Project Diary for "Cosmological parameter estimation using Bayesian accelerated Machine learning"

Aleksandr Petrosyan

April 23, 2020

This is a GNU emacs Org file. It's a sophisticated formate used mainly for literate programming, and for creating outlines, time management, project management and importantly for the purposes of the present project: Facilities to track and timestamp every aspect of the aforementioned features.

This file contains the notes and the project diary for the "Cosmological Parameter estimation using accelerated Bayesian machine learning" project. The diary is a time-stamped list of activities with links to their results.

The descriptions of activities are terse.

The reason is, that the entire point of keeping a diary is to be able to tell if I had a **plan**, and have enough information to **reproduce** the research. For the first, the timestamps as well as the scheduled and deadline keywords are enough to tell when and how much time was spent on each topic. For the latter, this diary's notes should provide the minimum theoretical knowledge to make sense of the code. Since the code itself is timestamped (it's a git repository), is based on simpe intuitive concepts that do not normally require explanation. The few cases that do, are covered in the notes.

Any theoretical findings are found and **properly** explained in the paper. Code is self-documenting, where explanations are needed, they are provided in the docstrings.

Surveyed literature is compressed into the Notes section. So are any calculations necessary to understand how the re-partitioning works.

1 Michaelmas

1.1 Week 3

1.1.1 DONE Meet Dr Handley

SCHEDULED: *<2019-10-25 Fri>*
Discussed the project. Talked about Nested Sampling. Applications to Machine Learning.

Primordial power spectrum, and discretisation. This causes the Fourier transform *multipole plot* of the CMB. Multipole moment related to size of blotches on the picture.

1.2 Week 4

1.2.1 DONE Read the Abstract. ,

SCHEDULED: *<2019-10-28 Mon>*
Read Bayes in the sky Read about Bayesian analysis. Ω_k more significant than thought.

1.2.2 DONE Install PolyChord, Cobaya, anesthetic, CosmoChord.

SCHEDULED: *<2019-10-26 Sat>*
Issues with Mac OS X/XCode. GFortran was installed but not on PATH. PolyChord not Pip installable. language=bash,label= ,caption= ,captionpos=b,numbers=none C=cc CXX=cxx python setup.py install —user doesn't work on OS X.

Use Python3.

1.2.3 DONE EMail will and create a PR to fix Python version.

SCHEDULED: *<2019-10-27 Sun>*
Fixed. Merged. Working.

1.2.4 FAILED Reproduce plots.

SCHEDULED: *<2019-11-08 Fri>*

Laptop inoperative due to Liquid damage.

1.3 Week 5

Laptop inoperative due to liquid damage. Until week 2 of Vacation laptop remained under liquid damage. These entries filled retroactively from Todoist.

1.3.1 Talk on nested sampling *<2019-11-14 Thu>*

It's a method of inference. Much the same as least squares fitting, except it's not assuming the central limit theorem and works for arbitrary distributions of error.

1.3.2 DONE Start final report writeup *<2019-11-19 Tue>*

See git history.

1.3.3 DONE Finish reading Skilling.

SCHEDULED: *<2019-11-15 Fri>*
See notes.

1.3.4 DONE Implement Skilling's algorithm

SCHEDULED: *<2019-11-17 Sun>*
Tried to do on the PWF, using example code at the end of the paper. cite:skilling. Hard to do. Need laptop repaired. C code in the repository.

1.3.5 Note from meeting. *<2019-11-15 Fri>*

Implementing Skilling's algorithm wasn't necessary.

1.3.6 Note from meeting. *<2019-11-15 Fri>*

The idea of re-partitioning. Read Chen-Feroz-Hobson.

1.3.7 Note from meeting *<2019-11-15 Fri>*

Project writeup

1.4 Week 7

1.4.1 DONE Implement Multivariate Gaussian Likelihood *<2019-11-17 Sun>*

Used example code as template. See toy-models/multivariate-gaussian.py

1.4.2 DONE Investigate the C++ front-end. *<2019-11-19 Tue>*

PolyChord works as a framework. Unable to control many things including verbosity of output.

1.4.3 DONE Project report. *<2019-11-21 Thu>*

Example writeups.

1.5 Week 8

1.5.1 DONE Finalise Project report. *<2019-11-25 Mon>*

1.5.2 DONE Proof read the report *<2019-11-29 Fri>*

1.5.3 DONE Submit the report. *<2019-12-04 Wed>*

Good staplers are not in Cavendish. Had to re-print and re-submit because the one in Kavli chewed up the paper and the one at Rayleigh library was not functional.

1.6 Vacation Weeks

1.6.1 DONE Re-install software *<2019-12-04 Wed>*

a) `polychord` (GitHub) b) `anesthetic` (`pip`) c) `fgivenx` (`pip`)

1.6.2 DONE Line Fitting example. *<2019-12-09 Mon>*

See `0/extended-example.py`.

1.6.3 DONE Set up CSD3 login information. *<2019-12-19 Thu>*

PI Name: Will Handley PI Status: Research fellow PI Email: wh260@cam.ac.uk PI Phone: +44-(0)1223-764042 PI Department: Cavendish Laboratory (Astrophysics) PI School: Physical Sciences

Research Group: Astrophysics Department: Cavendish Laboratory School: Physical Sciences Service Level: Non-paying (SL3) only Project: (Leave blank)

End Date: 01/01/2021 (To give us time to write up) Compute Platforms: (leave blank) Dedicated nodes: (None) PI Declaration: tick yes

1.6.4 DONE Read about Bayesian statistics. ^{2} test.

Notes.

2 Lent

2.1 Week 1

2.1.1 Meet Dr Handley *<2020-01-17 Fri>*

Talked about the line fitting example.

2.1.2 DONE Re-factor the line-fitting examples

SCHEDULED: *<2020-01-17 Fri>*
CLOCK: *[2020-01-17 Fri 22:58]–[2020-01-18 Sat 00:25] (1:27)*
CLOCK: *[2020-01-17 Fri 20:32]–[2020-01-17 Fri 22:40] (2:08)*
`./toy-models/0/0.1extended_example.py`. Do I need to generate the data? Can I use the parameter covariance matrix to emulate the data?

better to have a data generator and not use it than to not have it and need it.

2.1.3 DONE Implement the Data generator

SCHEDULED: *<2020-01-20 Mon>*
CLOCK: *[2020-01-22 Wed 09:15]–[2020-01-22 Wed 13:15] (4:00)*
CLOCK: *[2020-01-21 Tue 09:03]–[2020-01-21 Tue 16:30] (7:27)*
CLOCK: *[2020-01-20 Mon 09:17]–[2020-01-20 Mon 12:20] (3:03)*
DEADLINE: *<2020-01-22 Wed>* Implemented! It works! `./toy-models/0/0.2DataCovarianceWithGenerator/DataGenerator.py` Probably overengineered.

Simple noise overlayed on top of data predicted by model. Use chi-squared likelihood fit from `./toy-models/0/0.2DataCovarianceWithGenerator/Polychord.py`.

Credit

`http://jrmeyer.github.io/machinelearning/2017/08/18/mle.html`
update: $\ln_z$ is an exceedingly bad name for the loglikelihood.

2.2 Week 2

2.2.1 DONE meet Dr Handley.

SCHEDULED: *<2020-01-22 Wed>*
The data generator isn't necessary. I was using too many live points. 200 live points is good-enough for publication quality results. Reduce that to 20. Try it with Planck chains.

`https://doi.org/10.5281/zenodo.3371152`

Try to see if the parameter covariance matrices yield the same results as the Planck chains.

2.2.2 DONE Refactor the following script. Check that posteriors agree with paper

SCHEDULED: *<2020-01-23 Thu>*
CLOCK: *[2020-01-29 Wed 21:30]–[2020-01-29 Wed 22:00] (0:30)*
CLOCK: *[2020-01-23 Thu 15:27]–[2020-01-23 Thu 19:35] (4:08)*
CLOCK: *[2020-01-23 Thu 21:27]–[2020-01-23 Thu 23:35] (2:08)*

language=Python,label= ,caption= ,captionpos=b,numbers=none import numpy import pypolychord from pypolychord.settings import PolyChordSettings from pypolychord.priors import UniformPrior

nDims = 2 nDerived = 0 mu = numpy.array([0,0]) Sig = numpy.array([[1,0.99],[0.99,1]]) invSig = numpy.linalg.inv(Sig) norm = numpy.linalg.slogdet(2*numpy.pi*Sig)[1]/2

def likelihood(theta): """ Simple Gaussian Likelihood""" logL = - norm - (theta - mu) @ invSig @ (theta - mu) / 2 return logL, []

def prior(hypercube): """ Uniform prior from [-1,1]^D.""" $returnUniformPrior(-20,20)(hypercube)$

settings = PolyChordSettings(nDims, nDerived) settings.file$_root ='\ gaussian'_2 settings.nlive =$ 200$settings.do_clustering = Truesettings.read_resume = False$

output = pypolychord.run$_polychord(likelihood, nDims, nDerived, settings, prior)fromanesthetic$
$NestedSamples(root='\ ./chains/gaussian')fig, ax = samples.plot_2d([0,1])samples_1 =$
$NestedSamples(root='\ ./chains/gaussian'_1)samples_1.plot_2d(ax)samples_2 =$
$NestedSamples(root='\ ./chains/gaussian'_2)samples_2.plot_2d(ax)$

samples = NestedSamples(root='/data/will/tension/runs/lcdm/chains/planck') samples.plot$_2d(['logA','ns'])$

params = samples.columns[:27] Sig = samples[params].cov().values mu = samples[params].mean().values invSig = numpy.linalg.inv(Sig) norm = numpy.linalg.slogdet(2*numpy.pi*Sig)[1]/2 nDims = len(mu)

ranges = numpy.array([[0.019,0.025], [0.095,0.145], [1.03,1.05], [0.01,0.4], [2.5,3.7], [0.885,1.04], [0.9,1.1], [0,200], [0,1], [0,10], [0,400], [0,400], [0,400], [0,400], [0,10], [0,50], [0,50], [0,100], [0,400], [0,10], [0,10], [0,10], [0,10], [0,10], [0,10], [0,3], [0,3]])

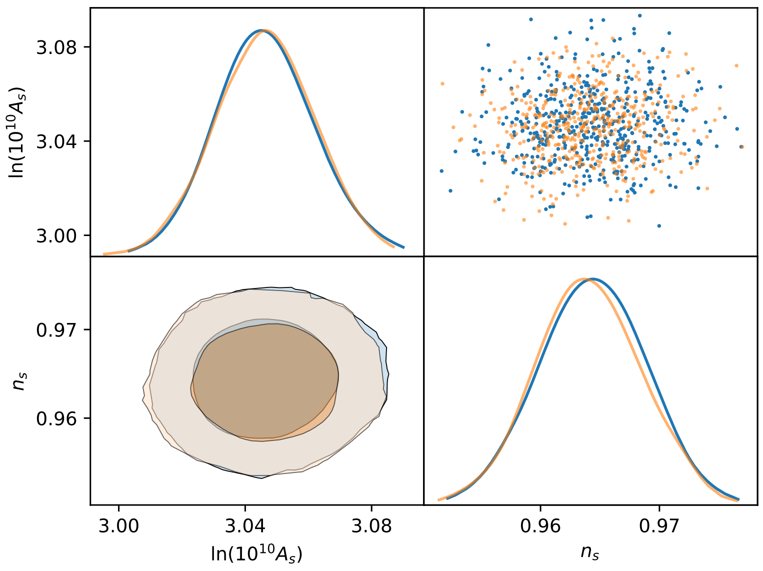
def prior(hypercube): """ Uniform prior from [-1,1]^D."""$returnranges[:,0] + hypercube * (ranges[:,1] - ranges[:,0])$

settings = PolyChordSettings(nDims, nDerived) settings.file$_root =' gaussian'_2 settings.nlive =$ 200$settings.do_clustering = Truesettings.read_resume = False$

samples = NestedSamples(root='/data/will/tension/runs/lcdm/chains/planck') samples.plot$_2d(['logA','ns'])$

output = pypolychord.run$_polychord(likelihood, nDims, nDerived, settings, prior)$

Produces weird misfit.



update: EUREKA! Neither PolyChord nor anesthetic when loading the samples from disk actually re-name the parameters. I was simply comparing the runs of different parameters, no wonder their posteriors had nothing in common.

Need to use a `numpy.ndarray` as input. Convert into a pandas data frame and convert it back in the output.

Updated code is `./toy-models/1/1.0Exampleofparametercovariance.py`

2.2.3 DONE Implement Uncorrelated Gaussian

CLOCK: *[2020-01-27 Mon 09:36]–[2020-04-19 Sun 00:42] (1983:06)*
Be careful: erf maps R to $[-1, 1]$ but we need a mapping from $[0, 1]$ back to R. So use $^{-1}(2x - 1)$.

See `./toy-models/1/1.1parameter-covariance-withGaussians.py`

2.2.4 DONE Implement Correlated Gaussian *<2020-01-28 Tue>*

CLOCK: *[2020-01-28 Tue 10:42]–[2020-01-28 Tue 17:43] (7:01)*
Chloesky decomposition makes the off-diagonal elements of the Σ matrix more important. This is because the covariance matrix has certain properties: it's positive definite, invertible. So we can't just invent one.

$$\Sigma = \begin{pmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{pmatrix} = beginpmatrix1 - 1 - 1110101$$

$(2, 0.5, 0.5)\frac{}{{}_{3}\begin{pmatrix} 1 & 1 & 1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix}(1)}$ Has all of the aforementioned properties. It is diagonaliseable. This is an ellipsoid stretched across the corners and the edge midpoints of a unit hyper-cube. This has all of our needed properties.

2.3 Week 3

This week was highly unproductive. Not my fault. If you're interested to know, I passed out from exhaustion and malnutrition, and decided to let it go easy for one week.

IF you're also interested in **why** I passed out, and **why** I was malnourished, I suggest reading :

2.3.1 DONE Implement PPR. *<2020-02-03 Mon>*

DEADLINE: *<2020-02-07 Fri>*
SCHEDULED: *<2020-02-03 Mon>* Calculations neeeded for doing PPR. I've done this by hadn as well. Originally made the mistake of assuming

that the Gaussian is unbounded. I then realised that it wasn't even in the ordinary cases that I've considered previously.

The correction was of the form containing erf$^-1(\ldots)$ while I assumed it was the intergral over all of space σ/β.

2.3.2 DONE Test the Lasenby parameter.

SCHEDULED: *<2020-03-08 Sun>*
The 'Lasenby' parameter is the procedure described by Chen-Ferroz-Hobson to do re-partitioning, except we don't re-scale the log-likelihood.

This should not run faster. But it does.

`./toy-models/2/2.0ComparisonofruntimewithLasenbyparameter.py`

3 Easter "Vacation"

3.1 Week 1.

3.1.1 DONE Set up SSH access to CSD3.

CLOCK: *[2020-03-18 Wed 12:19]–[2020-03-18 Wed 19:00] (6:41)*
Covid 19 caused a major disruption. I was forced out of College, required to return to Armenia. Spent the entire week under quarantine. Thankfully had some internet access. I'm sure no-one is going to account for the time/stress incurred losses. Why would they. That would be putting the Brits that only need to stay in their homes at an unfair disadvantage.

1. **DONE** Find out how to bypass restriction. **CLOCK:** *[2020-05-18 Mon 13:23]–[2020-04-18 Sat 18:23] (-715:00)*
   Can't `ssh`. Connection rejected. Cambridge VPN not helping. Can't connect to it either.

2. **DONE** Write a script to probe for forwarding ports **CLOCK:** *[2020-03-18 Wed 13:48]–[2020-03-18 Wed 18:24] (4:39)*
   Found a forwarding port on the router. Use language=bash,label= ,caption= ,captionpos=b,numbers=none ssh -R 52.194.1.73:8080:localhost:80 ap886@login-hpc.cam.ac.uk to connect.

3.1.2 DONE install Cobaya

CLOCK: *[2020-03-19 Thu 19:00]–[2020-03-21 Sat 22:30] (51:30)*
Did it very thoroughly. MKL not identified. Will need to debug. Later. `mpirun cobaya` works. It doesn't output, but at this stage it's not important.

3.1.3 DONE Modify cobaya

CLOCK: *[2020-03-19 Thu 11:47]–[2020-03-21 Sat 22:47] (59:00)*

Cobaya is mess. There's no notion of OOP design. Overridin a class should **not** have an `initialize` method. Instead it should have an `__init__` method.

Tried to extract commented blocks out into functions. This is hard work. I'm not even paid to do it. Implementing a proper channel in `.yaml` will be untenable. It will take more time than this entire project to figure out how to do it without breaking anything.

I should speak to Dr Handley about this and discuss how Cobaya's sampler should be implemented.

update: I should create a fork. Implement in situ and reference in the project writeup. As much as I'd like to clean it up, it's not really even my project to worry about.

3.1.4 DONE Meet Dr Handley

CLOCK: *[2020-03-20 Fri 13:00]–[2020-03-20 Fri 13:40] (0:40)*
Minutes. Try to have a writeup. Mentioned that I already have one. Asked about using the cluster to run benchmarks. Given a go-ahead. This should save a lot of time, given that my laptop is nowhere near 64-cores.

3.1.5 DONE Writeup

SCHEDULED: *<2020-03-21 Sat>*
CLOCK: *[2020-03-21 Sat 12:27]–[2020-03-21 Sat 14:28] (2:01)*
CLOCK: *[2020-03-21 Sat 15:30]–[2020-03-21 Sat 17:53] (2:23)*
CLOCK: *[2020-03-21 Sat 19:00]–[2020-03-21 Sat 21:28] (2:28)*
CLOCK: *[2020-03-21 Sat 22:00]–[2020-03-22 Sun 02:31] (4:31)*
Available on Github. Refined introduction. Added Bayes' theorem to writeup. Migrated to add mnras.

3.2 Week 2

3.2.1 DONE Get Home

CLOCK: *[2020-03-29 Sun 18:44]–[2020-03-25 Wed 19:00] (-95:44)*
SCHEDULED: *<2020-03-25 Wed>*

3.2.2 DONE Set up home Computer

SCHEDULED: *<2020-03-28 Sat>*
Fresh install of Arch.

1. **DONE** install software **CLOCK:** *[2020-03-29 Sun 16:39]–[2020-03-30 Mon 19:41] (27:02)*

   (a) **DONE** numpy
   (b) **DONE** MPI
   (c) **DONE** anesthetic
   (d) **DONE** Cobaya
   (e) **DONE** PolyChord gfortran in the repos not the required version. Used AUR. Created venv.

2. **DONE** Add home computer's ssh key to CSD3

3.2.3 DONE Migrate to Mnras

CLOCK: *[2020-03-26 Thu 10:41]–[2020-03-26 Thu 11:25] (0:44)*
Add everything needed to conform to the mnras style guide.

3.2.4 DONE Write an sbatch script to run Cobaya.

CLOCK: *[2020-03-24 Tue 12:49]–[2020-03-26 Thu 18:50] (54:01)*
Completely unsure about the memory and number of clusters. Need to ask Lukas Hergt about the values.

Cobaya's generated `.yaml` file is not working. Needs a few updates.

3.2.5 DONE Add figures

CLOCK: *[2020-03-29 Sun 14:22]–[2020-04-18 Sat 22:26] (488:04)*
CLOCK: *[2020-03-29 Sun 09:22]–[2020-03-29 Sun 11:10] (1:48)*
Used `tikzplotlib`. Plots in illustrations. Added illustrations of repartitioining functions.

3.2.6 DONE Writeup

CLOCK: *[2020-03-27 Fri 09:46]–[2020-03-27 Fri 10:47] (1:01)*
Added clairifcation. Bayes' theorem paper. Moved defs into table.

3.2.7 DONE Meet Dr Handley

CLOCK: *[2020-03-27 Fri 14:00]–[2020-03-27 Fri 14:55] (0:55)*
Received comments/compliments. More plots. Use the home computer and cluster to accelerate calculations.

3.2.8 DONE Implement offset.

SCHEDULED: *<2020-04-05 Sun>*
CLOCK: *[2020-04-05 Sun 17:00]–[2020-04-05 Sun 18:00] (1:00)*
Very easy to do. Just implement an intermediate class that takes a model and overrides the methods to offset the arguments of log-likelihood. `./framework/offset_model.py`

3.2.9 DONE Benchmark.

CLOCK: *[2020-03-24 Tue 09:32]–[2020-03-24 Tue 15:33] (6:01)*
CLOCK: *[2020-04-07 Tue 11:10]–[2020-04-07 Tue 15:10] (4:00)*
CLOCK: *[2020-04-05 Sun 09:08]–[2020-04-05 Sun 17:09] (8:01)*
Use nLike to estimate the time it took. Investigate the use of Kullback-leibler.

1. **DONE** Figure out why MPI is causing crashes if more than one instance of PolyChord is run. **CLOCK:** *[2020-04-05 Sun 19:28]–[2020-04-06 Mon 02:30] (4:32)*

   language=Python,label= ,caption= ,captionpos=b,numbers=none from mpi4py import MPI This is an example of why non-functional side-effect-y code is bad. importing a module should have **no** bearing on the code that's being run, except for making code available to the interpreter. This can cause Undefined behaviour if the module is being imported more than once. This can lead to many bugs like this one.

   DITTO, remove all global code from the modules and put the tests in their own functions: language=Python,label= ,caption= ,captionpos=b,numbers=none if $_{name}{}_{=='}{}_{main}{}_{':test()}$

2. **DONE** Plot the results.

   Benchmark PPR, SSPR mixture of Uniform and Gaussian. ./framework/benchmarks.py]].

If the mixture contains a Gaussian, and/or is a Gaussian itself with re-partitioning done properly, i.e. the Log-likelihood is coorrected, the PolyChord terminates with empty files which anesthetic cannot load.

This is weird. Need to ask Will about this.

3.2.10 DONE Investigate usefullenss of Kullback-leibler

CLOCK: *[2020-04-11 Sat 11:13]–[2020-04-11 Sat 13:13] (2:00)*
CLOCK: *[2020-04-07 Tue 17:11]–[2020-04-07 Tue 22:00] (4:49)*
Not a good metric. This is smaller if the posterior is wrong. It also doesn't correlate with performance.

It does correlate with perofmrnace, but not as much as one would hope.

`./illustrations/kullback-leibler.tex`

nLike is the dominant cost. Class takes 12 seconds to evaluate it, CAMB takes 3.

3.2.11 DONE Add Slurm warnings about failures

SCHEDULED: *<2020-03-28 Sat>*

3.2.12 FAILED Debug slurm failures

SCHEDULED: *<2020-03-28 Sat>*
DEADLINE: *<2020-03-29 Sun>* Weird tracebacks in output. MKL not recognised. Yaml doesnt recognise tabs.

3.2.13 DONE Migrate sbatch script to one of the examples.

SCHEDULED: *<2020-03-28 Sat>*

3.2.14 DONE Obtain posterior from Cobaya

SCHEDULED: *<2020-03-20 Fri>*
Takes too long. Many repeats of

language=Python,label= ,caption= ,captionpos=b,numbers=none [polychord] Calling PolyChord with arguments: [polychord] base$_dir : ./raw_polychord_output[polychord]boost_p$
$0[polychord]cluster_dir : ./raw_polychord_output/clusters[polychord]cluster_posteriors :$
$True[polychord]compression_factor : 0.36787944117144233[polychord]do_clustering :$
$True[polychord]equals : True[polychord]feedback : 1[polychord]file_root :$

$run[polychord]grade_{d}ims : [6, 21][polychord]grade_{f}rac : [12, 840][polychord]logzero : -1e{+}300[polychord]max_{n}dead : -1[polychord]maximise : False[polychord]nfail : -1[polychord]nlive : 675[polychord]nlives : [polychord]nprior : -1[polychord]num_{r}epeats : 135[polychord]posteriors : True[polychord]precision_{c}riterion : 0.001[polychord]read_{r}esume : True[polychord]seed : -1[polychord]write_{d}ead : True[polychord]write_{l}ive : True[polychord]write_{p}aramnames : False[polychord]write_{p}rior : True[polychord]write_{r}esume : True[polychord]write_{s}tats : True[polychord]Sampling$!

May be that it doesn't terminate correctly. edit: Lukas mentioned that the number of live points is too large. May be related.

Will need to edit `./cobaya/old/run.yaml`.

1. **DONE** Fix priors The priors are too broad. Ie already written an essay on why you can't just pick a prior out of thin air, so I'm very iffy about this decision.

2. **DONE** Reduce nLive Reduced the number of live points: `./cobaya/run.yaml`

   language=Python,label= ,caption= ,captionpos=b,numbers=none nlive: 126

3. **DONE** Re-install Cobaya. **CLOCK:** *[2020-03-27 Fri 16:44]–[2020-03-27 Fri 23:05] (6:21)*
   language=bash,label= ,caption= ,captionpos=b,numbers=none pip3 install cobaya –upgrade –user It doesn't respect the venv.

   It doesn't pull in PolyChord correctly. So much for "just works".

   Compile from source.

4. **DONE** Re-re-install Cobaya. **CLOCK:** *[2020-03-30 Mon 16:15]–[2020-03-31 Tue 02:51] (10:36)*
   It works only partially.

   (a) **DONE** Make sure that OpenBlas is loaded. **CLOCK:** *[2020-03-30 Mon 20:01]–[2020-03-30 Mon 23:02] (3:01)*
   language=bash,label= ,caption= ,captionpos=b,numbers=none python -c "from numpy import show$_{c}onfig; show_{c}onfig()$" $|grep'mkl\|openblas_{i}nfo' - A1$ Produces language=bash,label= ,caption= ,captionpos=b,numbers=none blas$_{m}kl_{i}nfo : NOTAVAILABLE--openblas_{i}nfo : libraries = ['openblas','openblas'] - -lapack_{m}kl_{i}nfo : NOTAVAILABLE$

   language=bash,label= ,caption= ,captionpos=b,numbers=none module load openblas doesn't work. Need to re-install `numpy`.

language=bash,label= ,caption= ,captionpos=b,numbers=none pip3 install numpy is not changing the output. Try installing from source.

Installing from source doesn't work.

EUREKA! Cobaya defaults to `icc`, meaning that the `openblas` is not the kind of linear algebra library it's even looking for.

SOLUTION: language=bash,label= ,caption= ,captionpos=b,numbers=none module load openblas module load intel/impi/2017.4/intel module load intel/mkl/2017.4 module load intel/compilers/2017.4 module load intel/libs/idb/2017.4 module load intel/libs/tbb/2017.4 module load intel/libs/ipp/2017.4 module load intel/libs/daal/2017.4 module load intel/bundles/complib/2017.4

(b) **DONE** add fix to `sbatch` script. **CLOCK:** *[2020-03-30 Mon 23:02]–[2020-03-30 Mon 23:30] (0:28)*

5. FAILED Run Cobaya **CLOCK:** *[2020-03-30 Mon 20:03]–[2020-04-23 Thu 09:50] (565:47)*

   Times out after 6 hours. relevant piece of sbatch script.

   language=bash,label= ,caption= ,captionpos=b,numbers=none SBATCH -J cobaya SBATCH –nodes=3 SBATCH –ntasks=96 SBATCH –time=06:00:00 SBATCH –mail-type=FAIL SBATCH –mail-type=BEGIN SBATCH –mail-type=TIME$_LIMIT_80$

SBATCH –qos INTR What's the deal? I keep being told by Dr Handley to do this, and it keeps failing.

   SBATCH –output latest

6. FAILED Re-run Cobaya Timing out. Need to ask Lukas

   (a) **DONE** Ask Lukas. Too many live points.

   FINALLY!!!! IT finished.

   I only needed to reduce nLive to 100, from 25d ($approx$700), this reduces the run-time sevenfold from a week down to one day!

   Since I've had many hurdles due to many versions of `pip`, maN y versions of python, and cobaya stubbornly choosing to ignore the virtual environmeN t it's in, I'd say that these difficulties have no academic relevance.

   If they do, they are a lesson to Anthony Lewis, that programming conventions exist for a reason. People like him, should probably not ignore the conventions of their language, mainly because as an academic, they cannot dedicate as much time as needed to maintaining their "I did it my way" versions of the program.

3.2.15 DONE Refactor of framework

SCHEDULED: *<2020-03-29 Sun>*
DEADLINE: *<2020-04-02 Thu>* Used PyCharm. Fixed a bug. Reference Python hash is not random, but linear in regions. For small values of θ this may and does cause issues.

fixed with

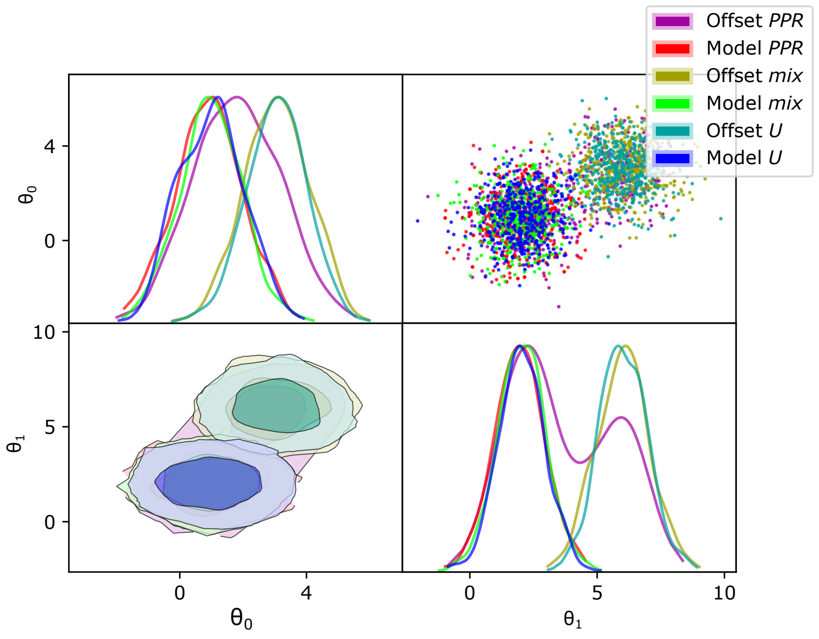
language=Python,label= ,caption= ,captionpos=b,numbers=none h = hash(tuple(t)) seed(h) r = random()

3.3 Week 3

3.3.1 DONE Test

SCHEDULED: *<2020-04-02 Thu>*
DEADLINE: *<2020-04-04 Sat>* All tests clear. Performance improved. Significantly. No bugs.

Also a discovery! Under some circumstances PPR can break.



. In the same environment, Gaussian under SSPR finishes faster and gets the right answer. Under the same circumstances PPR inside SSPR also finishes faster.

Choosing which one to include is like choosing your favourite child. I could make the case that SSPR makes the simulation more robust if wrapped inside PPR. On the other hand SSPR doesn't need PPR. Maybe give Hobson and Feroz some credit here. I'm already being overly negative about their discovery, even though my discovery is based on theirs. PPR it is then.

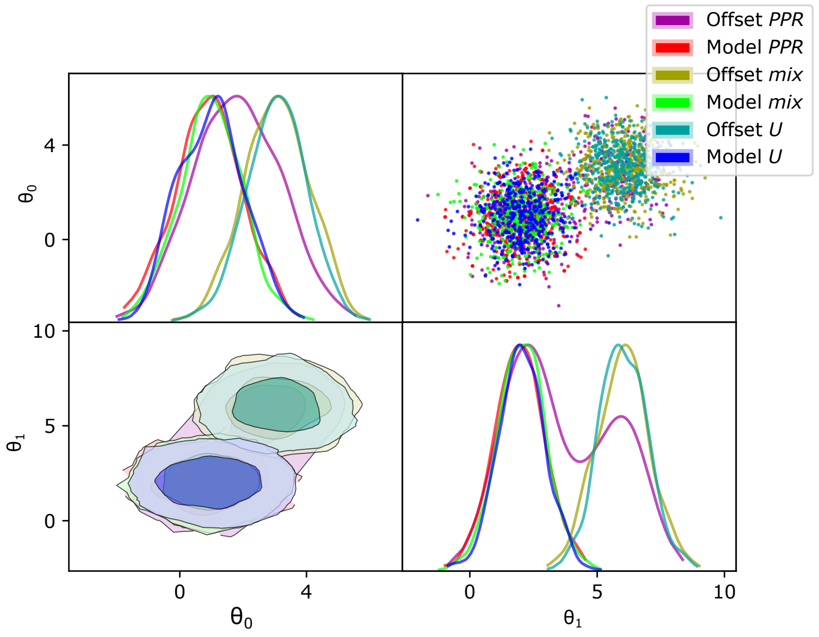
3.3.2 DONE Project presentations.

SCHEDULED: *<2020-04-06 Mon>*
According to Charles Smith we need to present our findings.

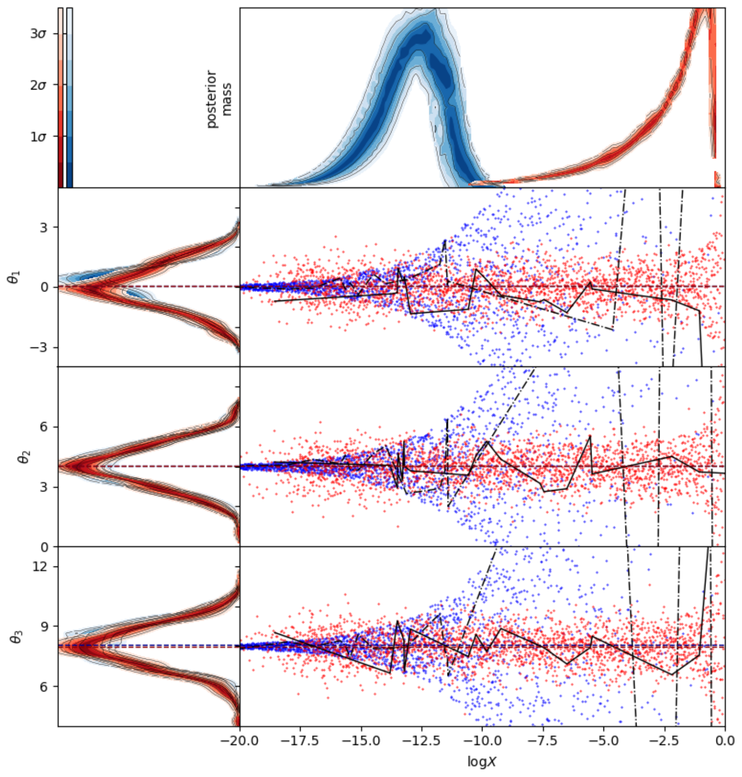
update: Meeting is scheduled.

3.3.3 DONE Document findings.

`./illustrations/benchmark.tex`



discussed here.



`./illustrations/scaling-kld.tex` documented in the write-up: Results and Discussion.

3.3.4 DONE e-mail Dr Handley about findings.

Says he's impressed. I thought I was behind schedule. Need to add more Kullback-Leibler stuff. Maybe see how offsets affect the Kullback Leibler divergence.

3.4 Week 4

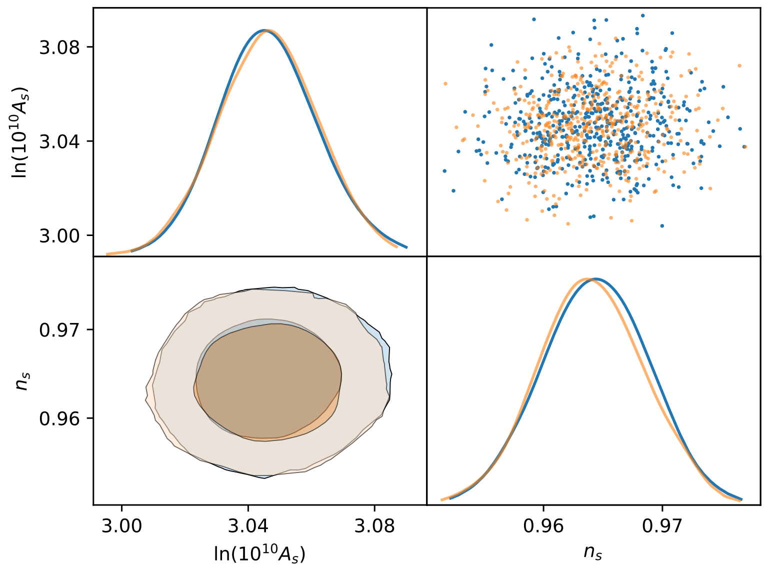
3.4.1 DONE Add Kullback-Leibler Divergence plots

CLOCK: *[2020-04-13 Mon 18:26]–[2020-04-18 Sat 22:47] (124:21)*
Kullback Leibler divergence is useful but only marginally so. Kullback

Leibler from prior to posterior indicates performance up to a point.

`./illustrations/scaling-kld.tex`

PolyChord may converge faster for a stronger bias: e.g. if the prior is sharply peaked at the origin. In that case $\mathcal{D}$ is larger but run-time is smaller.



.

3.4.2 DONE Install Cosmochord.

CLOCK: *[2020-04-18 Sat 13:04]–[2020-04-18 Sat 14:04] (1:00)*
Need external Planck likelihood.

3.4.3 DONE Install on Cluster

CLOCK: *[2020-04-18 Sat 14:09]–[2020-04-18 Sat 20:25] (6:16)*
Planck is typical academic abandonware. They have a "python" script that installs the dependencies called `waf`. In their wizdom, they decided that they can do a better job than either pypi, or conda. They wrote their own package manager that downloads an outdated dependency if it can't find it (which it can't because if you think and it can't. Because if you think that writing your own package manager is a good idea, you're really too stupid to

do it properly. find it, because people who do pip and conda, don't abandon their software and move on.

As this may be read by someone who's writing academic code; the proper procedure is telling the end-user that they need packages x, y and z. If you're writing a package and it needs a package manager, and you're not thinking of maintaining it – Don't write a package manager! Mainly because I don't think you'd bother to check the venv, and update the download links. Also, you're probably thinking that either pip or conda can't do what you need them to do… So you don't ask.

This is at least six hours of wasted time, that could have been avoided if people were not in the sweet-spot of writing difficult to replicate, but at the same time completely unmaintained and under-developed code. This is why I insist on **proper** software engineering techniques!

Also! Keep Science away from Python. It's design philosophy 'we're all consenting adults', basically means that people do however, whatever and there's no responsibility. Python is a unique language: it has a few surface-level good ideas, a bunch of terrible ones, and is moderated by people that clearly have no idea what they're doing. That latter point is especially painful, as for some reason they thought that creating a breaking change to Python 3 was a good idea, yet they were unable to phase out python2, and after all of that, they thought that keeping the names of packages the same was not going to backfire.

The bigger problem is that people are seduced by the mild syntactic sugar of python. Big projects are dragged down by the cesspool of moronic design. It's giving me nightmares at this point.

The project report guideline asks us to think of ways that could make this easier. My suggestions. Don't use python. Don't! All the time you save by avoiding the curly braces is being paid for in sleep-deprived tortured people that just want to get their degree done with. I'm fed up with Cobaya's overengineered nonsense. I wanted to use CosmoChord to avoid Python at all costs. It doesn't work. Because Python is used there as well. It's 2am. I'm trying to figure out why pip swears that astropy is installed. While at the same time no version of python seems to detect it. If that wasn't enough; there's 17 version of pip on the cluster. I don't know which one to use to install that damn thing! I shouldn't have to know!

ANd the worst part is, people assume that writing python code is easy, because of all the light-weight stuff. If I told someone that I was stuck for an entire 24 hours trying to fix a build on a cluster people would say "how hard can it be". Just pip install? Right?

Eventual workaround language=Python,label= ,caption= ,captionpos=b,numbers=none module load python@3.8 sudo /usr/local/software/master/python/3.8/bin/python -m ensurepip module purge module load python /usr/local/bin/python -m ensurepip /usr/local/bin/python -m pip install astropy cython Notice that I had to use a python3 pip to install a python2 pip. Naturally this is just stupidity.

3.4.4 DONE Have a more thorough discussion of the potential next-gen sampler.

CLOCK: *[2020-04-20 Mon 11:12]–[2020-04-20 Mon 12:12] (1:00)*
Use dynamic linking, refactor current code to allow multiple prior specifications. Fix the current overloading. Re-use as much of PolyChord's fortran code as possible. Everyone tries to do unbounded priors; but you can always do that using a bounded uniform.

3.4.5 DONE Rename all references to SSPR to SSPM *<2020-04-20 Mon>*

Posterior re-partitioning implies that it has something in common with Hobson-Feroz-CHen algorithm. It does **not**.

It's *much* more general, in that it can deal with multiple models! It can mix the priors completely, and choose the more representative model!

This is not posterior re-partitioning, this is superpositional prior mixing. It just so happens that we're re-partitioning the posterior in every model.

A more appropriate name for this is Stochastic Superpositional prior mixing. More accurately model mixing, but this is with the intent to mix different priors.

3.4.6 DONE describe model chaining! *<2020-04-21 Tue>*

See `./project-report.tex`.

3.4.7 TODO Produce CosmoChord output

CLOCK: *[2020-04-22 Wed 09:34]–[2020-04-23 Thu 15:34] (30:00)*
CosmoChord works well as soon as you replace `action=5` with `action=1`, in `test_planck.ini`.

4 Easter Term.

4.0.1 TODO prepare for the presentation.

CLOCK: *[2020-04-23 Thu 15:43]–[2020-04-23 Thu 17:00] (1:17)*
`./presentation.tex`

4.0.2 TODO Give presentation.

4.0.3 DONE extract parameter covariances.

CLOCK: *[2020-04-23 Thu 13:42]–[2020-04-23 Thu 15:42] (2:00)*

Coabaya's native output is not suitable for `anesthetic`. Use GetDist instead.

Outputs generated and saved in plain text for later use.

1. **DONE** e-mail Dr Handley about using HDF. This could reduce the memory usage significantly.

4.0.4 TODO Implement SSPM in Cobaya.

5 Notes

5.1 Michaelmas Term

Doing some research about the subject.

5.1.1 Terminology

Prior - $\pi(\theta) = P(\theta|M)$ Likelihood - $\mathcal{L}(\theta) = P(M, \theta|Data)$

Posterior - $\mathcal{P}(\theta) = P(\theta|\text{Data})$ Evidence - $\mathcal{Z} = P(Model|Data)$

Bayes' theorem says that

$$Likelihood \times Prior = Posterior \times Evidence$$

So can use this to find the parameter values of a model, + the likelihood that the model fits the data at all.

5.1.2 How does nested sampling work

1. Skilling's paper

   cite:skilling2006

Nested Sampling is a machine learning technique that allows to do Bayesian parameter estimation.

Fitting a line to data is an example of a parameter fitting model.

Set

$$\chi^2 \triangleq \sum_i \left(\frac{y_i - f(x_i, \theta)}{\sigma_i} \right).$$

We need to ask a question, how likely is the data observed, given that the model is true, and the Model parameters have the given values. The probability is usually given by a Gaussian (or normal distribution).

$$L = \frac{1}{N} \exp\left[-\chi^2\right]$$

So what we need to do for Nested Sampling to work, is to provide a model for estimating the fit to the hypothesis - likelihood, and a prior.

The likelihood, or how likely is the value of data given the model and the parameters, reflects how we expect the fluctuations to develop. Many distributions are possible, but due to the Central Limit theorem, best choice would be a Normal (Gaussian) distribution.

The prior represents our prior knowledge of the original parameters. For example, if we know nothing about the possible model parameters, we can expect a uniform distribution within constraints. These constraints may be artificial (for example, we may only be interested in model parameters that are within machine-representable floating point numbers), or natural (the Hubble parameter is positive).

If we know more about the model parameters, that information can also be presented as a guideline for parameter inference. For example if we have done parameter estimation of the same model, using a different set of data; the posterior of the aforementioned investigation can be used directly as the prior for this run.

Nested sampling exploits that extra data to converge upon the so-called typical set; which represents the data that has statistically significant phase volume. The latter point can be understood intuitively.

More accurate or tight constraints on the true data should lead to better convergence time. Ideally the convergence to the posterior of a distribution is the fastest, as this minimises the number of errors, and given a suitable sampling algorithm should lead to few wasted computations.

   (a) **TODO** Phase volume example.

2. Notes on the Algorithm itself:

   Rasterising the phase space is too computationally ineffective, as for a model with 27 parameters, the space would be 27 dimensional. This leads to many quirks of geometry and counter-intuitive outcomes, that will be touched on later.

   We must first select a number of live points randomly from the phase space, usually taken to a be a hypercube with edge length normalised to 1.

   For each point one expects there to be a locus of points with the exact same likelihood. This locus is often connected, and so in analogy with isotherms it is often referred to as the iso-likelihood contour.

   Then one selects the least likely point and picks according to some algorithm, a point of higher likelihood. The original point is now referred to as dead, while the new point is added to the set of live points.

   This process is then repeated until we have reached a typical set. This is often determined by estimating the *evidence* contained outside each contour (since the points are picked at random, if we have n_{live} points, each contour will statistically include $\frac{1}{n_{\text{live}}}$ of the total phase volume).

3. Piecewise power repartitioning notes. Are these issues you're encountering for the mixture model, or the temperature-dependent gaussian? (in the posterior repartitioning paper, the pi^β prior is terms 'power posterior repartitioning', so we should refer to it as that).

   For the power posterior repartitioning, remember we're doing it with a diagonal prior covariance, so everything is separable and Z(beta) should be derived as described in the posterior repartitioning paper, namely:

   $\tilde{\pi} = G[\mu, \sigma](\theta)^{\beta}/Z(\beta)$

   $$Z(\beta) = \int_a^b G[\mu, \sigma](\theta)^{\beta} d\theta$$

   . where G denotes a gaussian, and a and b are the limits of the uniform distribution. This is expressible using erf:

   $$Z(\beta) = \frac{erf(\frac{(b-\mu)}{\sqrt{2}}\sigma) - erf(\frac{(a-\mu)}{\sqrt{2}}\sigma)}{2} \tag{2}$$

I've spent a bit of time thinking this morning, and have realised that the mixture model is not quite as trivial as I had imagined.

To be clear, working in 1d for now, our normalised modified prior is of the form:

$$\tilde{\pi}(\theta) = \beta U[a,b](\theta) + (1-\beta)G[mu, sigma](\theta)$$

where there will be a,b,μ,σ for each dimension. To compute the prior transformation which maps x$\in$[0,1] onto θ, nominally we should do this via the inverse of the cdf:

$$F(\theta) = \frac{\beta(\theta-a)}{(b-a)} + (1-\beta)\frac{1}{2}\frac{1+erf(\theta-\mu)}{\sqrt{2}\sigma} \tag{3}$$

Unfortunately $x = F(\theta)$ is not invertible. There is another way around mapping $x \in [0,1]$.

In general, if you have a mixture of normalised priors:

$$\pi(\theta) = \sum_i A_i f_i(\theta)$$

$$\sum_i A_i = 1$$

where each f_i has an inverse CDF of $\theta = F_i^{-1}(x)$

one can define a piecewise mapping from $x \in [0,1]$ thus:

$\theta = F_i^{-1}\left(\frac{x-\alpha_{i-1}}{A_i}\right) : \alpha_{i-1} < x < \alpha_i$

$$\alpha_i = \sum_j^i A_j$$

Basically this uses x to first choose which component of the mixture is active (via the piecewise bit), and then rescales the portion of x within that mixture to [0,1].

This method seems a little hacky at first, but the more I think about it the more reasonable it seems. I would be interested to hear your opinion, and we can discuss on Wednesday morning. Until then, practically you should focus on the diagonal PPR approach, as that is much more straightforward, and captures the essence of the method.

4. Data and Parameter covariance matrices.

   To avoid having to generate data with a given distribution, we can simply and directly use the Parameter covariance matrix, for our toy models.

   This basically means that instead of using the model's functional form, we directly assume that the distribution is of Gaussian nature. This we simply plug into the log likelihood, and the rest of the algorithm proceeds as if we had data and the functional form, and the chi^2 computation was done for free.

   (a) Correlated vs Uncorrelated Gaussian log likelihoods

      If the parameter covariance matrix is completely diagonal, then the parameters are each individually Gaussian distributed, with a standard deviation being the diagonal element.

      An arbitrary coupling can lead to covariance on the off-diagonal. These mixtures can be unmixed by using either Singular Value or eigenvalue decomposition of the covariance matrix. This can be simply regarded as a coordinate transform, a passive one at that. Consequently, a Gaussian distribution in Loglikelihood takes the following form.

      Let $\vec{\mu}$ be the vector of mean values of Gaussian distributed parameters $\vec{\theta}$ (we shall drop the vector). The corresponding parameter covariance matrix is $G_{i,j}$.

      Therefore the corresponding loglilkelihood is

$$\ln \mathcal{L} = -N - (\theta - \mu)^T G^{-1} (\theta - \mu)$$

      , where the normalisation constant is given by

$$N = \frac{\det |2\pi G|}{2}$$

      .

5.2 Lent Term

5.2.1 Polychord *<2020-01-10 Fri>*

Polychord is a nested sampling program that uses directional slicing, which is not (citation needed) a form of rejection sampling.

To run polychord one needs to do three things:

5.2.2 `settings`

which needs the number of dimesnions with which we're working, (very procedural, probably a consequence of fortran-centric implementation).

The Settings have information about the verbosity of the system.

language=Python,label= ,caption= ,captionpos=b,numbers=none settings.feedback = 0 seems to be a good default.

Polychord can resume the older run, if instructed (rather by default), so in order to have clean bench-marking do

language=Python,label= ,caption= ,captionpos=b,numbers=none settings.read$_{r}esume = False$

To control running-time vs precision trade-off, use

language=Python,label= ,caption= ,captionpos=b,numbers=none settings.nlive=175

Changing it to a lower value makes the program run faster.

Another way to control termination is the

language=Python,label= ,caption= ,captionpos=b,numbers=none settings.precision$_{t}arget = 0.1$

But we should normally not tinker with it.

5.2.3 logLikelihood

This is essentially a χ^2, which represents the probability of our data, given the model and the parameter values.

We need to define it for each model. Ideally what it needs to return is the normalised probability, but not giving it the proper normalisation doesn't seem to affect the run-time, only the result.

5.2.4 Prior

This is a weird function. What this is called, probably has nothing to do with what it actually is: it's taking a point in a unit hypercube and maps it onto the real θ values.

This function is where we can get most of our performance uplift.

Ideally assuming that the *real* prior is the posterior the algorithm should converge the fastest. This should however affect the loglikelihood calls, because we're re-scaling the space.

I **think** that this simply means that the absolute value of the **loglikelihood** is **not a meaningful** number.

1. UPDATE: it is meaningful. Just not without the prior. *<2020-02-14 Fri>* AutoPR relies on

5.2.5 Why PolyChord converges faster for more informative priors.

Polychord stops as soon as the evidence remainig is a predetermined fraction of the original.

It's working in a unit hypercube, so we can assume that the evidence is normalised.

So the value oF language=Python,label= ,caption= ,captionpos=b,numbers=none settings.precision$_criterion = 0.01$ means stop when the evidence is less than one percent.

So if we give a more informative prior the algorithm is more efficient at finding all the evidence that it needs.

The prior basically says, draw from the region of higher likelihood more frequently. So you are more likely to draw more evidence in a shorter time.

It's not guaranteed to converge the fastest if the prior is the posterior, hence why Kullback Leibler divergence is not that important.

5.2.6 Kullback Leibler divergence.

KL-D is explained in the blog. It basically is the Shannon entropy between two distributions. It's

$$D = \log \langle \frac{P_1}{P_2} \rangle \tag{4}$$

so if the two distributions coincide $D = 0$, if they don't it's going to be slightly positive. And if the distribution is defined somewhere where the other one is zero it can be very negative.

As a result of this, I can come up with examples where the Kullback Leibler divergence is the same, byut the speed of convergence is different.

This is why it can't be used as the only performance metric.

Another issue is that it assumes that everything goes well. What if repartitioning casues many fasle positives to be drawn from the distribution and get rejected. This will cause a slowdown but will not affect D.

5.2.7 Approaches to modelling systems. *<2020-01-17 Fri>*

One way to model all of our systems is by looking at the χ^2 and dealing with generated data. While this is close to what the system might actually do, this is not itself a good solution, it's slow and it requires extra computations in generating the data with the properties that we need.

Instead we might simply treat the system as if it was already diagonalised in the model parameter space. So if our model has

$$\mu = \begin{pmatrix} \mu_0 \\ \mu_1 \\ \vdots \\ \mu_n \end{pmatrix} \tag{5}$$

and

$$G = \begin{pmatrix} \sigma_1^2 & \sigma_{12}^2 & \cdots & \sigma_{1n}^2 \\ \vdots & \ddots & \vdots & \vdots \\ \sigma_{n1}^2 & \sigma_{n2}^2 & \cdots & \sigma_n^2 \end{pmatrix} \tag{6}$$

which is itself a gaussian assumption, we get the following: as our log-likelihood

$$\ln \mathcal{L} = -N - (\theta - \mu)^T G^{-1} (\theta - \mu) \tag{7}$$

where N can be found by integrating a multivariate Gaussian. See hand-out for Phase Transitions:

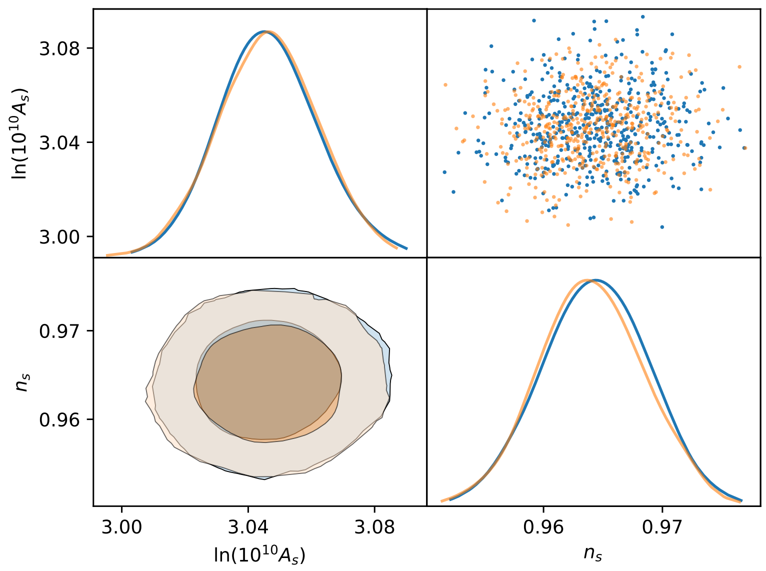
$$N = \ln \det |2\pi G| \tag{8}$$

this can be evaluated in one fell swoop using
language=Python,label= ,caption= ,captionpos=b,numbers=none numpy.linalg.slogdet(2*pi*G)

This allows us to do calculations in a fraction of the time.

5.2.8 Comparison of runs. *<2020-01-24 Fri>*

Planck data can be downloaded from (see references), and using the following constraints, we can compute the misfit between data.



This shows profound agreement, usingThe following constraints on the parameters.

language=Python,label= ,caption= ,captionpos=b,numbers=none planck$_range s$ = $numpy.array([[0.019, 0.025], [0.095, 0.145], [1.03, 1.05], [0.01, 0.4], [2.5, 3.7], [0.885, 1.04], [0.9, 1.1], [0, 200]$

samples = anesthetic.NestedSamples(root='./data.1908.09139/lcdm/chains/planck') fig, ax = samples.plot$_2d(['logA', 'ns'])plt.show()$

params = samples.columns[:27] params = samples.columns[:27] Sig = samples[params].cov().values mu = samples[params].mean().values nDims = len(mu)

Run of the original

args = 'root$_name' :' planck',' m' : mu,' s' : Sig,' likelihood' : lambdax :$ $gaussian_likelihood(x, mu, Sig),' renew_plots' : True,' renew_plots' : False,' nLive' :$ $2,' prior' : lambdax : uniform_prior_with_ranges(x, planck_ranges),' ax' : axexec_polychord(** args)$

... newSamples = anesthetic.NestedSamples(root='./chains/planck') newSamples.plot$_2d(ax)$

plt.show() fig = plt.figure()

5.2.9 Automated Power Posterior Repartitioning. *<2020-01-24 Fri>*

Looking at `https://arxiv.org/pdf/1908.04655.pdf` we can see that one can get better convergence if we use something called the Automated posterior repartitioning.

We start with a Gaussian prior.

$$\pi(\theta) = G(\mu, \sigma)(\theta) \tag{9}$$

We then introduce an extra parameter into our system:

$$\tilde{\theta} = \begin{pmatrix} \theta_1 \\ \downarrow \\ \theta_n \\ \beta \end{pmatrix} \tag{10}$$

We then use this parameter to rescale the prior that we originally had:

$$\tilde{\pi}(\tilde{\theta}) = \frac{G(\mu, \sigma)(\theta)^{\beta}}{Z_{\pi}(\beta)} \tag{11}$$

And normalise it to one Having done that we need to keep the posterior the same, so we need to rescale (citation needed) the loglikelihood to account for this change.

5.2.10 Calculations neeeded for doing PPR.

As a result, to use it with PolyChord we need to do the following three things.

1. Estimate the quantile (called prior in PC).
2. Estimate the correction needed to the Gaussian likelihood.
3. Make sure that the correct dimesnionality is used.
4. Maybe think of the prior on β.

To estimate the quantile we need to know the CDF:

$$\int_{-\inf}^{\theta} \pi(\theta') d\theta' = \Pi(\theta) \tag{12}$$

Then we need to invert it.

$$quantile(x) = \Pi^{-1}(x) \tag{13}$$

So if it's a truncated uniform distribution:

$$\Pi(\theta) = m\theta + b \tag{14}$$

Where

$$m = \theta^m - \theta_m \tag{15}$$

and

$$b = \theta_m \tag{16}$$

if it's a truncated Gaussian$^\beta$, then

$$quantile = \mu + \frac{\sqrt{2}\sigma}{\sqrt{\beta}} erf^{-1}\left(xerf\left[\sqrt{\frac{\beta}{2}}\frac{b-\mu}{\sigma}\right] + (1-x)erf\left[\sqrt{\frac{\beta}{2}}\frac{a-\mu}{\sigma}\right]\right) \tag{17}$$

(Dr Handley missed the prefactor of $\sqrt{2}/\beta$, which caused headaches).

Sanity check: as $a \to -\infty$ $b \to \infty$,

$$quantile \to \mu + \frac{1}{2\pi\sigma}^{\beta} \tag{18}$$

Which is what we expect from Poisson integrals.

5.2.11 When and why do repartitioning. *<2020-02-19 Wed>*

After multiple experiments I arrived at the following.

Running PolyChord with a Gaussian is **not the same** as what we want to accomplish.

When we consider two different priors, a Uniform from $(a, b)^{\otimes} n$ and a Gaussian given by an $n \times n$ matrix σ.

The histogram of the loglikelihood calls and their results will be different, it will differ due to the *effective volume* which each of those will occupy.

Naturally a Gaussian, even if it has the same effective volume cannot simultaneously give the same evidence and the same Posterior.

So we can ask two kinds of questions:

a) What is the Posterior given the prior that has a different volume.

In this case the loglikelihood calls will cluster around different values. This is a legitimate question to ask, but it requires more information.

If we take a system where we don't have the extra information about the location of the posterior peaks and their shape, and plug in a prior that does, we're biasing the system, and effectively *fudging* the answer. This can in some cases be useful, but it's not quite what the project aims to do.

So instead of asking what would our answer be with a different prior, we ask a different question. What would our answer be if our system was biased to picke the values that *suspect* are the correct posterior values, without that affecting the posterior distribution and injecting extra information that we don't have/ can't quantify.

This is a philosophical issue. Intuitively, if we have the extra information it *must* be reflected in the prior. It can't otherwise. In fact by biasing the system, even if we repartition the combination $\mathcal{L}\pi$ we can still end up with a biased and therefore useless result.

In fact, my experiments clearly show this; if the Prior corresponding to *a* Gaussian which is not the same as the posterior (has a different value of the mean), it can result in the algorithm terminating and generating a completely false posterior.

See `./toy-models/1/1.0Exampleofparametercovariance.py/`.

By doing repartitioning we allow our guess to be wrong, without that affecting the outcomes: posterior and evidence.

5.2.12 Correlated and Uncorrelated Gaussian:*<2020-02-03 Mon>*

Knowing that the parameter covariance matrix, is usually positive defininte, one can argue that the question of whether or not the parameters in the model are correlated, or completely uncorrelated (i.e. each has a single standard deviation value) is moot.

We can always perform a linear operation that diagonalises the parameter covariance matrix, and what the algorithm needs to do is only to work with the uncorrelated Gaussians.

That of course is true, but some repartitioning schemes are more sensitive to this fact, and can only work after the coordinate transformation has been performed, which itself adds to the complexity.

Other algorithms are more capable of doing this properly. SSPR, what we've developed doesn't require any knowledge of the underlying distribution. So we don't really care if it's correlated or uncorrelated.

5.2.13 Stochastic supepositional re-partitioning.

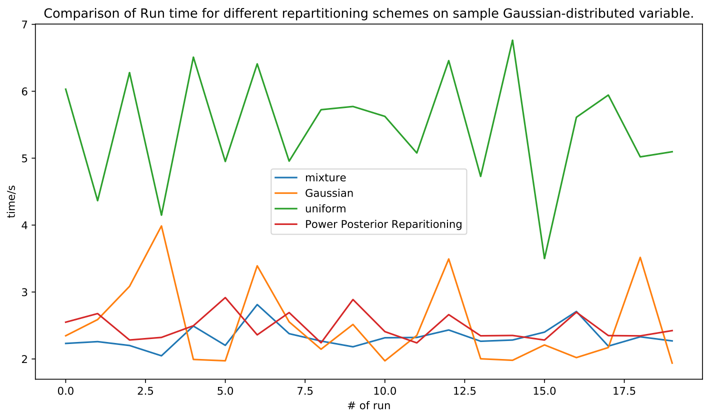
We take a mixture of re-partitioning schemes. Instead of saying we invert the sum of them partially, like Dr Handley suggested (call it additive re-partitioning), we may try to do everything stochastically.

That is, we point wise set $\tilde{\pi}\tilde{\mathcal{L}}$ to be constant. But at each point $\tilde{\pi}(\theta)$ may be different. This needs to be done deterministically, because PolyChord revisits the same points and complains about the non-determinism.

We can achieve the determininsm by usgin a PRNG with a deterministic seed. I would think that this is just a Hash function? (edit: no it's not enough, because Python's hash function is linearly distributed. This **can** work, but it will only mix in one half of the domain. It's going to converge, but may cause errors if we have doubled peaks etc).

The relevant code for this is in the `./framework/general_mixture_model.py`. This uses my implementation of the broken stick probabity to choose which bin to deterministically put a point with coordinates $\vec{\theta}$ in. The probability of being put in each bin is **not** β, but is related to it. Why? Because β vary from 0 to 1 independently. So if $\beta = (1, 1, \ldots, 1)$ then the probability is simply $\frac{1}{m}$ where m is the number of models in the mixture.

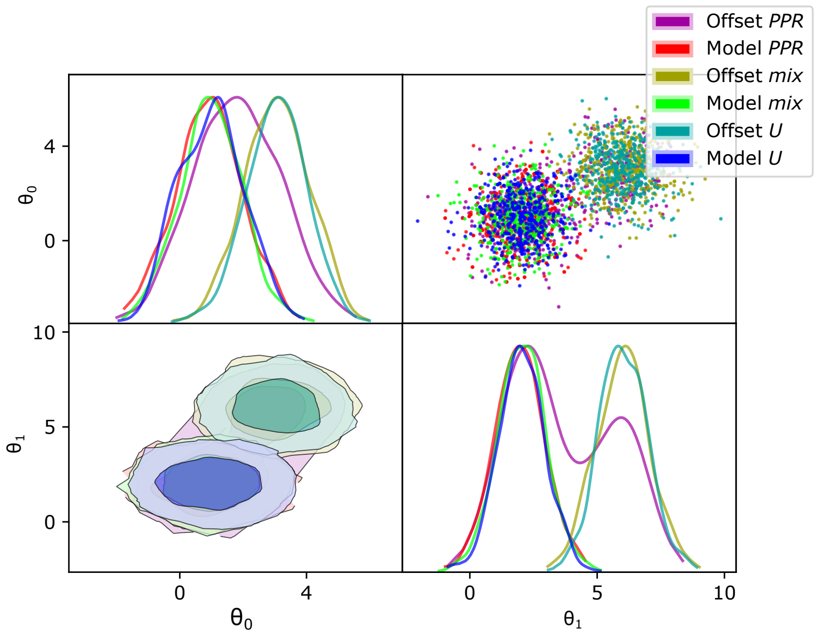
This is much faster:



. I don't know why yet, but it is.

edit: it's because the bias is either full, or non-extant. With PPR or an additive mixtue the bias is proportional to the value of β. In SSPR, for each point, the bias is either the same is it would have been for a gaussian (in which case values of β, favouring the representative prior, making it skyrocket) or is basically flat. This may suggest that it's also more robust.

edit: it is more robust.



. It can make PPR more robust too. If it's not a panacaea, it's pretty damn close to being perfect based on being both faster, and more reliable.

edit: `[[https://github.com/ejhigson/nestcheck][nestcheck]]` shows that it's more precise, there's a lot less variance in evidence, in each of the parameters. It does occasionally produce `inf` s and `nan` s. I need to figure out why.

edit: It can also mitigate the miscalculaterd evidence.

5.2.14 Mixing non-repartitionned priors.

What would happen if we mixed prios that didn't have the same posterior? What if we genuinely had a prior starting with a uniform, and another one that was genuinely Gaussian?

I think that Stochastic Superpositional mixtures can deal with both. They can retroactively adjust the conclusions we reached earlier, and produce a better result in that case. It can compare two datasets.

This is not an optimisation to nested sampling. This is a new algorithm! This is Bayesian metainferece. It can look at the entire pre-history of the data. We can dynamically adjust the priors!

5.2.15 Purpose built sampler to facilitate stochastic mixing.

1. Quantum computers. Surprisingly, this algorithm maps onto quantum computers quite easily (ask Prof Mike Payne, and Prof. Crispin Barnes). The thing is, we need to explore both branches simultaneously, and use a qubit to represent the probability of belonging to one sub-space or the other. So if mixing m models, need m qubits. This superposition is why this is so powerful. We can make this algorithm even faster, because then there would be no extra parameter, and we're exploring the entire space.

2. Modifying PolyChord.

   May be the easiest. I've built the new API front-end in the ./framework/ folder. You just need to provide an implementation of the prior inversion function (edit quantile), and the log-likelihood. Then set the dimensionality (parameter covariance models do it automagically).

   So as a result, you can provide building blocks: models related by inheritance. (surprisingly, there are cases where inheritance is better than composition, since we're using Python, we might as well write it object-oriented).

   These models can be mixed. Arbitrarily. I can mix as many as I wish, and I can build them up using the provided corrections: i.e. if I have a Gaussian likelihood, that's related to data in a complicated way, I can just add the necessary corrections and it will work: e.g.

   language=Python,label= ,caption= ,captionpos=b,numbers=none def loglike(self, data, beta): ll = self.complicated$_{l}og_{l}ike(data)ll+ = gaussian_{m}odels.ppr_{c}orrection(ll,$

   Meaning, that very little adjustment to existing workflows is needed.

3. Better API for PolyChord.

   Currently PolyChord has the following problems.

   - The settings variable can be written to but not read. language=Python,label= ,caption= ,captionpos=b,numbers=none settings.file$_{r}oot$ gives no such attribute error.
   - The settings are non-trivial. Can't run PolyChord without settings, so need to make the settings part of either a PolyChord object or a model object.
   - MPI. It caused about seven bugs in my code. Because module loading should not have side-effects.

- The code may be functional. There are many cases where Currying would have saved **lots** of typing.
- Maybe re-write in Haskell? Rust?

4. Next generation sampler. Motivation: mixture re-partitioning can hugely benefit from having low-level access to the sampler's internal state. For example:

    - We may want to chain posteriors from different models as priors to others.
    - This may allow us to gain the speed-ups up to and beyond what we've shown! We can, as a person would, do a crude estimate, and use that to do finer adjustments.
    - We can do **Bayesian meta-analysis**. Imagine that you can re-run all of the inferences that were run up to date, and you can also require that the data are consistent? This **can** be done!
    - Sampling in $\vec{\beta}$ and the branch index need not use the same technique. In fact, Metropolis Hastings for β, and slice sampling for all other parameters, may lead to better convergence.
    - Want to eliminate cross-talk between re-partitioning parameters for things in the mixture. For example, if I'm mixing two PPR's I don't want for both of their powers to be the same and sampled the same way.
    - The branch index is an integer and not a nuisance parameter. It was chosen so that the hashing doesn't cause undefined behaviour if Python's "smart" numerical routines and duck-typing decides to duck with SSPR. Ideally we want for the chosen branch to be global to both the prior and likelihood scope.
    - The slice sampler needs to be aware which branch do points belong to. Ideally, it can estimate the likelihood in both branches by evaluating the common likelihood, and then comparing the corrections.

    We can also learn from `PolyChord` 's mistakes. As of now, it only supports bounded priors and even then, only on spaces that can be mapped onto from a unit hypercube. It has confusing nomenclature. The prior is not a prior: it's not a PDF, but a specification of the prior via its quantile.

The bridge between Python and Fortran is unreliable. You can pass a python callable, that's not being compiled to machine code into Fortran. This is wrong. This is why the exception handling facilities of Python and the static type checking of Fortran are both made toothless.

The Fortran code (for performance reasons) always does unchecked assignments, while `Python` inherently assumes that all assignments are checked. A lot of the optimisations come in at compile time. In fact, almost all of the functions are pure. Yet Python doesn't support functional optimisations. Fortran does support them, but only if all routines have no side-effects, which current logging forbids.

Polychord is very obtuse about logging. Its messages cannot be switched off. Control over verbosity and location of the produced data needs to be made more transparent. I understand that the copyright notice is the wish of the authors. That said, the program reproduces these not on module load, but every time its run! Making several runs pollute the interpreter `stdout`.

PolyChord cannot be used in-memory. A couple of my benchmarks, since they deal with low-dimensional models were held back by the file-system input output latency. Moving them into a `ramdisk` is a solution, but not an idiomatic one. Since PolyChord has easily definable inputs it should be functional.

As a result I suggest the following improvements:

- Proper error handling.
- An interface that **requires** a compiled likelihood and dynamically links against it.
- A memory-safe language with functional tendencies (Julia, Rust, C)

5.2.16 Financial situation, equal opportunities and inflexibility. ARCHIVE