],[ "PopEst", "VolEst"])
117   for (nx,x) in enumerate(x)
118     enspth=pth*"/$x"
119     if isfile(enspth*"/ens")
120       e=deserialize(enspth*"/ens")
121     else
122       ↵   e=LogNormal_Ensemble(enspth,n_models,filter(i→i>0,measure_dict[estst
  ↵   prior, box, gmc ... )
123 end
124

```

```

125     pth in keys(evdict) ? (evdict[pth] +=
126         converge_ensemble!(e, backup=true, 10000), upper_displays=uds,
127         lower_displays=lds, disp_rot_its=10000,
128         converge_criterion="compression", converge_factor=1e-6)) :
129     (evdict[pth] =
130         converge_ensemble!(e, backup=true, 10000), upper_displays=uds,
131         lower_displays=lds, disp_rot_its=10000,
132         converge_criterion="compression", converge_factor=1e-6))
133
134     end
135   end
136
137   pop_evidence_ratio=evdict[plne_pth]-evdict[pne_pth]
138   vol_evidence_ratio=evdict[vlne_pth]-evdict[vne_pth]
139
140   println("CMZ Population & $(round(evdict[pne_pth], digits=3)) &
141           $(round(evdict[plne_pth], digits=3)) &
142           $(round(pop_evidence_ratio, digits=3)) &
143           $(pop_evidence_ratio.val/pop_evidence_ratio.err)\\\\\\ \\hline")
144   println("Estimated Retinal Volume & $(round(evdict[vne_pth], digits=3)) &
145           $(round(evdict[vlne_pth], digits=3)) &
146           $(round(vol_evidence_ratio, digits=3)) &
147           $(vol_evidence_ratio.val/vol_evidence_ratio.err)\\\\\\ \\hline")

```

---

### 16.9.5 /cmz/a10periodisation.jl

```

1 using CSV,DataFrames,Distributions,StatsBase,GMC_NS,Serialization,
2   BioSimpleStochastic, Random, KernelDensityEstimate,
3   KernelDensityEstimatePlotting, Plots, Gadfly
4
5 gr()
6
7 a10pth="/bench/PhD/datasets/A10 measurements 2018update.csv"
8 e2ph="/bench/PhD/NGS_binaries/BSS/A10/e2ph"
9 e3ph="/bench/PhD/NGS_binaries/BSS/A10/e3ph"
10
11 paths=[e3ph]
12
13 a10df=DataFrame(CSV.read(a10pth))
14 a10df.BSR=[string(row.Block,',',row.Slide,',',row.Row) for row in
15   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 &&
32             → push!(ldvec,mean(skipmissing(i_df."Lens diameter")))
33     end
34
35     for i_df in groupby(t_df,"BSR")
36         if length([skipmissing(i_df."CMZ Sum") ... ])>0 &&
37             → mean(skipmissing(i_df."CMZ Sum")) > 0
38             push!(tvec,mean(skipmissing(i_df."CMZ Sum")))
39             if length([skipmissing(i_df."Lens diameter") ... ])>0
40                 push!(pevec,mean(skipmissing(i_df."CMZ
41                     → Sum"))*mean(skipmissing(i_df."Lens diameter"))/14.)
42             else
43                 push!(pevec,mean(skipmissing(i_df."CMZ
44                     → Sum"))*mean(ldvec)/14.)
45             end
46     end
47
48     if length([skipmissing(i_df."Retina thickness") ... ])>0 &&
49         length([skipmissing(i_df."IPL") ... ])>0 &&
50         length([skipmissing(i_df."OPL") ... ])>0 &&
51         length([skipmissing(i_df."RPE length") ... ])>0
52
53         rthi=mean(skipmissing(i_df."Retina
54             → thickness"))-mean(skipmissing(i_df."OPL"))-mean(skipmissing(i_df."IPL"))
55         push!(thivec,rthi)
56         rpel=mean(skipmissing(i_df."RPE length"))
57         if length([skipmissing(i_df."Lens diameter") ... ])>0
58             rcirc=rpel+mean(skipmissing(i_df."Lens diameter"))
59         end
60     end
61 end

```

```

53         else
54             rcirc=rpel+mean(ldvec)
55         end
56
57         push!(cvec,rcirc)
58
59         or=rcirc/2π
60         ir=or-.5*rthi
61
62         ov=(4/3)*π*(or^3)
63         iv=(4/3)*π*(ir^3)
64
65         push!(spvec,(ov-iv)*(4/5))
66     end
67 end
68
69     push!(measure_dict["PopEst"],pevec)
70     push!(measure_dict["SphEst"],spvec)
71     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=[ph3_constants]
91
92 cycle_prior=[Uniform(10.,144.), LogNormal(log(.9),log(2))]
93 end_prior=[Uniform(4.,359.)]
94 phase_max=[144.,5.]

```

```

95 phase_min=[10.,0.]
96 time_max=359
97 time_min=4.

98
99 function compose_priors(phases)
100     prs=Vector{Vector{Distribution}}(){}
101     bxes=Vector{Matrix{Float64}}(){}
102     for p in phases
103         pr=[repeat(cycle_prior,p) ... ,repeat(end_prior,p-1) ... ]
104         push!(prs,pr)
105         bx=hcat([repeat(phase_min,p) ... ,fill(time_min,p-1) ... ],
106                 [repeat(phase_max,p) ... ,fill(time_max,p-1) ... ])
107         push!(bxes,GMC_NS.to_unit_ball.(bx,pr))
108     end
109     return prs,bxes
110 end

111
112 phases=[3]
113 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 for (pth,prior,constants,box) in zip(paths,priors,constants,boxes)
119     if isfile(pth*"/ens")
120         e=deserialize(pth*"/ens")
121     else
122         e=CMZ_Ensemble(pth,3000,obs, prior, constants, box, GMC_DEFAULTS)
123     end
124
125     converge_ensemble!(e,backup=(true,50),upper_displays=uds,
126                         lower_displays=lds, disp_rot_its=100, mc_noise=.3,
127                         converge_criterion="compression", converge_factor=1.)
128 end

129 e2=deserialize(e2ph*"/ens")
130 e3=deserialize(e3ph*"/ens")

131 e2ev=measure_evidence(e2)
132 e3ev=measure_evidence(e3)

133
134 evidence_ratio=e2ev-e3ev

135

```

```

136 println("\begin{tabular}{l|l|l|l}")
137 println("\hline")
138 println("{\bf 2-phase logZ} & {\bf 3-phase logZ} & {\bf logZR} & {\bf
    \sigma Significance}\hline")
139 println("\textbf{\$(round(e2ev,digits=3))} & \$(round(e3ev,digits=3)) &
    \$(round(evidence_ratio,digits=3)) &
    \$(round(evidence_ratio.val/evidence_ratio.err,digits=3))\\\\
    \hline")
140 println("\end{tabular}")

141
142 map2=deserialize(e2.models[findmax([m.log_Li for m in
    e2.models])[2]].path)
143 map3=deserialize(e3.models[findmax([m.log_Li for m in
    e3.models])[2]].path)

144
145 println("\begin{tabular}{l|l|l}")
146 println("\hline")
147 println("{\bf Parameter} & {\bf 2-phase MAP} & {\bf 3-phase MAP}\
    \hline")
148 println("Phase 1 \$CT\$ (h) & \$(round(map2.θ[1], digits=1)) &
    \$(round(map3.θ[1], digits=1))\\\hline")
149 println("Phase 1 \$\epsilon\$ & \$(round(map2.θ[2], digits=2)) &
    \$(round(map3.θ[2], digits=2))\\\hline")
150 println("Phase 2 \$CT\$ (h) & \$(round(map2.θ[3], digits=1)) &
    \$(round(map3.θ[3], digits=1))\\\hline")
151 println("Phase 2 \$\epsilon\$ & \$(round(map2.θ[4], digits=2)) &
    \$(round(map3.θ[4], digits=2))\\\hline")
152 println("Phase 3 \$CT\$ (h) & NA & \$(round(map3.θ[5], digits=1))\\\\
    \hline")
153 println("Phase 3 \$\epsilon\$ & NA & \$(round(map3.θ[6], digits=2))\\\\
    \hline")
154 println("Transition 1 age & \$(round(map2.θ[5], digits=1)) &
    \$(round(map3.θ[7], digits=1))\\\hline")
155 println("Transition 2 age & NA & \$(round(map3.θ[8], digits=1))\\\\
    \hline")
156 println("\end{tabular}")

157
158 X=e2.constants[1]
159 plotticks=[30*i for i in 1:12]

160
161 catpobs=vcat([e2.obs[t][1] for t in 1:length(X)]...)
162 catvobs=vcat([e2.obs[t][2] for t in 1:length(X)]...)
163 Xs=vcat([[X[n] for i in 1:length(e2.obs[n][1])] for n in 1:length(X)]...)

```

```

164
165 p2_mean=map2.disp_mat[:,2,1]
166 p2_upper=map2.disp_mat[:,1,1]--map2.disp_mat[:,2,1]
167 p2_lower=map2.disp_mat[:,2,1]--map2.disp_mat[:,3,1]
168
169 map2_popplt=scatter(Xs,catpobs, marker=:cross, color=:black,
    ↵ markersize=3, label="CMZ population estimate", ylabel="Population",
    ↵ showaxis=:y, xticks=plotticks, xformatter=_→"", legend=:top,
    ↵ background_color_legend=nothing)
170 plot!(map2_popplt, X, p2_mean, ribbon=(p2_lower,p2_upper),
    ↵ color=:darkmagenta, label="2-phase model population")
171 lens!([2, 32], [300, 5700], inset = (1, bbox(0.6, 0.0, 0.38, 0.7)))
172 annotate!([(30,1.25e4,Plots.text("A1",18))])
173
174 v2_mean=map2.disp_mat[:,2,2]
175 v2_upper=map2.disp_mat[:,1,2]--map2.disp_mat[:,2,2]
176 v2_lower=map2.disp_mat[:,2,2]--map2.disp_mat[:,3,2]
177
178 map2_volplt=scatter(Xs,catvobs, marker=:cross, color=:black,
    ↵ markersize=3, label="Retinal volume estimate", ylabel="Volume
    ↵ (μm^3)", xlabel="Age (dpf)", legend=:topleft, showaxis=:y,
    ↵ xticks=plotticks, xformatter=_→"", background_color_legend=nothing)
179 plot!(map2_volplt, X, v2_mean, ribbon=(v2_lower,v2_upper), color=:green,
    ↵ label="2-phase model volume")
180 lens!([2, 32], [2e6, 3e7], inset = (1, bbox(0.6, 0.0, 0.34, 0.5)))
181 annotate!([(30,3.5e8,Plots.text("A2",18))])
182
183
184 p3_mean=map3.disp_mat[:,2,1]
185 p3_upper=map3.disp_mat[:,1,1]--map3.disp_mat[:,2,1]
186 p3_lower=map3.disp_mat[:,2,1]--map3.disp_mat[:,3,1]
187
188 map3_popplt=scatter(Xs,catpobs, marker=:cross, color=:black,
    ↵ markersize=3, label="CMZ population estimate", ylabel="Population",
    ↵ showaxis=:y, xticks=plotticks, xformatter=_→"", legend=:top,
    ↵ background_color_legend=nothing)
189 plot!(map3_popplt, X, p3_mean, ribbon=(p3_lower,p3_upper),
    ↵ color=:darkmagenta, label="3-phase model population")
190 lens!([2, 32], [300, 5700], inset = (1, bbox(0.6, 0.0, 0.38, 0.7)))
191 annotate!([(30,1.5e4,Plots.text("B1",18))])
192
193 v3_mean=map3.disp_mat[:,2,2]
194 v3_upper=map3.disp_mat[:,1,2]--map3.disp_mat[:,2,2]

```

```

195 v3_lower=map3.disp_mat[:,2,2]-map3.disp_mat[:,3,2]
196
197 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)
198 plot!(map3_volplt, X, v3_mean, ribbon=(v3_lower,v3_upper), color=:green,
    ↵ label="3-phase model volume")
199 lens!([2, 32], [2e6, 3e7], inset = (1, bbox(0.6, 0.0, 0.34, 0.5)))
200 annotate!([(30,3.5e8,Plots.text("B2",18))])
201
202 combined_map=Plots.plot(map2_popplt,map2_volplt,map3_popplt,map3_volplt,layout=grid(4
    ↵ size=(1200,1200), link=:x)
203
204 savefig(combined_map,"/bench/PhD/Thesis/images/cmz/a10pMAP.png")
205
206 kde2=posterior_kde(e2)
207 # kde3=posterior_kde(e3)
208
209 ph1marg=marginal(kde2,[1;2])
210 ph2marg=marginal(kde2,[3;4])
211 ph12marg=marginal(kde2,[1;3])
212 transmarg=marginal(kde2,[5])
213
214 ph1mplt=KernelDensityEstimatePlotting.plot(ph1marg;dimLbls=[ "Phase 1 CT
    ↵ (hr)", "Phase 1 e rate"], axis=[0. 144.; 0. 4.])
215 ph2mplt=KernelDensityEstimatePlotting.plot(ph2marg;dimLbls=[ "Phase 2 CT
    ↵ (hr)", "Phase 2 e rate"], axis=[0. 144.; 0. 4.])
216 ph12mplt=KernelDensityEstimatePlotting.plot(ph12marg; dimLbls=[ "Phase 1
    ↵ CT (hr)", "Phase 2 CT (hr)"],axis=[0. 144.; 0. 144.])
217 tmplt=KernelDensityEstimatePlotting.plotKDE(transmarg,xlbl="Phase
    ↵ transition age (dpf)", points=false, c=[ "green"])
218
219 combined_marg=Gadfly.gridstack([ph1mplt ph12mplt; ph2mplt tmplt])
220 img=SVG("/bench/PhD/Thesis/images/cmz/a10pmarginals.svg",24cm,24cm)
221 draw(img,combined_marg)

```

---

### 16.9.6 /cmz/a10popsurvey.jl

---

```

1 using CSV,DataFrames,Distributions,NGRefTools,StatsBase,Plots
2 gr()
3 default(fontsize = 10, guidefont = 11, tickfont = 10)

```

```

4
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["CMZ Sum"]=Vector{Vector{Float64}}(){}
16 measure_dict["Dorsal CMZ (#)"]=Vector{Vector{Float64}}(){}
17 measure_dict["Ventral CMZ (#)"]=Vector{Vector{Float64}}(){}
18 measure_dict["VolEst"]=Vector{Vector{Float64}}(){}
19 measure_dict["Retina thickness"]=Vector{Vector{Float64}}(){}
20 measure_dict["RPE length"]=Vector{Vector{Float64}}(){}
21 measure_dict["Lens diameter"]=Vector{Vector{Float64}}(){}
22 measure_dict["ON diameter"]=Vector{Vector{Float64}}(){}
23 measure_dict["ONL"]=Vector{Vector{Float64}}(){}
24 measure_dict["OPL"]=Vector{Vector{Float64}}(){}
25 measure_dict["INL"]=Vector{Vector{Float64}}(){}
26 measure_dict["IPL"]=Vector{Vector{Float64}}(){}
27 measure_dict["GCL"]=Vector{Vector{Float64}}(){}
28 measure_dict["Pop/VolEst"]=Vector{Vector{Float64}}(){}
29
30 #EXTRACT ALL MEASUREMENTS FROM DF TO VECTORS
31 for t_df in groupby(a10df, "Time point (d)")
32     push!(X,Float64(unique(t_df."Time point (d)")[1]))
33     pevec,tvec,dvec,vvec,vevec, rtvec, rplvec, rptvec, ldvec, ondvec,
→     onlvec, oplvec, inlvec, iplvec, gclvec, pvest = [zeros(0) for i
→     in 1:16]
34
35     for i_df in groupby(t_df,"BSR")
36         length([skipmissing(i_df."Lens diameter") ... ])>0 &&
→         push!(ldvec,mean(skipmissing(i_df."Lens diameter")))
37     end
38
39     for i_df in groupby(t_df,"BSR")
40         ipop=false
41         ivol=false
42         if length([skipmissing(i_df."CMZ Sum") ... ])>0

```

```

43     push!(tvec,mean(skipmissing(i_df."CMZ Sum")))
44     if length([skipmissing(i_df."Lens diameter") ... ])>0
45         push!(pevec,mean(skipmissing(i_df."CMZ
46             ↳ Sum"))*mean(skipmissing(i_df."Lens diameter"))/14.)
47         ipop=true
48     else
49         push!(pevec,mean(skipmissing(i_df."CMZ
50             ↳ Sum"))*mean(ldvec)/14.)
51         ipop=true
52     end
53 end
54
55 length([skipmissing(i_df."Dorsal CMZ (#)") ... ])>0 &&
56     ↳ push!(dvec,mean(skipmissing(i_df."Dorsal CMZ (#)")))
57 length([skipmissing(i_df."Ventral CMZ (#)") ... ])>0 &&
58     ↳ push!(vvec,mean(skipmissing(i_df."Ventral CMZ (#)")))
59 if length([skipmissing(i_df."Retina thickness") ... ])>0 &&
60     length([skipmissing(i_df."IPL") ... ])>0 &&
61     length([skipmissing(i_df."OPL") ... ])>0 &&
62     length([skipmissing(i_df."RPE length") ... ])>0
63
64     rthi=mean(skipmissing(i_df."Retina
65         ↳ thickness"))-mean(skipmissing(i_df."OPL"))-mean(skipmissing(i_df."IPL"))
66     rpel=mean(skipmissing(i_df."RPE length"))
67     if length([skipmissing(i_df."Lens diameter") ... ])>0
68         rcirc=rpel+mean(skipmissing(i_df."Lens diameter"))
69     else
70         rcirc=rpel+mean(ldvec)
71     end
72
73     or=rcirc/2π
74     ir=or-.5*rthi
75
76     ov=(4/3)*π*(or^3)
77     iv=(4/3)*π*(ir^3)
78
79     push!(vevec,(ov-iv)*(4/5))
80 end
81 length([skipmissing(i_df."Retina thickness") ... ])>0 &&
82     ↳ push!(rtvec,mean(skipmissing(i_df."Retina thickness")))
83 length([skipmissing(i_df."RPE length") ... ])>0 &&
84     ↳ push!(rptvec,mean(skipmissing(i_df."RPE length")))

```

```

78     length([skipmissing(i_df."ON diameter") ... ])>0 &&
    ↵   push!(ondvec,mean(skipmissing(i_df."ON diameter")))
79     length([skipmissing(i_df.ONL) ... ])>0 &&
    ↵   push!(onlvec,mean(skipmissing(i_df.ONL)))
80     length([skipmissing(i_df.OPL) ... ])>0 &&
    ↵   push!(oplvec,mean(skipmissing(i_df.OPL)))
81     length([skipmissing(i_df.INL) ... ])>0 &&
    ↵   push!(inlvec,mean(skipmissing(i_df.INL)))
82     length([skipmissing(i_df.IPL) ... ])>0 &&
    ↵   push!(iplvec,mean(skipmissing(i_df.IPL)))
83     length([skipmissing(i_df.GCL) ... ])>0 &&
    ↵   push!(gclvec,mean(skipmissing(i_df.GCL)))
84     ivol && ipop &&
85     push!(pvest,
86       mean(skipmissing(i_df."CMZ Sum"))/
87         ((mean(skipmissing(i_df."Retina
88           ↵   thickness"))-mean(skipmissing(i_df."OPL"))-mean(skipmissing(i_df."IPL
89             ↵   *mean(skipmissing(i_df."RPE length"))*(π/4)))
90   end
91
92   push!(measure_dict["PopEst"],pevec)
93   push!(measure_dict["CMZ Sum"],tvec)
94   push!(measure_dict["Dorsal CMZ (#)"],dvec)
95   push!(measure_dict["Ventral CMZ (#)"],vvec)
96   push!(measure_dict["VolEst"],vevec)
97   push!(measure_dict["Retina thickness"],rtvec)
98   push!(measure_dict["RPE length"],rptvec)
99   push!(measure_dict["Lens diameter"],ldvec)
100  push!(measure_dict["ON diameter"],ondvec)
101  push!(measure_dict["ONL"],onlvec)
102  push!(measure_dict["OPL"],oplvec)
103  push!(measure_dict["INL"],inlvec)
104  push!(measure_dict["IPL"],iplvec)
105  push!(measure_dict["GCL"],gclvec)
106  push!(measure_dict["Pop/VolEst"],pvest)
107
108 #TEST IF DATA IS BETTER MODELLED NORMALLY OR LOGNORMALLY
109 for ms in ["CMZ Sum", "Lens diameter", "PopEst", "RPE length", "Retina
110   ↵   thickness", "VolEst"]
111   normal_mts=[fit(Normal,measure_dict[ms][n]) for n in 1:length(X)]
112   logn_mts=[fit(LogNormal,filter!(i→!iszero(i),measure_dict[ms][n]))
113   ↵   for n in 1:length(X)]

```

```

112
113     normal_lh=sum([sum(logpdf(normal_mts[n],measure_dict[ms][n])) for n
114     ↪   in 1:length(X)])
114     logn_lh=sum([sum(logpdf(logn_mts[n],measure_dict[ms][n])) for n in
115     ↪   1:length(X)])
116
116     println("$ms & $(round(normal_lh,digits=3)) &
117     ↪   $(round(logn_lh,digits=3)) &
118     ↪   $(round(logn_lh-normal_lh,digits=3))\\\hline")
119
119 end
120
120 pe_n_mts=[fit(MarginalTDist,measure_dict["PopEst"][n]) for n in
121     ↪   1:length(X)]
121 pen_lower=[mean(mt)-quantile(mt,.025) for mt in pe_n_mts]
122 pen_upper=[quantile(mt,.975)-mean(mt) for mt in pe_n_mts]
123
124 pe_logn_mts=[fit(MarginalTDist,log.(filter!(i→!iszero(i),measure_dict["PopEst"][n])))
125     ↪   for n in 1:length(X)]
125 pe_mean=[exp(mean(mt)) for mt in pe_logn_mts]
126 pe_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in pe_logn_mts]
127 pe_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in pe_logn_mts]
128
129 plotticks=[30*i for i in 1:12]
130
131 ann_chart=scatter(vcat([[X[n] for i in
132     ↪   1:length(measure_dict["PopEst"][n])] for n in
133     ↪   1:length(X)] ... ),vcat(measure_dict["PopEst"] ... ), marker=:cross,
134     ↪   color=:black, markersize=3, label="Pop. data", ylabel="Est. CMZ
135     ↪   annulus population", showaxis=:y, xticks=plotticks, xformatter=_→"")
132 plot!(ann_chart, X, pe_mean, ribbon=(pe_lower,pe_upper),
136     ↪   color=:darkmagenta, label="Pop. mean")
133 lens!([2, 20], [500, 2500], inset = (1, bbox(0.3, 0.1, 0.45, 0.45)))
134 annotate!([(8,4500,Plots.text("A",18))])
135
136 poprate_mean,poprate_lower,poprate_upper=[Vector{Float64}() for i in 1:3]
137 for (nx, x) in enumerate(X)
138     if nx > 1
139         d=X[nx]-X[nx-1]
140         l,m,u=MTDist_MC_func((a,b)→((exp(a)-exp(b))/d),
141         ↪   [log.(filter!(i→!iszero(i),measure_dict["PopEst"][nx])),,
142         ↪   log.(filter!(i→!iszero(i),measure_dict["PopEst"][nx-1])),,
143         ↪   summary=true)

```

```

141      ↳ push!(poprate_mean,m);push!(poprate_lower,m-l);push!(poprate_upper,u-m)
142    end
143  end
144
145 poprate_chart=plot(X[2:end],poprate_mean,ribbon=(poprate_lower,
146   ↳ poprate_upper), color=:grey, ylabel="Est. CMZ pop. rate of change
147   ↳ (cells/d)", label="Simulated mean rate", legend=:topright,
148   ↳ showaxis=:y, xticks=plotticks, xformatter=_→"")
149 annotate!([(8,400,Plots.text("B",18))])
150
151 l3,m3,u3=MTDist_MC_func((a,b)→(exp(a)-exp(b))/5.,
152   ↳ [filter!(:isinf,log.(measure_dict["PopEst"][[2]])),
153   ↳ filter!(:isinf,log.(measure_dict["PopEst"][[1]]))), summary=true)
154
155 ratedist23=MTDist_MC_func((a,b)→(exp(a)-exp(b))/6.,
156   ↳ [filter!(:isinf,log.(measure_dict["PopEst"][[6]])),
157   ↳ filter!(:isinf,log.(measure_dict["PopEst"][[5]]))), summary=false)
158
159 println("3dpf poprate mean: $(m3) 23dpf poprate mean
160   ↳ $(mean(ratedist23))poprate 23dpf quantile :
161   ↳ $(quantile(ratedist23,.025))")
162
163 ve_logn_mts=[fit(MarginalTDist,log.(filter!(i→!iszero(i),measure_dict["VolEst"][[n]]))
164   ↳ for n in 1:length(X)]
165 ve_mean=[exp(mean(mt)) for mt in ve_logn_mts]
166 ve_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in ve_logn_mts]
167 ve_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in ve_logn_mts]
168
169 vol_chart=scatter(vcat([[X[n] for i in
170   ↳ 1:length(measure_dict["VolEst"][[n]])] for n in
171   ↳ 1:length(X)] ... ),vcat(measure_dict["VolEst"] ... ), marker=:cross,
172   ↳ color=:black, markersize=3, label="Vol. data", ylabel="Est. retinal
173   ↳ volume (μm³)", showaxis=:y, xticks=plotticks, xformatter=_→"",
174   ↳ legend=:bottomright)
175 plot!(vol_chart, X, ve_mean, ribbon=(ve_lower,ve_upper), color=:blue,
176   ↳ label="Vol. mean")
177 lens!([2, 20], [2.5e6, 1e7], inset = (1, bbox(0.12, 0., 0.39, 0.40)))
178 annotate!([(8,4.5e8,Plots.text("C",18))])
179
180
181
182
183
184

```

```

165 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")
166
167 volrate_mean,volrate_lower,volrate_upper=[Vector{Float64}() for i in 1:3]
168 for (nx, x) in enumerate(X)
169     if nx > 1
170         d=X[nx]-X[nx-1]
171         l,m,u=MTDist_MC_func((a,b)→(exp(a)-exp(b))/d,
172             ↳ [filter{!isinf,log.(measure_dict["VolEst"])[nx]}),
173             ↳ filter{!isinf,log.(measure_dict["VolEst"])[nx-1]}),
174             ↳ summary=true)
175         push!(volrate_mean,m);push!(volrate_lower,l);push!(volrate_upper,u)
176     end
177 end
178 volrate_lower=volrate_mean-volrate_lower
179 volrate_upper=volrate_upper-volrate_mean
180
181 l=@layout [a{0.3h}
182             ↳ b{0.2h}
183             ↳ c{0.3h}
184             ↳ d]
185 combined=plot(ann_chart, poprate_chart, vol_chart, volrate_chart, layout=(4,1), size=(1200
186 savefig(combined, "/bench/PhD/Thesis/images/cmz/CMZoverall.png")
187
188 l30,m30,u30=MTDist_MC_func((a,b)→(exp(a)-exp(b))/7.,
189     ↳ [filter{!isinf,log.(measure_dict["VolEst"])[7]}),
190     ↳ filter{!isinf,log.(measure_dict["VolEst"])[6]}), summary=true)
191
192 ratedist60=MTDist_MC_func((a,b)→(exp(a)-exp(b))/30.,
193     ↳ [filter{!isinf,log.(measure_dict["VolEst"])[8]}),
194     ↳ filter{!isinf,log.(measure_dict["VolEst"])[7]}), summary=false)
195
196 println("6*30dpf volrate mean: $(6*m30) volrate 60dpf quantile :
197     ↳ $(quantile(ratedist60,.01))")
```

---

### 16.9.7 /cmz/a19lineagetrace.jl

---

```

1 using CSV,DataFrames,Distributions,NGRefTools,StatsBase,Plots,
2   ↵ Measurements,GMC_NS,Serialization
3 gr()
4
5 a19_1pth="/bench/PhD/datasets/A19GR1.csv"
6 a19_2pth="/bench/PhD/datasets/A19GR2.csv"
7 a19_3pth="/bench/PhD/datasets/A19GR3.csv"
8
9 gr1df=DataFrame(CSV.read(a19_1pth))
10 gr2df=DataFrame(CSV.read(a19_2pth))
11 gr3df=DataFrame(CSV.read(a19_3pth))
12
13 X=sort(unique(gr1df."Pulse age (dpf)"))
14 t_measure_dict=Dict{String,Vector{Vector{Float64}}}()
15 d_measure_dict=Dict{String,Vector{Vector{Float64}}}()
16 v_measure_dict=Dict{String,Vector{Vector{Float64}}}()
17
18 mds=[t_measure_dict,d_measure_dict,v_measure_dict]
19
20 for md in mds
21   md["ONL"]=[zeros(0) for i in 1:length(X)]
22   md["INL"]=[zeros(0) for i in 1:length(X)]
23   md["GCL"]=[zeros(0) for i in 1:length(X)]
24   md["Isl"]=[zeros(0) for i in 1:length(X)]
25   md["Pax6"]=[zeros(0) for i in 1:length(X)]
26   md["Isl_Pax6"]=[zeros(0) for i in 1:length(X)]
27   md["INL_Pax6"]=[zeros(0) for i in 1:length(X)]
28   md["PKCB"]=[zeros(0) for i in 1:length(X)]
29   md["GS"]=[zeros(0) for i in 1:length(X)]
30   md["HM"]=[zeros(0) for i in 1:length(X)]
31   md["Zpr1"]=[zeros(0) for i in 1:length(X)]
32 end
33
34 function check_layerfrac(row)
35   !ismissing(row.#GCL) && !ismissing(row.#INL) &&
36   ↵ !ismissing(row.#PRL) && !ismissing(row.Total) ? (return true)
37   ↵ : (return false)
38 end
39
40 function gr1_dfcheck(subdf)
41   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 for df in [gr1df,gr2df,gr3df]
47     for agedf in groupby(df, "Pulse age (dpf)")
48         xidx=findfirst(i→i==unique(agedf."Pulse age (dpf)")[1],X)
49         for inddf in groupby(agedf, "Individual")
50             if df == gr1df
51                 if gr1_dfcheck(inddf)
52                     clean=false(2)
53                     for row in eachrow(inddf)
54                         if row."D/V" == "D"
55                             push!(d_measure_dict["Isl"][xidx],row."GCL -
56                                 Isl"/row.#GCL")
57                             push!(d_measure_dict["Pax6"][xidx],row."GCL -
58                                 Pax6"/row.#GCL")
59
60                             push!(d_measure_dict["Isl_Pax6"][xidx],row."GCL -
61                                 - Isl/Pax6"/row.#GCL")
62
63                             push!(d_measure_dict["INL_Pax6"][xidx],row."INL -
64                                 - Pax6"/row.#INL")
65                         if check_layerfrac(row)
66
67                             push!(d_measure_dict["GCL"][xidx],row.#GCL/row.
68
69                             push!(d_measure_dict["INL"][xidx],row.#INL/row.
70
71                             push!(d_measure_dict["ONL"][xidx],row.#PRL/row.
72                             clean[1]=true
73
74                         end
75                     else
76                         push!(v_measure_dict["Isl"][xidx],row."GCL -
77                             Isl"/row.#GCL")
78                         push!(v_measure_dict["Pax6"][xidx],row."GCL -
79                             Pax6"/row.#GCL")

```

```

67          ↵ push!(v_measure_dict["Isl_Pax6"][xidx],row."GCL
68          ↵ - Isl/Pax6"/row."#GCL")
69      if check_layerfrac(row)
70          ↵ push!(v_measure_dict["GCL"][xidx],row."#GCL"/row.
71          ↵ push!(v_measure_dict["INL"][xidx],row."#INL"/row.
72          ↵ push!(v_measure_dict["ONL"][xidx],row."#PRL"/row.
73          ↵ clean[2]=true
74      end
75  end
76
77  push!(t_measure_dict["Isl"][xidx],sum(inddf."GCL -
78  ↵ - Isl")/sum(inddf."#GCL"))
79  push!(t_measure_dict["Pax6"][xidx],sum(inddf."GCL -
80  ↵ - Pax6")/sum(inddf."#GCL"))
81  push!(t_measure_dict["Isl_Pax6"][xidx],sum(inddf."GCL
82  ↵ - Isl/Pax6")/sum(inddf."#GCL"))
83  push!(t_measure_dict["INL_Pax6"][xidx],sum(inddf."INL
84  ↵ - Pax6")/sum(inddf."#INL"))
85  if all(clean)
86      ↵ push!(t_measure_dict["GCL"][xidx],sum(inddf."#GCL")/sum(i
87      ↵ push!(t_measure_dict["INL"][xidx],sum(inddf."#INL")/sum(i
88      ↵ push!(t_measure_dict["ONL"][xidx],sum(inddf."#PRL")/sum(i
89  end
90
91  elseif df === gr2df
92      for row in eachrow(inddf)
93          if row."D/V" == "D"
94              ↵ push!(d_measure_dict["PKCB"][xidx],row."INL-PKCB"/row."#I
95              ↵ push!(d_measure_dict["GS"][xidx],row."INL-GS"/row."#INL")
96              ↵ push!(d_measure_dict["HM"][xidx],row."INL-HM"/row."#INL")

```

```

93          ↳ push!(d_measure_dict["GCL"][xidx],row. "#GCL"/row. "Total")
94          ↳ push!(d_measure_dict["INL"][xidx],row. "#INL"/row. "Total")
95          ↳ push!(d_measure_dict["ONL"][xidx],row. "#PRL"/row. "Total")
96    else
97          ↳ push!(v_measure_dict["PKCB"][xidx],row. "INL-PKCB"/row. "#INL")
98          ↳ push!(v_measure_dict["GS"][xidx],row. "INL-GS"/row. "#INL")
99          ↳ push!(v_measure_dict["HM"][xidx],row. "INL-HM"/row. "#INL")
100         ↳ push!(v_measure_dict["GCL"][xidx],row. "#GCL"/row. "Total")
101         ↳ push!(v_measure_dict["INL"][xidx],row. "#INL"/row. "Total")
102         ↳ push!(v_measure_dict["ONL"][xidx],row. "#PRL"/row. "Total")
103    end
104  end
105
106          ↳ push!(t_measure_dict["PKCB"][xidx],sum(inddf. "INL-PKCB")/sum(inddf.
107          ↳ push!(t_measure_dict["GS"][xidx],sum(inddf. "INL-GS")/sum(inddf. "#INL")
108          ↳ push!(t_measure_dict["HM"][xidx],sum(inddf. "INL-HM")/sum(inddf. "#INL")
109          ↳ push!(t_measure_dict["GCL"][xidx],sum(inddf. "#GCL")/sum(inddf. "Total"))
110          ↳ push!(t_measure_dict["INL"][xidx],sum(inddf. "#INL")/sum(inddf. "Total"))
111    elseif df === gr3df
112      for row in eachrow(inddf)
113        if row. "D/V" == "D"
114          push!(d_measure_dict["Zpr1"][xidx],row. "PRL" -
115          ↳ "Zpr1"/row. "#PRL")
116          ↳ push!(d_measure_dict["GCL"][xidx],row. "#GCL"/row. "Total")
117          ↳ 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_n_mts=[fit(MarginalTDist,t_measure_dict["GCL"][n]) for n in
  ↪ 1:length(X)]
156 gcl_mean=[mean(mt) for mt in gcl_n_mts]
157 gcl_lower=[mean(mt)-quantile(mt,.025) for mt in gcl_n_mts]
158 gcl_upper=[quantile(mt,.975)-mean(mt) for mt in gcl_n_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_n_mts=[fit(MarginalTDist,t_measure_dict["Isl"][n]) for n in
  ↪ 1:length(X)]
169 isl_mean=[mean(mt) for mt in isl_n_mts]
170 isl_lower=[mean(mt)-quantile(mt,.025) for mt in isl_n_mts]
171 isl_upper=[quantile(mt,.975)-mean(mt) for mt in isl_n_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")
175 annotate!([(15,.05,Plots.text("B",18))])

176
177 pax6_ln_mts=[fit(MarginalTDist,log.(t_measure_dict["Pax6"][n])) for n in
  ↵ 1:length(X)]
178 pax6_mean=[exp(mean(mt)) for mt in pax6_ln_mts]
179 pax6_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in pax6_ln_mts]
180 pax6_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in pax6_ln_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(λ)",18))])

185
186 islp_n_mts=[fit(MarginalTDist,t_measure_dict["Isl_Pax6"][n]) for n in
  ↵ 1:length(X)]
187 islp_mean=[mean(mt) for mt in islp_n_mts]
188 islp_lower=[mean(mt)-quantile(mt,.025) for mt in islp_n_mts]
189 islp_upper=[quantile(mt,.975)-mean(mt) for mt in islp_n_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_n_mts=[fit(MarginalTDist,t_measure_dict["INL_Pax6"][n]) for n in
  ↵ 1:length(X)]
198 pax6i_mean=[mean(mt) for mt in pax6i_n_mts]
199 pax6i_lower=[mean(mt)-quantile(mt,.025) for mt in pax6i_n_mts]
200 pax6i_upper=[quantile(mt,.975)-mean(mt) for mt in pax6i_n_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(λt)",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(λ)",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(λ)",18))])
232
233 inl_subplot=plot(inlpax6chart, pkcchart, gschart, hmchart,
  ↵   layout=grid(4,1), link=:x)
234
235 z_ln_mts=[fit(MarginalTDist,log.(t_measure_dict["Zpr1"])[n])) for n in
  ↵  1:length(X)]
236 z_mean=[exp(mean(mt)) for mt in z_ln_mts]
237 z_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in z_ln_mts]
238 z_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in z_ln_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(λ+)",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_μ=1.
270 min_μ=0.
271 max_λ=4000.
272 min_λ=1.
273
274 prior=[Uniform(min_μ,max_μ),Uniform(min_λ,max_λ)]
275 box=[min_μ max_μ;min_λ max_λ]

```

```

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/BSS/A19/n_}*ms
280         x=X[n]
281         enspth=e_basepth*"/$x"
282         if isfile(enspth*"/ens")
283             e=deserialize(enspth*"/ens")
284         else
285             e=Normal_Engsemble(enspth,n_models,filter(i→i>0,ovec), prior,
286             → box, gmc ... )
287         end
288
289         "n_}*ms in keys(evdict) ? (evdict["n_}*ms] +=
290             → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
291             → lower_displays=lds, disp_rot_its=10000,
292             → converge_criterion="compression", converge_factor=1e-6)) :
293             (evdict["n_}*ms] =
294             → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
295             → lower_displays=lds, disp_rot_its=10000,
296             → converge_criterion="compression", converge_factor=1e-6))
297         end
298
299         for (n,ovec) in enumerate(t_measure_dict[ms])
300             e_basepth="/bench/PhD/NGS_binaries/BSS/A19/ln_}*ms
301             x=X[n]
302             enspth=e_basepth*"/$x"
303             if isfile(enspth*"/ens")
304                 e=deserialize(enspth*"/ens")
305             else
306                 e=LogNormal_Engsemble(enspth,n_models,filter(i→i>0,ovec),
307                 → prior, box, gmc ... )
308             end
309
310             "ln_}*ms in keys(evdict) ? (evdict["ln_}*ms] +=
311                 → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
312                 → lower_displays=lds, disp_rot_its=10000,
313                 → converge_criterion="compression", converge_factor=1e-6)) :
314                 (evdict["ln_}*ms] =
315                 → converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
316                 → lower_displays=lds, disp_rot_its=10000,
317                 → converge_criterion="compression", converge_factor=1e-6))
318             end

```

```

303 end

304

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")
312 end

313 for ms in keys(t_measure_dict)
314     if ms in ["INL_Pax6","INL","Isl","Isl_Pax6","ONL","Zpr1"]
315         efunc=Normal_Ensemble
316         pf="cn_"
317     else
318         efunc=LogNormal_Ensemble
319         pf="cln_"
320     end

321 ovec=vcat(t_measure_dict[ms]...)
322 e_basepth="/bench/PhD/NGS_binaries/BSS/A19/*pf*ms

323 if isfile(e_basepth*/ens")
324     e=deserialize(e_basepth*/ens")
325 else
326     e=efunc(e_basepth,n_models,filter(i→i>0,ovec), prior, box,
327             & gmc...)
328 end

329 evdict[pf*ms] =
330     converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
331     & lower_displays=lds, disp_rot_its=10000,
332     & converge_criterion="compression", converge_factor=1e-6)
333 end

334 for ms in keys(t_measure_dict)
335     if ms in ["INL_Pax6","INL","Isl","Isl_Pax6","ONL","Zpr1"]
336         pf="n_"
337     else
338         pf="ln_"
339     end

```

```

340     cev=evdict["c"]*pf*ms]
341     sev=evdict[pf*ms]
342     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/BSS/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,
372         ↪ converge_criterion="compression", converge_factor=1e-6)
373 end
374
375 for ms in keys(v_measure_dict)
376     if ms in ["INL_Pax6","INL","Isl","Isl_Pax6","ONL","Zpr1"]
377         efunc=Normal_Ensemble
378         pf="v_cn_"
379     else
380         efunc=LogNormal_Ensemble
381         pf="v_cln_"
382     end
383
384     dv_evdict[pf*ms] =
385         ↪ converge_ensemble!(e,backup=(true,10000),upper_displays=uds,
386         ↪ lower_displays=lds, disp_rot_its=10000,
387         ↪ converge_criterion="compression", converge_factor=1e-6)
388 end

```

```

377
378     ovec=vcat(d_measure_dict[ms] ... )
379     e_basepth="/bench/PhD/NGS_binaries/BSS/A19/"*pf*ms
380
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,
391         → converge_criterion="compression", converge_factor=1e-6)
392 end
393
394
395 for ms in keys(t_measure_dict)
396     if ms in ["INL_Pax6","INL","Isl","Isl_Pax6","ONL","Zpr1"]
397         pf="n_"
398     else
399         pf="ln_"
400     end
401
402     cev=evdict["c"*pf*ms]
403     dev=dv_evdict["d_c"*pf*ms]
404     vev=dv_evdict["v_c"*pf*ms]
405     sev=dev+vev
406     ratio=cev-sev
407     println("$ms & $(round(cev,digits=3)) & $(round(sev,digits=3)) &
408         → $(round(ratio,digits=3)) &
409         → $(round(ratio.val/ratio.err,digits=1))\\\\ \\hline")
410 end

```

---

### 16.9.8 /cmz/a25GMC\_NS.jl

---

```

1 using GMC_NS, BioSimpleStochastic, ConjugatePriors, CSV, DataFrames,
    ↪ Distributed, Distributions, NGRefTools, ProgressMeter, Serialization
2
3 a10path="/bench/PhD/datasets/A10 measurements 2018update.csv"
4 a10df=DataFrame(CSV.File(a10path))

```

```

5 df3=a10df[a10df."Time point (d)" .== 3, :]
7
8 inddict=Dict{Tuple,Vector}(){}
9 for row in eachrow(df3)
10    ind=(row.Block, row.Slide, row.Row)
11    if !(ind in keys(inddict))
12        inddict[ind]=[(row["Dorsal CMZ (#)"]+row["Ventral CMZ (#)"])]
13    else
14        push!(inddict[ind],((row["Dorsal CMZ (#)"]+row["Ventral CMZ
15           (#)"])))
16    end
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 sp_max_mean,sp_max_std=get_lognormal_desc(fit(LogNormal,popvec))
54 sp_max_var=sp_max_std^2
55
56 pop_mu_prior=Uniform(1.,sp_max_mean)
57 pop_std_prior=Uniform(1.,sp_max_var)
58 r_prior=Normal(5,1.5)
59 tc_prior=NormalInverseGamma(3.,1.,5.,2.)
60 s_prior=NormalInverseGamma(6.,2.5,1.,1.)
61 sister_prior=Beta(8.,75.)
62
63 priors_1_pop=[pop_mu_prior, pop_std_prior, r_prior,
    ↪ marginals(tc_prior) ... , marginals(s_prior) ... , sister_prior]
64 priors_2_pop=vcat(priors_1_pop,priors_1_pop)
65
66 p1_box=[1. sp_max_mean
67           1. sp_max_var
68           1. 10.
69           1. 4.5
70           eps() .5
71           eps() .5]
72
73 p2_box=vcat(p1_box,p1_box)
74
75 prior_sets=[priors_1_pop,priors_2_pop]
76 ensemble_paths=[ "/bench/PhD/NGS_binaries/BSS/A25/1pop",
    ↪ "/bench/PhD/NGS_binaries/BSS/A25/2pop"]
77 const pulse_time=10.5
78 const mc_its=Int64(5e5)
79 const end_time=10.5
80 const retain_run=false
81 constants=[X, pulse_time, mc_its, end_time, retain_run]
82
83
84 ensembles=Vector{Thymidine_Engsemble}()
85 for (ps, ep, box) in zip(prior_sets,ensemble_paths,boxes)
86     if isfile(ep*"/ens")
87         push!(ensembles,deserialize(ep*"/ens"))

```

```

88     else
89         @info "Assembling ensemble at $ep"
90         push!(ensembles,Thymidine_Ensemble(ep, 50, y, ps, constants, box,
91             → GMC_DEFAULTS))
92     end
93 end
94 #uds=Vector{Vector{Function}}([[tuning_display],[convergence_display],[tuning_display]
95 uds=Vector{Vector{Function}}([[tuning_display]])
96 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
97
98 evidence=zeros(length(ensembles))
99 for (ne,e) in enumerate(ensembles)
100     evidence[ne]=converge_ensemble!(e, τ₀=.03,
101         → backup=true,1),upper_displays=uds, lower_displays=lds,
102         → disp_rot_its=5, mc_noise=.139)
103 end

```

---

### 16.9.9 /cmz/a25dvl.inreg.jl

```

1 using Distributions, Serialization, CSV, BayesianLinearRegression, Plots,
2     → DataFrames, Measurements
2 import Measurements:value,uncertainty
3
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 function blr_plt(x,y,Xs,blrs,colors,labels,q=.975)

```

```

62     ↵ plt=scatter(x,y,marker=:cross,markersize=3,markercolor=:black,label="Data",yl
63     ↵ labelled fraction", xlabel="Pulse time (hr)", legend=:bottomright,
64     ↵ ylims=(-0.1,1.))
65     for (X,blr,color,label) in zip(Xs, blrs,colors,labels)
66         line=zeros(size(X,1))
67         ribbon=zeros(size(X,1))
68         predictions=BayesianLinearRegression.predict(blr,X)
69         for (n,p) in enumerate(predictions)
70             normal=Normal(value(p),uncertainty(p))
71             ribbon[n]=quantile(normal,q)-mean(normal)
72             line[n]=mean((value(p)))
73         end
74         plot!(plt,x,line,ribbon=ribbon,color=color,label=label)
75     end
76     return plt
77 end

78 dplt=blr=plt(xYD[:,2],YD,[xYD],[dm],[green],"Dorsal model")
79 annotate!([(1,.9,Plots.text("A",12))])
80 vplt=blr=plt(xYV[:,2],YV,[xYV],[vm],[magenta],"Ventral model")
81 tplt=blr=plt(xY[:,2],Y,[xY],[tm],[black],"Combined model")
82 annotate!([(1,.9,Plots.text("C",12))])

83 combined=plt(dplt,vplt,tplt,layout=(3,1), size=(600,900))

84 savefig(combined=plt, "/bench/PhD/Thesis/images/cmz/3ddvlinreg.png")
85 @info "Saved fig."
86
87
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}")

```

---

### 16.9.10 /cmz/a25thymidinesim.jl

```

1 using GMC_NS, BioSimpleStochastic, ConjugatePriors, CSV, DataFrames,
   ↵ Distributions, Serialization, NGRefTools
2
3 a10path="/bench/PhD/datasets/A10 measurements 2018update.csv"
4 a10df=DataFrame(CSV.File(a10path))
5
6 df3=a10df[a10df."Time point (d)" .== 3, :]
7
8 inddict=Dict{Tuple,Vector}(){}
9 for row in eachrow(df3)
10    ind=(row.Block, row.Slide, row.Row)
11    if !(ind in keys(inddict))
12       inddict[ind]=[(row["Dorsal CMZ (#)"]+row["Ventral CMZ (#)"])]
13    else
14       push!(inddict[ind],((row["Dorsal CMZ (#)"]+row["Ventral CMZ
          ↵ (#)"])))
15    end
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_prior=fit(NormalInverseGamma,log.(popvec))
54
55 r_prior=Normal(5,1.5)
56 tc_prior=NormalInverseGamma(3.,1.,5.,2.)
57 s_prior=NormalInverseGamma(6.,2.5,1.,1.)
58 sister_prior=Beta(8.,75.)
59
60 priors_1_pop=[marginals(pop_prior) ... , r_prior, marginals(tc_prior) ... ,
61   ↳ marginals(s_prior) ... , sister_prior]
62
63 p1_box=[3.3 5.8
64           .07 .26
65           1. 10.
66           .1 10.
67           eps() 2.7
68           4. 20.
69           .1 200.
70           eps() .5]
71
72 p1_box=GMC_NS.to_unit_ball.(p1_box,priors_1_pop)
73
74 ep="/bench/PhD/NGS_binaries/BSS/A25/1pop"
75 const pulse_time=10.5
76 const mc_its=Int64(5e5)
77 const end_time=10.5
78 constants=[X, pulse_time, mc_its, end_time]
79 if isfile(ep*"ens")

```

```

80     e=deserialize(ep*"/ens")
81 else
82     @info "Assembling ensemble at $ep"
83     gmcd=GMC_DEFAULTS
84     gmcd[1]=5
85     e=Thymidine_Ensemble(ep, 100, y, priors_1_pop, constants, p1_box,
86     ↳ gmcd)
86 end
87
88 uds=Vector{Vector{Function}}([[liwi_display],[convergence_display],[evidence_display]]
89 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display],[ensemble_displa
90
91 converge_ensemble!(e,backup=(true,1),upper_displays=uds,
92     ↳ lower_displays=lds, disp_rot_its=5, mc_noise=.14,
93     ↳ converge_factor=1e-3)

```

---

### 16.9.11 /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
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/BSS/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_criterion="compression", 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_criterion="compression", converge_factor=1e-6))
62         end
63     end
64 end
65
66 for (obs,pstring) in zip([cobs,mobs],["cCR","cmo1"])

```

```

60     enspth="/bench/PhD/NGS_binaries/BSS/A27/*pstring
61     if isfile(enspth*"/ens")
62         e=deserialize(enspth*"/ens")
63     else
64         e=LogNormal_Ensemble(enspth,n_models,obs, prior, box, gmc ... )
65     end
66
67     → evdict[pstring]=converge_ensemble!(e,backup=(true,1000),upper_displays=uds,
68     → lower_displays=lds, disp_rot_its=10000,
69     → converge_criterion="compression", converge_factor=1e-6)
70 end
71
72 for ms in ["CR", "mo1"]
73     cev=evdict["c"*ms]
74     sev=evdict[ms]
75     ratio=cev-sev
76     println("$ms & $(round(cev,digits=3)) & $(round(sev,digits=3)) &
77     → $(round(ratio,digits=3)) &
78     → $(round(ratio.val/ratio.err,digits=1))\\\\ \\hline")
79 end

```

---

### 16.9.12 /cmz/a27linreg.jl

```

1 using Distributions, Serialization, CSV, BayesianLinearRegression, Plots,
2   ↪ DataFrames, Measurements
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 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
51   → plt=scatter(x,y,marker=:diamond,markersize=8,markerstrokestyle=:bold,markerco
52   → size", xlabel="Age
53   → (dpf)", legend=:topright, ylims=[0,maximum(y)+25])
54   for (X,blr,color,label) in zip(Xs, blrs,colors,labels)
55     line=zeros(size(X,1))
56     ribbon=zeros(size(X,1))
57     predictions=BayesianLinearRegression.predict(blr,X)
58     for (n,p) in enumerate(predictions)
59       normal=Normal(value(p),uncertainty(p))
60     end
61   end
62   return plt
63 end

```

```

57         ribbon[n]=quantile(normal,q)-mean(normal)
58         line[n]=mean((value(p)))
59     end
60     plot!(plt,i,line,ribbon=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],
66   ↪ "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,
69   ↪ "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
77   ↪ logZ} & {\bf logZR}\hline")
77 println("1dpf Central Remnant & {\bf
78   ↪ $(round(logEvidence(uncorrCR2),digits=3))} &
78   ↪ $(round(logEvidence(corrCR2),digits=3)) &
78   ↪ $(round(logEvidence(uncorrCR2)-logEvidence(corrCR2),digits=3))\hline")
78 println("30dpf Cohort & {\bf $(round(logEvidence(uncorrmo1),digits=3))} &
78   ↪ & $(round(logEvidence(corrmo1),digits=3)) &
78   ↪ $(round(logEvidence(uncorrmo1)-logEvidence(corrmo1),digits=3))\hline")
79 println("\end{tabular}")

```

---

### 16.9.13 /rys/a37.jl

---

```

1 using CSV,DataFrames,Distributions,Measurements,GMC_NS, NGRefTools,
2   ↪ Plots,Measures, Serialization
2 gr()
3 default(fontsize = 10, guidefont = (12,:bold), tickfont = 10, guide =
4   ↪ "x")

```

```

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             ↪ dict["Sphericity"][]+=row."Sphericity";dict["Sphericity"][]/=2
40             dict["Volume"][]+=row."Volume";dict["Volume"][]/=2
41         end
42     end
43 end
44 for md in msd

```

```

45     md["NucPer"] = md["NucPop"] ./ md["TCMZ"]
46 end
47
48 gmc=GMC_DEFAULTS
49 gmc[1]=5
50 gmc[2]=1.e-15
51 gmcdir="/bench/PhD/NGS_binaries/BSS/A37"
52
53
54 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]]
55 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
56
57 n_models=50
58
59 max_mu=300.;min_mu=0.
60 max_lambda=1.;min_lambda=1e-5
61 prior=[Uniform(min_mu,max_mu),Uniform(min_lambda,max_lambda)]
62 box=[min_mu max_mu;min_lambda max_lambda]
63
64 np_prior=[Uniform(min_mu,5000),Uniform(min_lambda,max_lambda)]
65 np_box=[min_mu 5000;min_lambda max_lambda]
66
67 s_max_mu=1.;s_min_mu=0.
68 s_max_lambda=5e5;s_min_lambda=1e3
69 s_prior=[Uniform(s_min_mu,s_max_mu),Uniform(s_min_lambda,s_max_lambda)]
70 s_box=[s_min_mu s_max_mu;s_min_lambda s_max_lambda]
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"])
77     evdict=Dict{String,Measurement}()
78
79     for ms in totalmsvec
80         evdict[ms*"_Combined"] = measurement(0,0)
81         evdict[ms*"_Separate"] = measurement(0,0)
82     end
83
84     for ms in popmsvec
85         ms=="NucPop" ? (pr=np_prior; bx=np_box) : (pr=prior; bx=box)
86
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     ← evdict[ms*"_Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_di
98     ← lower_displays=lds, disp_rot_its=10000,
99     ← converge_criterion="compression", converge_factor=1e-6)
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122

```

```

123
124
125     ↵ evdict[ms*"_Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_di
126     ↵ lower_displays=lds, disp_rot_its=10000,
127     ↵ converge_criterion="compression", converge_factor=1e-6)
128
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
145         ↵ evdict[ms*"_Separate"]+=converge_ensemble!(e,backup=(true,10000),upper_
146         ↵ lower_displays=lds, disp_rot_its=10000,
147         ↵ converge_criterion="compression", converge_factor=1e-6)
148
149     end
150
151
152     for ms in totalmsvec
153         sev=evdict[ms*"_Separate"]
154         cev=evdict[ms*"_Combined"]
155         ratio=sev-cev
156         sig=ratio.val/ratio.err
157
158         println("$morph $ms & $(round(sev,digits=3)) &
159             ↵ $(round(cev,digits=3)) & $(round(ratio,digits=3)) &
160             ↵ $(round(sig,digits=1)) \\\\" \\\\hline")
161     end
162
163     colvec=[:darkmagenta,:darkorange,:green]
164

```

```

155 np_plt=plot_logn_MTDist([atg_measure_dict["NucPer"],spl_measure_dict["NucPer"],ctl_me
  ↪ "Mo.", "Spl Mo.", "Ctrl Mo."], "Mean Sectional Population per CMZ
  ↪ "Cell", "Posterior mean lh.")
156
157 savefig(np_plt, "/bench/PhD/Thesis/images/rys/morphnuclei.png")

```

---

### 16.9.14 /rys/a38.jl

```

1 using CSV,DataFrames,Distributions,NGRefTools,StatsBase,StatsPlots
2 gr()
3 default(legendfont = 10, guidefont = (12,:bold), tickfont = 10, guide =
  ↪ "x")
4
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
  ↪ eachrow(a38df)]
10 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,
  ↪ "Group"),[sib_measure_dict,rys_measure_dict])
27     for row in eachrow(gdf)
28         if row."D/V"=="D"
29             push!(dict["DCMZ"],row."PCNA +ve")
30             push!(dict["TCMZ"],row."Total PCNA")
31             push!(dict["NucPop"],row."Total nuclei")

```



```

63 log_mean_mass_comparator(rys_measure_dict["DCMZ"],sib_measure_dict["DCMZ"],labels=[ "r
  ↵ 5dpf sectional DCMZ population","sib 5dpf sectional DCMZ
  ↵ population"])
64 log_mean_mass_comparator(rys_measure_dict["VCMZ"],sib_measure_dict["VCMZ"],labels=[ "r
  ↵ 5dpf sectional VCMZ population","sib 5dpf sectional VCMZ
  ↵ population"])
65
66 VCMZ_plt=plot_logn_MTDist([sib_measure_dict["VCMZ"],rys_measure_dict["VCMZ"]],[:green
  ↵ CMZ Population","Posterior mean lh."]
67 annotate!([(15,2.25,Plots.text("D",18))])
68
69
70 Vol_plt=plot_n_MTDist([sib_measure_dict["Volume"],rys_measure_dict["Volume"]],[:green
  ↵ nuclear volume","Posterior mean lh."]
71 annotate!([( (.2,105,Plots.text("E",18))])
72
73 Sph_plt=plot_n_MTDist([sib_measure_dict["Sphericity"],rys_measure_dict["Sphericity"]]
  ↵ nuclear sphericity","Posterior mean lh."]
74 annotate!([( (.65,325,Plots.text("F",18))])
75 plot!(legend=:top)
76
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")

```

---

### 16.9.15 /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,
24   ↪ "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
34       ↪ 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]=5
46 gmc[2]=1.e-15
47 gmmdir="/bench/PhD/NGS_binaries/BSS/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=50
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_μ=300.;min_μ=0.
67 max_λ=.3;min_λ=0.
68 prior=[Uniform(min_μ,max_μ),Uniform(min_λ,max_λ)]
69 box=[min_μ max_μ;min_λ max_λ]
70
71 s_max_μ=1.;s_min_μ=0.
72 s_max_λ=5e5;s_min_λ=1e3
73 s_prior=[Uniform(s_min_μ,s_max_μ),Uniform(s_min_λ,s_max_λ)]
74 s_box=[s_min_μ s_max_μ;s_min_λ s_max_λ]
75
76 for ms in popmsvec
77     c_ens=gmcdir*/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
86     ← evdict[ms*"Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_displa
87     ← lower_displays=lds, disp_rot_its=10000,
88     ← converge_criterion="compression", converge_factor=1e-6)
89
90     for (msd,p) in zip([sib_measure_dict,rys_measure_dict],[ "/s_" , "/r_" ])
91         ens=gmcdir*p*ms
92         obs=msd[ms]
93         if isfile(ens*/ens")
94             e=deserialize(ens*/ens")
95         else
96             e=LogNormal_Ensemble(ens,n_models, obs, prior, box, gmc ... )
97         end

```

```

95     ↵ evdict[ms*"_Separate"]+=converge_ensemble!(e,backup=(true,10000),upper_di
96     ↵ lower_displays=lds, disp_rot_its=10000,
97     ↵ converge_criterion="compression", converge_factor=1e-6)
98 end
99 for ms in normmmsvec
100    c_ens=gmcdir*"/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
113    ↵ evdict[ms*"_Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_displa
114    ↵ lower_displays=lds, disp_rot_its=10000,
115    ↵ converge_criterion="compression", converge_factor=1e-6)
116
117 for (msd,p) in zip([sib_measure_dict,rys_measure_dict],["/s_","/r_"])
118    ens=gmcdir*p*ms
119    obs=msd[ms]
120    if isfile(ens*"/ens")
121        e=deserialize(ens*"/ens")
122    else
123        if ms=="Sphericity"
124            e=Normal_Ensemble(ens,n_models, obs, s_prior, s_box,
125                                ↵ gmc ... )
126        else
127            e=Normal_Ensemble(ens,n_models, obs, prior, box, gmc ... )
128        end
129    end

```

```

127      ↵ evdict[ms*"Separate"]+=converge_ensemble!(e,backup=(true,10000),upper_di
128      ↵ lower_displays=lds, disp_rot_its=10000,
129      ↵ converge_criterion="compression", converge_factor=1e-6)
130 end
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

```

---

### 16.9.16 /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")
28 end
29
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
40             ↪ subplot=bar(1:1:sl,s_data,barwidths=1,color=:green,xticks=(1:1:sl+rl,
41             ↪ for i in 1:sl],[ "R" for i in 1:rl])),legend=:none,
42             ↪ xlabel=ms, margin=0mm,ylabel="Caspase-3+ cells")
43         else
44
45             ↪ subplot=bar(1:1:sl,s_data,barwidths=1,color=:green,xticks=(1:1:sl+rl,
46             ↪ for i in 1:sl],[ "R" for i in 1:rl])),legend=:none,
47             ↪ xlabel=ms, leftmargin=-7.5mm, yformatter=_→ "")
48         end
49         bar!(sl+1:1:sl+rl,r_data,color=:darkmagenta, barwidths=1.)
50         push!(subplots,subplot)
51     end
52     title = plot(title = "$(Int64(age)) dpf", grid = false, showaxis =
53         ↪ nothing, bottom_margin = -35Plots.px)
54     l=@layout [t{.02h};[a{.25w} b{.25w} c{.25w} d{.25w}]]
55     half=plot(title, subplots...,layout=l,link=:y)
56     push!(plot_halves,half)
56 end
57
58 combined=plot(plot_halves...,layout=grid(1,2), size=(900,450))
59
60 savefig(combined, "/bench/PhD/Thesis/images/rys/caspase.png")
61
62 println("Rys CMZ caspase 4dpf: $(mean(rys_ms_dict["CMZ"])[1]) ±
63     ↪ $(std(rys_ms_dict["CMZ"])[1]))"

```

```

57 println("Rys CMZ caspase 6dpf: $(mean(rys_ms_dict["CMZ"][[2])) ±
→   $(std(rys_ms_dict["CMZ"][[2]))")
58
59 println("Rys central caspase 4dpf: $(mean(rys_ms_dict["Central"][[1])) ±
→   $(std(rys_ms_dict["Central"][[1))))")
60 println("Rys central caspase 6dpf: $(mean(rys_ms_dict["Central"][[2])) ±
→   $(std(rys_ms_dict["Central"][[2))))")
61 println("Sib central caspase 4dpf: $(mean(sib_ms_dict["Central"][[1])) ±
→   $(std(sib_ms_dict["Central"][[1))))")
62 println("Sib central caspase 6dpf: $(mean(sib_ms_dict["Central"][[2])) ±
→   $(std(sib_ms_dict["Central"][[2))))")

```

---

### 16.9.17 /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
→ 1.jpg")
4 sib36=sib36[1:2465,:]
5 sib110=load("/bench/PhD/Thesis/images/rys/C2 cell 1/Rys sib C2 110k cell
→ 1.jpg")
6 sib110=sib110[1:2465,:]
7 sib210=load("/bench/PhD/Thesis/images/rys/C2 cell 1/Rys sib C2 210k cell
→ 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")

```

---

### 16.9.18 /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
    ↪ qPCR.csv"
5 rand8pth="/bench/PhD/Thesis/datasets/8dpfRys-Histones random primers
    ↪ 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)]
55             length(wt_hk)>1 ?
56                 → wt_trg_m=measurement(mean(wt_trg),std(wt_trg)) :
57                 → wt_trg_m=measurement(mean(wt_trg),0.)
58             wt_ΔCt=wt_trg_m-wt_hk_m

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.)

```

```

60         smpl_trg=measure_dict[("rand",d,plate,smpl,ms)]
61         length(wt_hk)>1 ?
62             → smpl_trg_m=measurement(mean(smpl_trg),std(smpl_trg))
63             → : smpl_trg_m=measurement(mean(smpl_trg),0.)
64         smpl_DeltaCt=smpl_trg_m-smpl_hk_m
65         DeltaDeltaCt=smpl_DeltaCt-wt_DeltaCt
66         fold_change=2^(-DeltaDeltaCt)
67         ms in keys(msdict) ? push!(msdict[ms][xidx],fold_change)
68             → : msdict[ms]=[[fold_change],Vector{Float64}]()
69     end
70   end
71 end
72
73 msvec=[ "H2A" , "H2B" , "H3" , "H4" ]
74
75 for ms in msvec
76   for d in [6,8]
77     xidx=findfirst(isequal(d),X)
78     for plate in [1,2,3]
79       wt_hk=measure_dict[("dt",d,plate,"WT cDNA","ActinB")]
80       length(wt_hk)>1 ? wt_hk_m=measurement(mean(wt_hk),std(wt_hk))
81           → : wt_hk_m=measurement(mean(wt_hk),0.)
82       wt_trg=measure_dict[("dt",d,plate,"WT cDNA",ms)]
83       length(wt_trg)>1 ?
84           → wt_trg_m=measurement(mean(wt_trg),std(wt_trg)) :
85           → wt_trg_m=measurement(mean(wt_trg),0.)
86       wt_DeltaCt=wt_trg_m-wt_hk_m
87
88       for (smpl, msdict) in zip(["Sib cDNA","Rys
89           → cDNA"],[dt_sib,dt_rys])
90         smpl_hk=measure_dict[("dt",d,plate,smpl,"ActinB")]
91         length(smpl_hk)>1 ?
92             → smpl_hk_m=measurement(mean(smpl_hk),std(smpl_hk)) :
93             → smpl_hk_m=measurement(mean(smpl_hk),0.)
94         smpl_trg=measure_dict[("dt",d,plate,smpl,ms)]
95         length(smpl_trg)>1 ?
96             → smpl_trg_m=measurement(mean(smpl_trg),std(smpl_trg))
97             → : smpl_trg_m=measurement(mean(smpl_trg),0.)
98         smpl_DeltaCt=smpl_trg_m-smpl_hk_m
99         DeltaDeltaCt=smpl_DeltaCt-wt_DeltaCt
100        fold_change=2^(-DeltaDeltaCt)

```

```

91         ms in keys(msdict) ? push!(msdict[ms][xidx],fold_change)
92             : msdict[ms]=[[fold_change],Vector{Float64}()])
93     end
94   end
95 end
96
97 plot_halves=Vector{Plot}()
98 for ((sib_dict, rys_dict),assay_name) in
99   zip([(rand_sib,rand_rys),(dt_sib,dt_rys)],["Total transcript",
100        "Polyadenylated Transcript"])
101    subplots=Vector{Plot}()
102    for (n,ms) in enumerate(msvec)
103      plt=0.
104      mx=0.
105      mnn=Inf
106      for (dict,color,symbol,leg) in zip([sib_dict,
107          rys_dict],[:green,:darkmagenta,:circle,:diamond],["Sib","rys"])
108        means=[mean(valvec) for valvec in dict[ms]]
109        stdevs=[std(valvec) for valvec in dict[ms]]
110        upper=Vector{Float64}()
111        mns=[mn.val for mn in means]
112        stds=[sd.val for sd in stdevs]
113        lower=Vector{Float64}()
114
115        for (mn,sd) in zip(mns,stds)
116          dist=Normal(mn,sd)
117          push!(upper,quantile(dist,.975)-mn)
118          push!(lower,mn-quantile(dist,.025))
119        end
120
121        xs=vcat([[X[n] for i in 1:length(dict[ms][n])] for n in
122            1:length(X)]...)
123        ys=vcat([[m.val for m in dict[ms][n]] for n in
124            1:length(X)]...)
125
126        mx=max(mx,maximum(vcat(upper,ys)))
127        mnn=min(mnn,minimum(vcat(lower,ys)))
128
129        if dict==sib_dict
130          if n!=4 && n!=1

```

```

126     plt=scatter(xs,ys, marker=symbol, color=color,
127                 ↪ markerstrokecolor=color, markersize=3,
128                 ↪ label=leg*" data", showaxis=:y, xticks=X,
129                 ↪ xformatter=_→ "", ylabel="Fold
130                 ↪ change", xlims=[5.5,8.5], legend=:none)
131
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
140     else
141         plt=scatter(xs,ys, marker=symbol, color=color,
142                     ↪ markerstrokecolor=color, markersize=3,
143                     ↪ label=leg*" data", xticks=X, ylabel="Fold
144                     ↪ change", xlabel="Age (dpf)",
145                     ↪ xlims=[5.5,8.5], legend=:none)
146
147     end
148
149     assay_combined=plot(subplots ..., layout=grid(4,1), link=:x)
150     push!(plot_halves,assay_combined)
151
152 end

```

```

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 rr_h2a=[Normal(m,s) for (m,s) in
190   ← zip(mean.(rand_rys["H2A"]),std.(rand_rys["H2A"]))]
191 rs_h2a=[Normal(m,s) for (m,s) in
192   ← zip(mean.(rand_sib["H2A"]),std.(rand_sib["H2A"]))]
193 rand_h2a_joint=exp(sum(logccdf.(rr_h2a,[d.μ for d in rs_h2a])))
194
195
196

```

```

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

```

---

### 16.9.19 /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"
55                     rys_measure_dict["EdU"][xidx].+=mdf."EdU +ve, PCNA
56                                     → +ve"
57             end
58         end
59     end
60 end
61
62 end

```

```

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
106 uds=Vector{Vector{Function}}([[convergence_display],[evidence_display],[info_display]])
107 lds=Vector{Vector{Function}}([[model_obs_display],[ensemble_display]])
108
109 n_models=50
110
111 evdict=Dict{String,Measurement}()
112
113 for ms in ["TCMZ","EdU"]
114     evdict[ms*"Combined"]=measurement(0,0)
115     evdict[ms*"Separate"]=measurement(0,0)
116 end
117
118 max_μ=2000.
119 min_μ=0.
120 max_λ=500.
121 min_λ=1e-6
122
123 prior=[Uniform(min_μ,max_μ),Uniform(min_λ,max_λ)]
124 box=[min_μ max_μ;min_λ max_λ]
125
126 gmmdir="/bench/PhD/NGS_binaries/BSS/A38/"
127
128 for ms in ["TCMZ","EdU"]
129     for (n,x) in enumerate(XCMZ)
130         c_ens=gmmdir*/c_*ms*string(Int64(x))
131         c_obs=vcat(sib_measure_dict[ms][n],rys_measure_dict[ms][n])
132         if isfile(c_ens*/ens")
133             e=deserialize(c_ens*/ens")
134         else
135             e=LogNormal_Ensemble(c_ens,n_models, c_obs, prior, box,
136             → gmc ... )
136         end
137
138         → evdict[ms*"Combined"]+=converge_ensemble!(e,backup=(true,10000),upper_di
139         → lower_displays=lds, disp_rot_its=10000,
140         → converge_criterion="compression", converge_factor=1e-6)

```

```

141     for (msd,p) in
142         → zip([sib_measure_dict,rys_measure_dict],[ "/s_ ", "/r_ "])
143         ens=gmcdir*p*ms*string(Int64(x))
144         obs=msd[ms][n]
145         if isfile(ens*"/ens")
146             e=deserialize(ens*"/ens")
147         else
148             e=LogNormal_Ensemble(ens,n_models, obs, prior, box,
149             → gmc ... )
150         end
151     end
152 end

```

---

### 16.9.20 /rys/ryspont.jl

```

1 using CSV,DataFrames,Distributions,NGRefTools,StatsBase,StatsPlots
2 gr()
3 default(fontsize = 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
16   → eachrow(a38df)]
17
18 sib_measure_dict=Dict{String,Vector{Vector{Float64}}}()
19 rys_measure_dict=Dict{String,Vector{Vector{Float64}}}()
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
42                         +ve")
43                 else
44                     push!(rys_measure_dict["DCMZ"],mdf."PCNA +ve")
45                     push!(rys_measure_dict["TCMZ"],mdf."PCNA +ve")
46                     push!(rys_measure_dict["EdU"],mdf."EdU +ve, PCNA
47                         +ve")
48             end
49             elseif unique(mdf."D/V/P/WE")[1] == "V"
50                 if sr=="sib"
51                     push!(sib_measure_dict["VCMZ"],mdf."PCNA +ve")
52                     sib_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"
53                     sib_measure_dict["EdU"][xidx].+=mdf."EdU +ve, PCNA
54                         +ve"
55                 else
56                     push!(rys_measure_dict["VCMZ"],mdf."PCNA +ve")
57                     rys_measure_dict["TCMZ"][xidx].+=mdf."PCNA +ve"
58                     rys_measure_dict["EdU"][xidx].+=mdf."EdU +ve, PCNA
59                         +ve"
60             end
61         end
62     end
63 end
64 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
96     in 1:length(XCMZ)]
97 sp_mean=[exp(mean(mt)) for mt in sp_login_mts]
98 sp_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in sp_login_mts]
99 sp_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in sp_login_mts]
100 rp_login_mts=[fit(MarginalTDist,log.(rys_measure_dict["TCMZ"])[n])) for n
101     in 1:length(XCMZ)]

```

```

101 rp_mean=[exp(mean(mt)) for mt in rp_logn_mts]
102 rp_lower=[exp(mean(mt))-exp(quantile(mt,.025)) for mt in rp_logn_mts]
103 rp_upper=[exp(quantile(mt,.975))-exp(mean(mt)) for mt in rp_logn_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_logn_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"][[n]]))
    ↵ 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=: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,"/bench/PhD/Thesis/images/rys/CMZontogeny.png")

```

---

### 16.9.21 /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)

```

---

### 16.9.22 /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 =
    ↳ "/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/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
      @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
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!"

```

---

### 16.9.23 /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
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!"

```

---

### 16.9.24 /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,
71                        ↳ security_group_desc, skeys, zone, ami, instance_type)
72 @info "Giving instances 90s to boot ... "
73 sleep(90)
74 @info "Spawning AWS cluster workers ... "
75 for ip in aws_ips
76     instance_pool=addprocs([(ip, instance_workers)], tunnel=true,
77                            ↳ topology=:master_worker, sshflags="-o StrictHostKeyChecking=no")
78     for worker in instance_pool
79         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)

```

---

### 16.9.25 /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 partplots=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
41     end
42 end
43
44 combined=plot(partplts..., layout=grid(3,3), link=:xy, size=(600,600))
45 savefig(combined, "/bench/PhD/Thesis/images/rys/bghmm.png")

```

---

### 16.9.26 /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
27     report=folder.partition_report
28     for id in report.best_repset
29         push!(job_ids,id)
30     end
31 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
37     @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
41         refined_chains[job_id]=[EM_step(1, chains[job_id][end].hmm, 0, 0,
        ← false)]
42     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)
62 for worker in remote_pool
63     load_dict[worker]=remote_config
64 end
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)
72 execute_EM_jobs!(worker_pool, em_jobset..., refined_path;
73     ↪ delta_thresh=delta_thresh, max_iterates=max_iterates,
74     ↪ load_dict=load_dict)

```

---

### 16.9.27 /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,
    ↳ Vector{Matrix{Float64}}())
45 rys_ui_sp = BioMotifInference.assemble_source_priors(no_sources,
    ↳ 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 =
60   → deserialize(string(sib_e_pth,'/','ens'))) :
61   (sib_e = BioMotifInference.IPM_Elusive(worker_pool, sib_e_pth,
62     → ensemble_size, sib_ui_sp, (falses(0,0), mixing_prior),
63     → sib_matrix, sib_obs, source_length_range, posterior_switch=true);
64     → serialize(string(sib_e_pth,'/','ens'),sib_e))
65 sib_e=[]; Base.GC.gc();
66
67
68 @info "Assembling rys IPM ensemble on uninformative priors ... "
69 isfile(string(rys_e_pth,'/','ens')) ? (rys_e =
70   → deserialize(string(rys_e_pth,'/','ens'))) :
71   (rys_e = BioMotifInference.IPM_Elusive(worker_pool, rys_e_pth,
72     → ensemble_size, rys_ui_sp, (falses(0,0), mixing_prior),
73     → rys_matrix, rys_obs, source_length_range, posterior_switch=true);
74     → serialize(string(rys_e_pth,'/','ens'),rys_e))
75 rys_e=[]; Base.GC.gc();
76
77
78 @info "Assembling combined IPM ensemble on uninformative priors ... "
79 isfile(string(combined_e_pth,'/','ens')) ? (combined_e =
80   → deserialize(string(combined_e_pth,'/','ens'))) :
81   (combined_e = BioMotifInference.IPM_Elusive(worker_pool,
82     → combined_e_pth, ensemble_size, combined_ui_sp, (falses(0,0),
83     → mixing_prior), combined_matrix, combined_obs,
84     → source_length_range, posterior_switch=true);
85     → serialize(string(combined_e_pth,'/','ens'),combined_e))
86 combined_e=[]; Base.GC.gc();
87
88 rmprocs(worker_pool)
89
90 @info "Job done!"

```

---

### 16.9.28 /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,
18   ↳ topology=:master_worker)
19
20 worker_pool=vcat(worker_pool, remote_pool)
21
22 #AWS PARAMS
23 @info "Setting up AWS wrangling ... "
24
25 security_group_name="calc1"
26 security_group_desc="calculation group"
27 ami="ami-0bb0f5609b995d39e"
28 skeys="AWS"
29 instance_type="c5a.24xlarge"
30 zone,spot_price=get_cheapest_zone(instance_type)
31 no_instances=5
32 instance_workers=48
33 bid=spot_price+.1
34
35 @assert bid ≥ spot_price
36
37 @info "Wrangling AWS instances ... "
38 aws_ips = spot_wrangle(no_instances, bid, security_group_name,
39   ↳ security_group_desc, skeys, zone, ami, instance_type)
40 @info "Giving instances 150s to boot ... "
41 sleep(150)
42
43 # aws_ips=["18.223.214.57", "18.224.4.81",
44   ↳ "3.17.13.150", "3.21.113.229", "18.222.211.44"]
45
46 @info "Spawning AWS cluster workers ... "

```

```

45 for ip in aws_ips
46     instance_pool=addprocs([(ip, instance_workers)], tunnel=true,
47     → topology=:master_worker, sshflags="-o StrictHostKeyChecking=no")
48     global worker_pool=vcat(worker_pool, instance_pool)
49 end
50
51 @info "Loading worker libraries everywhere ... "
52 @everywhere using BioMotifInference, Random
53 @everywhere Random.seed!(myid()*100000)
54
55 #JOB CONSTANTS
56 funcvec=full_perm_funcvec
57 models_to_permute=10000
58 func_limit=25
59 push!(funcvec, BioMotifInference.perm_src_fit_mix)
60 push!(funcvec, BioMotifInference.random_decorrelate)
61
62 min_clamps=fill(.01,length(funcvec))
63 min_clamps[2:3]==.1 #perm_src_fit_mix & permute_mix
64 min_clamps[8]==.1 #difference_merge
65 max_clamps=fill(.5,length(funcvec))
66 max_clamps[6:7]==.15 #ss and am
67 max_clamps[9]==.15 #sim merge
68
69 initial_weights= ones(length(funcvec))/length(funcvec)
70 # override_weights=fill(.034375,length(funcvec))
71 # override_weights[6:9]==.1;override_weights[13:14]==.1625
72 # override_time=20.
73
74 args=[Vector{Tuple{Symbol,Any}}() for i in 1:length(funcvec)]
75 args[end-1]=[:weight_shift_freq,0.],(:length_change_freq,1.),(:length_perm_range,1:1)
76 args[end]=[:iterates,50],(:source_permute_freq,.3),(:mix_move_range,1:10)
77
78 instruct = Permute_Instruct(funcvec, initial_weights, models_to_permute,
79     → func_limit;min_clmps=min_clamps, max_clmps=max_clamps, args=args)
80
81 display_rotation=[true,10,1,[[:tuning_disp,:lh_disp,:src_disp],[ :conv_plot,:liwi_disp
82
83 @info "Beginning nested sampling for sib ensemble ... "
84 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,25),
→   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!"

```

---

### 16.9.29 /rys/nested\_sampling/dif\_pos\_sample\_prep.jl

---

```

1 using BioSequences, Distributed, DataFrames, BioBackgroundModels, CSV,
→   Serialization
2 import StatsBase:sample
3
4 include("/bench/PhD/Thesis/Analyses/rys/position_analysis/position_overlap.jl")
5

```

```

6 #JOB PATHS AND CONSTANTS
7
8 sample_rows = 7092 #2 million bases
9
10 sib_pos =
11     ↳ "/bench/PhD/danpos_results/pooled/sib.Fnor.smooth.positions.xls"
12 rys_pos =
13     ↳ "/bench/PhD/danpos_results/pooled/rys.Fnor.smooth.positions.xls"
14
15 danio_genome_path =
16     ↳ "/bench/PhD/seq/GRCz11/GCA_000002035.4_GRCz11_genomic.fna"
17 danio_gen_index_path =
18     ↳ "/bench/PhD/seq/GRCz11/GCA_000002035.4_GRCz11_genomic.fna.fai"
19 danio_gff_path = "/bench/PhD/seq/GRCz11/Danio rerio.GRCz11.94.gff3"
20
21 refined_folders_path = "/bench/PhD/NGS_binaries/BBM/refined_folders"
22 selected_hmms = "/bench/PhD/NGS_binaries/BBM/selected_hmms"
23
24 sib_df_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_positions"
25 rys_df_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_positions"
26 combined_df_binary =
27     ↳ "/bench/PhD/NGS_binaries/BMI/combined_diff_positions"
28
29 sib_code_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_codes"
30 rys_code_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_codes"
31 combined_code_binary = "/bench/PhD/NGS_binaries/BMI/combined_diff_codes"
32
33 sib_diff_bg = "/bench/PhD/NGS_binaries/BMI/sib_diff_bg"
34 rys_diff_bg = "/bench/PhD/NGS_binaries/BMI/rys_diff_bg"
35 combined_diff_bg = "/bench/PhD/NGS_binaries/BMI/combined_diff_bg"
36
37 sib_fa = "/bench/PhD/thicweed_results/sib_nuc_position_sequences.fa"
38 sib_arch =
39     ↳ "/bench/PhD/thicweed_results/sib_nuc_position_sequences.fa_archs.txt"
40
41 rys_fa = "/bench/PhD/thicweed_results/rys_nuc_position_sequences.fa"
42 rys_arch =
43     ↳ "/bench/PhD/thicweed_results/rys_nuc_position_sequences.fa_archs.txt"
44
45 @info "Reading from position.xls ... "
46 sib_df=CSV.read(sib_pos)
47 rys_df=CSV.read(rys_pos)

```

```

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!"

```

---

### 16.9.30 /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 test_logZ = 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)
42
43 @info "Job done!"

```

---

### 16.9.31 /rys/nested\_sampling/prior\_test.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"
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
11 sib_code_binary = "/bench/PhD/NGS_binaries/BMI/sib_diff_codes"
12 rys_code_binary = "/bench/PhD/NGS_binaries/BMI/rys_diff_codes"
13 combined_code_binary = "/bench/PhD/NGS_binaries/BMI/combined_diff_codes"
14
15 sib_diff_bg = "/bench/PhD/NGS_binaries/BMI/sib_diff_bg"
16 rys_diff_bg = "/bench/PhD/NGS_binaries/BMI/rys_diff_bg"
17 combined_diff_bg = "/bench/PhD/NGS_binaries/BMI/combined_diff_bg"
18
19 sib_e_ui_pth = "/bench/PhD/NGS_binaries/BMI/sib_e_ui"
20 sib_e_inf_pth = "/bench/PhD/NGS_binaries/BMI/sib_e_inf"
21 rys_e_ui_pth = "/bench/PhD/NGS_binaries/BMI/rys_e_ui"
22 rys_e_inf_pth = "/bench/PhD/NGS_binaries/BMI/rys_e_inf"
23 combined_e_ui_pth = "/bench/PhD/NGS_binaries/BMI/combined_e_ui"
24 combined_e_inf_pth = "/bench/PhD/NGS_binaries/BMI/combined_e_inf"
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_Elenseble(worker_pool, sib_e_ui_pth,
    ↵ ensemble_size, sib_ui_sp, (falses(0,0), mixing_prior),
    ↵ sib_matrix, sib_obs, source_length_range);
    ↵ serialize(string(sib_e_ui_pth,'/','ens'),sib_e_ui))
86
87 @info "Assembling sib IPM ensemble on informative priors ... "
88 isfile(string(sib_e_inf_pth,'/','ens')) ? (sib_e_inf =
    ↵ deserialize(string(sib_e_inf_pth,'/','ens'))) :
89     (sib_e_inf = BioMotifInference.IPM_Elenseble(worker_pool,
    ↵ sib_e_inf_pth, ensemble_size, sib_inf_sp, (falses(0,0),
    ↵ mixing_prior), sib_matrix, sib_obs, source_length_range);
    ↵ serialize(string(sib_e_inf_pth,'/','ens'),sib_e_inf))
90
91 @info "Assembling rys IPM ensemble on uninformative priors ... "
92 isfile(string(rys_e_ui_pth,'/','ens')) ? (rys_e_ui =
    ↵ deserialize(string(rys_e_ui_pth,'/','ens'))) :
93     (rys_e_ui = BioMotifInference.IPM_Elenseble(worker_pool, rys_e_ui_pth,
    ↵ ensemble_size, rys_ui_sp, (falses(0,0), mixing_prior),
    ↵ rys_matrix, rys_obs, source_length_range);
    ↵ serialize(string(rys_e_ui_pth,'/','ens'),rys_e_ui))
94
95 @info "Assembling rys IPM ensemble on informative priors ... "
96 isfile(string(rys_e_inf_pth,'/','ens')) ? (rys_e_inf =
    ↵ deserialize(string(rys_e_inf_pth,'/','ens'))) :

```

```

97     (rys_e_inf = BioMotifInference.IPM_ElusiveEnsemble(worker_pool,
98      ↳ rys_e_inf_pth, ensemble_size, rys_inf_sp, (falses(0,0),
99      ↳ mixing_prior), rys_matrix, rys_obs, source_length_range);
100     ↳ serialize(string(rys_e_inf_pth,'/','ens'),rys_e_inf))

101 @info "Assembling combined IPM ensemble on uninformative priors ... "
102 isfile(string(combined_e_ui_pth,'/','ens')) ? (combined_e_ui =
103   ↳ deserialize(string(combined_e_ui_pth,'/','ens'))) :
104   (combined_e_ui = BioMotifInference.IPM_ElusiveEnsemble(worker_pool,
105     ↳ combined_e_ui_pth, ensemble_size, combined_ui_sp, (falses(0,0),
106     ↳ mixing_prior), combined_matrix, combined_obs,
107     ↳ source_length_range);
108     ↳ serialize(string(combined_e_ui_pth,'/','ens'),combined_e_ui))

109 @info "Assembling combined IPM ensemble on informative priors ... "
110 isfile(string(combined_e_inf_pth,'/','ens')) ? (combined_e_inf =
111   ↳ deserialize(string(combined_e_inf_pth,'/','ens'))) :
112   (combined_e_inf = BioMotifInference.IPM_ElusiveEnsemble(worker_pool,
113     ↳ combined_e_inf_pth, ensemble_size, combined_inf_sp,
114     ↳ (falses(0,0), mixing_prior), combined_matrix, combined_obs,
115     ↳ source_length_range);
116     ↳ serialize(string(combined_e_inf_pth,'/','ens'),combined_e_inf))

117 rmprocs(worker_pool)

118 @info "Fitting distributions ... "

119 sib_ui_dist=fit_mle(Normal,[model.log_Li for model in sib_e_ui.models])
120 sib_inf_dist=fit_mle(Normal,[model.log_Li for model in sib_e_inf.models])
121 @info "Is informative a better prior for sibs?"
122 println(sib_inf_dist.μ > sib_ui_dist.μ && quantile(sib_inf_dist,.5) >
123   ↳ sib_ui_dist.μ)

124 rys_ui_dist=fit_mle(Normal,[model.log_Li for model in rys_e_ui.models])
125 rys_inf_dist=fit_mle(Normal,[model.log_Li for model in rys_e_inf.models])
126 @info "Is informative a better prior for rys?"
127 println(rys_inf_dist.μ > rys_ui_dist.μ && quantile(rys_inf_dist,.5) >
128   ↳ rys_ui_dist.μ)

129 combined_ui_dist=fit_mle(Normal,[model.log_Li for model in
130   ↳ combined_e_ui.models])
131 combined_inf_dist=fit_mle(Normal,[model.log_Li for model in
132   ↳ combined_e_inf.models])

```

```

123 @info "Is informative a better prior for combined?"
124 println(combined_inf_dist.μ > combined_ui_dist.μ &&
    → quantile(combined_inf_dist,.5) > combined_ui_dist.μ)
125
126 @info "Job done!"

```

---

### 16.9.32 /rys/nested\_sampling/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=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
    .1 .1 .1 .7
    .1 .7 .1 .1]
20
21 struc_sig_2=[.1 .1 .7 .1
    .1 .1 .7 .1
    .7 .1 .1 .1]
22
23 periodicity=8
24 struc_frac_obs=.75
25
26
27
28
29 tata_box=[.05 .05 .05 .85
    .85 .05 .05 .05
    .05 .05 .05 .85
    .85 .05 .05 .05
    .425 .075 .075 .425
30
31
32
33

```

```

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,
    ↳ ones(length(funcvec))./length(funcvec),models_to_permute,100, .02;
    ↳ args=args)
96
97 @info "Assembling source priors ... "
98 prior_array= Vector{Matrix{Float64}}{}()
99 source_priors = BioMotifInference.assemble_source_priors(no_sources,
    ↳ prior_array)
100
101 @info "Assembling ensemble 1 ... "
102 isfile(e1*"/ens") ? (ens1 = deserialize(e1*"/ens")) :
103     (ens1 = IPM_Eensemle(worker_pool, e1, ensemble_size, source_priors,
    ↳ (falses(0,0), mixing_prior), bg_scores1, obs1,
    ↳ source_length_range))
104
105 @info "Converging ensemble 1 ... "
106 logZ1 = BioMotifInference.converge_ensemble!(ens1, instruct, worker_pool,
    ↳ .001, backup=(true,250), wk_disp=false, tuning_disp=true,
    ↳ ens_disp=false, conv_plot=true, src_disp=true, lh_disp=false,
    ↳ liwi_disp=false)
107
108 @info "Assembling ensemble 2 ... "

```

```

109 isfile(e2*/ens") ? (ens1 = deserialize(e2*/ens")) :
110     (ens2 = IPM_Engineer(worker_pool, e2, ensemble_size, source_priors,
111     ↳ (falses(0,0), mixing_prior), bg_scores2, obs2,
112     ↳ source_length_range))
113
114 @info "Converging ensemble 2 ... "
115 logZ2 = BioMotifInference.converge_ensemble!(ens2, instruct, worker_pool,
116     ↳ .001, backup=(true,250), wk_disp=false, tuning_disp=true,
117     ↳ ens_disp=false, conv_plot=true, src_disp=true, lh_disp=false,
118     ↳ liwi_disp=false)
119
120 @info "Assembling ensemble 3 ... "
121 isfile(e3*/ens") ? (ens1 = deserialize(e3*/ens")) :
122     (ens3 = IPM_Engineer(worker_pool, e3, ensemble_size, source_priors,
123     ↳ (falses(0,0), mixing_prior), bg_scores3, obs3,
124     ↳ source_length_range))
125
126 @info "Converging ensemble 3 ... "
127
128 logZ3 = BioMotifInference.converge_ensemble!(ens3, instruct, worker_pool,
129     ↳ .001, backup=(true,250), wk_disp=false, tuning_disp=true,
130     ↳ ens_disp=false, conv_plot=true, src_disp=true, lh_disp=false,
131     ↳ liwi_disp=false)
132
133 @info "Evidence ratio of joint 182 / 3:"
134 ratio=exp((logZ1+logZ2)-logZ3)
135 println(ratio)
136
137 rmprocs(worker_pool)

```

---

### 16.9.33 /rys/position\_analysis/position\_overlap.jl

---

```

1 using DataFrames, BioSequences, BioBackgroundModels, CSV, FASTX,
2     ↳ Statistics
3
4 function add_position_sequences!(df::DataFrame, genome_path::String,
5     ↳ genome_idx_path::String)
6     scaffold_seq_record_dict::Dict{String,BioSequences.LongSequence} =
7         ↳ BioBackgroundModels.build_scaffold_seq_dict(genome_path,
8             ↳ genome_idx_path)

```

```

6     seqs=[BioSequences.LongSequence{DNAAlphabet{4}}() for i in
7         1:size(df,1)]
8     df[!, :seq] .= seqs
9
10    Threads.@threads for entry in eachrow(df)
11        entry.seq=BioBackgroundModels.fetch_sequence(entry.chr,
12            → scaffold_seq_record_dict ,entry.start, entry.end, '+')
13    end
14
15    function get_cluster!(df::DataFrame, fasta::String, arch::String)
16        df[!, :cluster] .= zeros(Int64, size(df,1))
17        art=CSV.read(arch, header=0)
18
19        position_vec=Vector{Vector{Int64}}()
20        reader=FASTA.Reader(open(fasta))
21        for (n, entry) in enumerate(reader)
22            scaffold = FASTA.identifier(entry)
23            desc_array = split(FASTA.description(entry))
24            pos_start = parse(Int64, desc_array[2])
25            pos_end = parse(Int64, desc_array[4])
26            seq = FASTA.sequence(entry)
27
28            idx = filter(in(findall(chr→chr=scaffold, df.chr)),
29                → findall(start→start=pos_start, df.start))
30
31            if length(idx)==1
32                match=df[idx[1],:]
33                @assert match.end == pos_end
34                @assert match.seq==seq
35                match.cluster=art.Column1[n]
36            end
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)

```

```

45
46     if scaffold != "MT"
47         desc_array = split(FASTA.description(entry))
48         pos_start = parse(Int64, desc_array[2])
49         pos_end = parse(Int64, desc_array[4])
50         seq = FASTA.sequence(entry)
51
52         if !hasambiguity(seq)
53             push!(position_df, [scaffold, pos_start, pos_end, seq])
54         end
55     end
56 end
57
58 close(position_reader)
59 return position_df
60 end
61
62 function observation_setup(position_df::DataFrame; order::Int64=0,
63     ↪ symbol::Symbol=:Seq)
63     order_seqs =
64         ↪ BioBackgroundModels.get_order_n_seqs(Vector{LongSequence{DNAAlphabet{2}}})(pos
65         ↪ symbol],order)
64     coded_seqs = BioBackgroundModels.code_seqs(order_seqs)
66
66     return coded_seqs
67 end
68
69 function map_positions!(base_df, map_df)
70     df_rows = size(base_df,1)
71     base_df[!, :mapped_pos] = zeros(Int64, df_rows)
72     base_df[!, :overlap_bp] = zeros(Int64, df_rows)
73     base_df[!, :rel_overlap] = zeros(df_rows)
74
75     chrgroups=groupby(base_df, :chr)
76
77     for chr_df in chrgroups #split the df by scaffold so base numbers are
78         ↪ local to scaffold
79         base_chr = chr_df.chr[1]
80         matched=false
81         for map_chr_df in groupby(map_df, :chr)
82             base_chr = map_chr_df.chr[1] && subDFmap!(chr_df,
83                 ↪ map_chr_df)
84             matched=true

```



```

109         elseif (mapped_start ≤ search_end <
110             ↵ mapped_end) #3' end of searched position
111             ↵ overlaps with the mapped position
110             ↵ push!(overlaps,(search_end-mapped_start+1))
111             #println("3' overlap")
112         end
113     end
114
115     if length(mapped_idxs)>0 #1 or more mapped
116         ↵ candidates: take the one with the most
117         ↵ overlap
116         ↵ overlap, oidx = findmax(overlaps)
117         ↵ map_idx = mapped_idxs[oidx]
118         ↵ position.mapped_pos=map_idx
119         ↵ position.overlap_bp=overlap
120         ↵ position.rel_overlap=overlap /
121             ↵ (search_end-search_start+1)
121     end
122   end
123 end
124
125 sib_pos =
126   ↵ "/bench/PhD/danpos_results/pooled/sib.Fnor.smooth.positions.xls"
126 rys_pos =
127   ↵ "/bench/PhD/danpos_results/pooled/rys.Fnor.smooth.positions.xls"
127 sib_refpos =
128   ↵ "/bench/PhD/danpos_results/pooled/sib.Fnor.smooth.positions.ref_adjust.xls"
128 rys_refpos =
129   ↵ "/bench/PhD/danpos_results/pooled/rys.Fnor.smooth.positions.ref_adjust.xls"
130
130 sib_df=CSV.read(sib_pos)
131 rys_df=CSV.read(rys_pos)
132 sib_refdf=CSV.read(sib_refpos)
133 rys_refdf=CSV.read(rys_refpos)
134
135 Threads.@threads for t in 1:4
136     t==1 && nucpos.map_positions!(sib_df, rys_df)
137     t==2 && nucpos.map_positions!(rys_df, sib_df)
138     t==3 && nucpos.map_positions!(sib_refdf, rys_refdf)
139     t==4 && nucpos.map_positions!(rys_refdf, sib_refdf)
140 end
141

```

```
142 println("Unmapped positions: sib
    ↳ $(length(findall(iszero,sib_df.mapped_pos))), sib_ref
    ↳ $(length(findall(iszero,sib_refdf.mapped_pos)))")
143 println("Unmapped positions: rys
    ↳ $(length(findall(iszero,rys_df.mapped_pos))), rys_ref
    ↳ $(length(findall(iszero,rys_refdf.mapped_pos)))")
144 println("Avg overlap: sib $(mean(sib_df.rel_overlap)), sib_ref
    ↳ $(mean(sib_refdf.rel_overlap))")
145 println("Avg overlap: rys $(mean(rys_df.rel_overlap)), rys_ref
    ↳ $(mean(rys_refdf.rel_overlap))")
```

---

# 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.
- [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.
- [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.
- [Bar15] Marcello Barbieri. *Code Biology*. Springer International Publishing, Cham, 2015.
- [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.
- [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.
- [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/ $\beta$ -Catenin Pathway. *Neuron*, 47(1):43–56, July 2005.
- [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.
- [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.
- [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.
- [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.

- [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.
- [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.
- [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.
- [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.
- [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.
- [HKB08] Herbert Hoijtink, Irene Klugkist, and Paul A. Boelen, editors. *Bayesian Evaluation of Informative Hypotheses*. Statistics for Social and Behavioral Sciences. Springer, New York, 2008.
- [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.
- [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.
- [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.
- [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.
- [Kan06] Kunihiko 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.
- [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.
- [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.
- [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.

- [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.
- [Lyo13] Louis Lyons. Discovering the Significance of 5 sigma. *arXiv:1310.1284 [hep-ex, physics:hep-ph, physics:physics]*, October 2013.
- [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.
- [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.
- [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.
- [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.
- [Rv08] Arjun Raj and Alexander van Oudenaarden. Stochastic gene expression and its consequences. *Cell*, 135(2):216–226, October 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.
- [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. page 8, 2019.
- [SMFW10] Erik B Sudderth, 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.
- [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.
- [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.
- [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, Marjijke 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. Vilter 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.

- [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.
- [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.
- [Wes00] M. Westerfield. The Zebrafish Book : A Guide for the Laboratory Use of Zebrafish. [http://zfin.org/zf\\_info/zfbook/zfbk.html](http://zfin.org/zf_info/zfbook/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.
- [Won15] Loksum Wong. Mutual antagonism of the paired-type homeobox genes, vsx2 and dmbx1, regulates retinal progenitor cell cycle exit upstream of ccnd1 expression. *Developmental Biology*, page 13, 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 Dungarwala, 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.