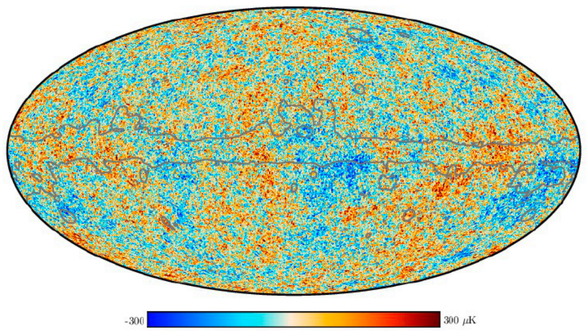
Cosmological Inference and Parameter Estimation - I

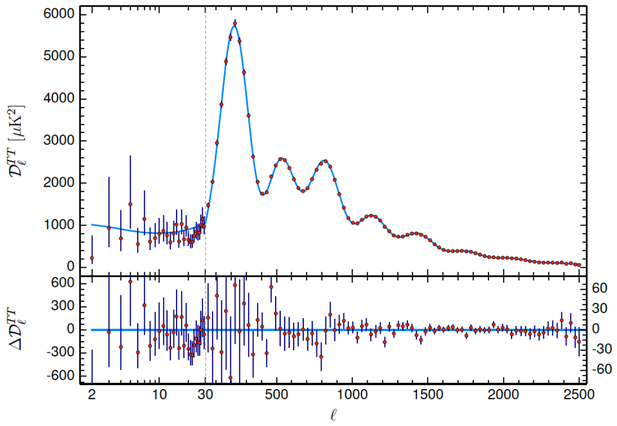
Introduction to Markov Chain Monte Carlo

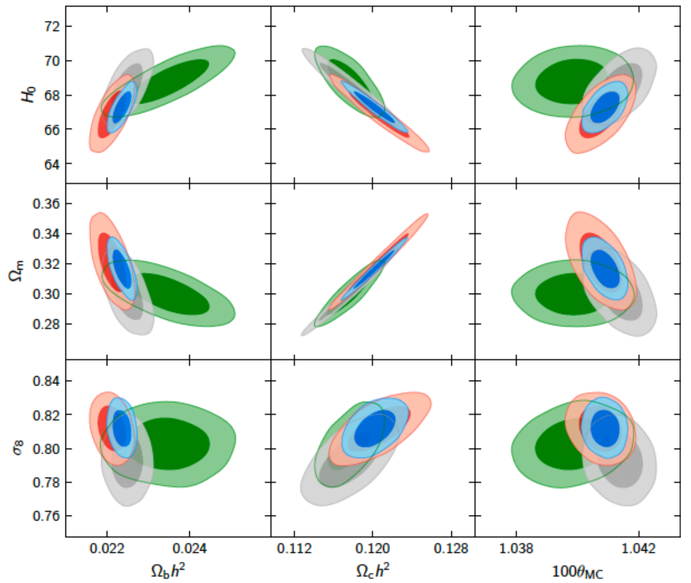
Why do we need statistics?

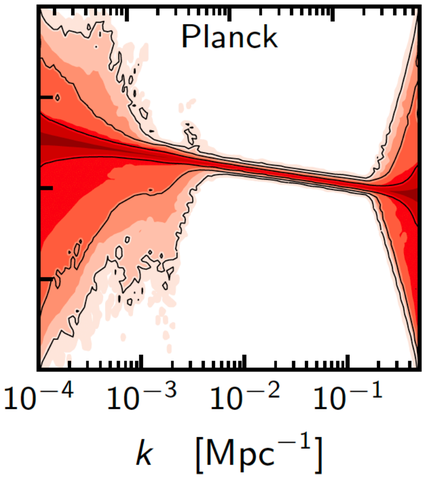
- Parameter estimation (e.g. what are the values of $H_0, \Omega_m, \Omega_\Lambda \cdots$?)
- Testing a hypothesis (e.g. did NANOGrav detect SGWB?)
- Model comparison (e.g. GR vs Modified GR?)
- Forecasting (e.g. how well can LISA determine H_0?)

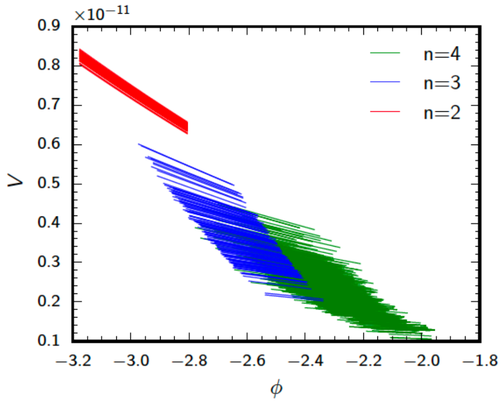
Statistical Inference











Planck 2018 results

Bayesians vs Frequentists

Probability related to our own knowledge of events, ‘a degree of belief in a hypothesis’

Try to obtain the probability distribution of the model parameters given the data — **posterior**

$$P(\vec{\theta}\,|\,\vec{d})$$

Probabilities related to frequency of occurrence of an event,

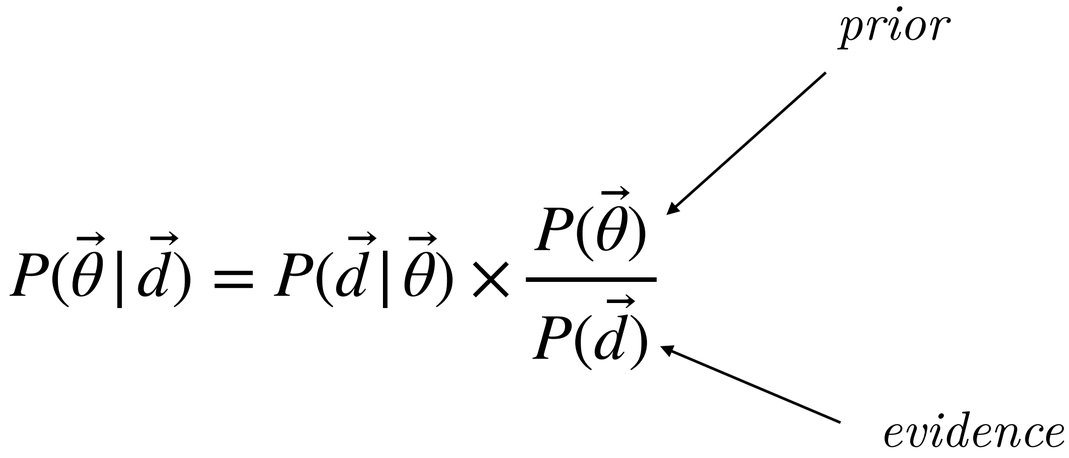
$$P = \lim_{N_{\text{trials}} \to \infty} \frac{N_{\text{event}}}{N_{\text{trials}}}$$

Use the **likelihood**, which is the probability of getting the data given the model parameters and try to find best-fit point

$$P(\vec{d}\,|\,\vec{\theta})$$

Bayes Theorem

Posterior and Likelihood related via Bayes' theorem



Forget about evidence for now

Bayesian vs Frequentist inference

Computationally more intensive but also more powerful

Full parameter distribution vs maximum likelihood for model selection

Frequentist methods are based on asymptotic properties of estimators, no such limitation for Bayesian methods

Easy to implement physical priors (e.g. $A^2 > 0$) but results are prior dependent

Trivial to marginalise over uninteresting parameters (e.g. noise)

Some reasons why Bayesian methods are preferred, but it does not mean one is ‘correct’ and other ‘incorrect’…

From now on, we are all Bayesians

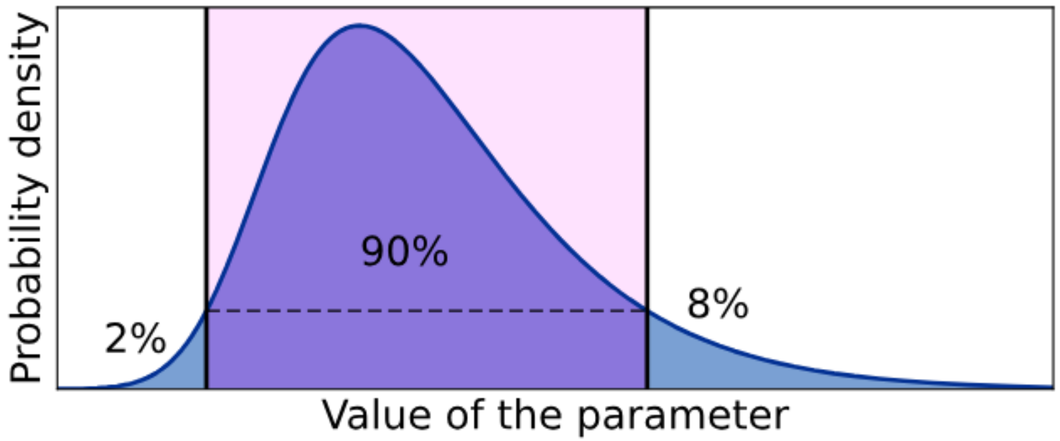
Reverend Thomas Bayes
(1702-1761)

Bayesian Inference

We are interested in the parameter posterior $P(\vec{\theta}\,|\,D)$

Sufficient to get $P(d\,|\,\theta)P(\theta)$, related to the posterior by a normalising constant

Use to construct credible intervals, marginalised distributions etc…



$$P(\vec{\theta}_1\,|\,D) = \int d\theta_2\, P(\vec{\theta}\,|\,D)$$

Integrating out θ_2 (e.g. noise parameters)

Bayesian Inference

The recipe

Specify model in terms of its parameters $\vec{\theta}$

Choose priors on $\vec{\theta}$

Define the likelihood function

Obtain the posterior distribution, **but how?**

Parameter Grid

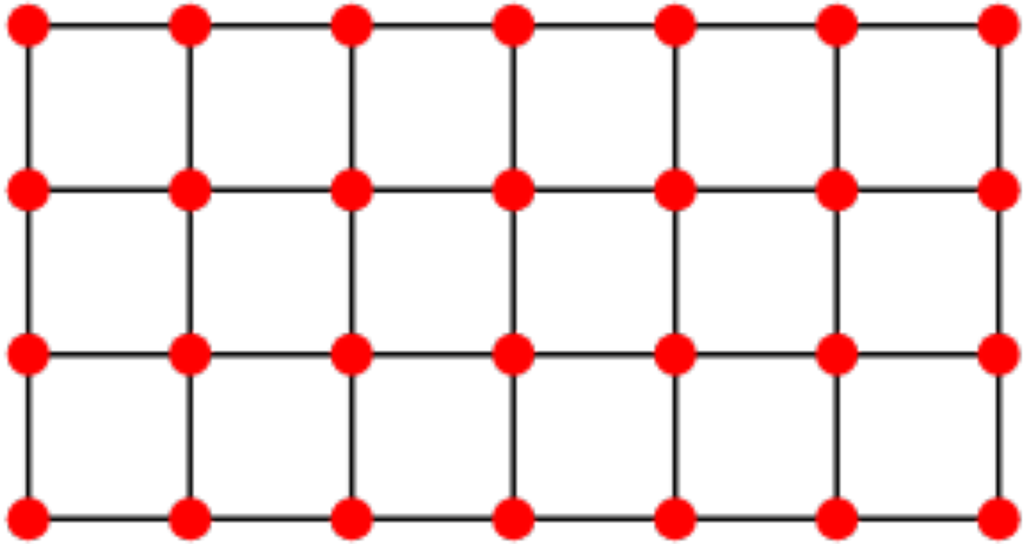
N points per dimension

N^d total number of points

E.g. 10 points per dimension (coarse grid)

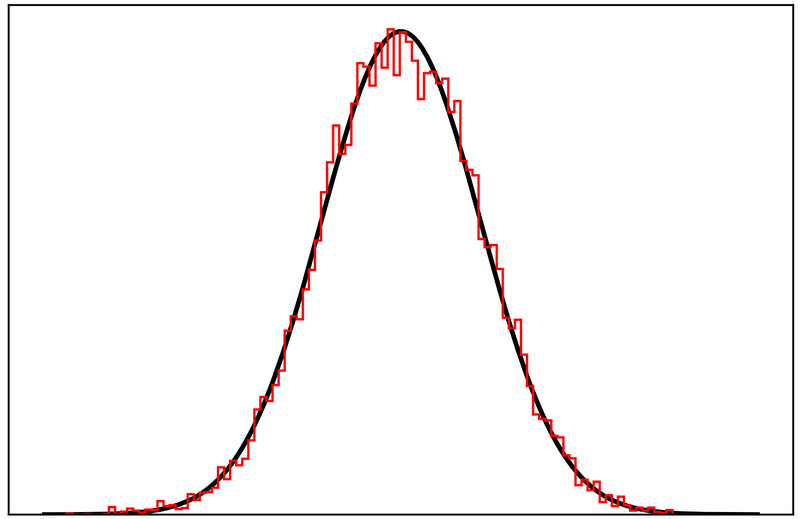
If 1s per evaluation, then for $d = 6$, need 10^6 evals ~ 12 days 💀

Evaluate $P(\theta \mid D)$ on grid?



Sampling Distributions

$$P(\theta|D) \simeq \frac{1}{N_{\rm s}} \sum_{i=1}^{N_{\rm s}} \delta(\theta - \theta_i)$$



e.g Estimating the mean from samples

$$\langle \theta_1 \rangle = \int d\theta_1 \, \theta_1 P(\vec{\theta}|D) \sim \frac{1}{N_{\rm s}} \sum_{i=1}^{N_{\rm s}} \theta_{1,i}$$

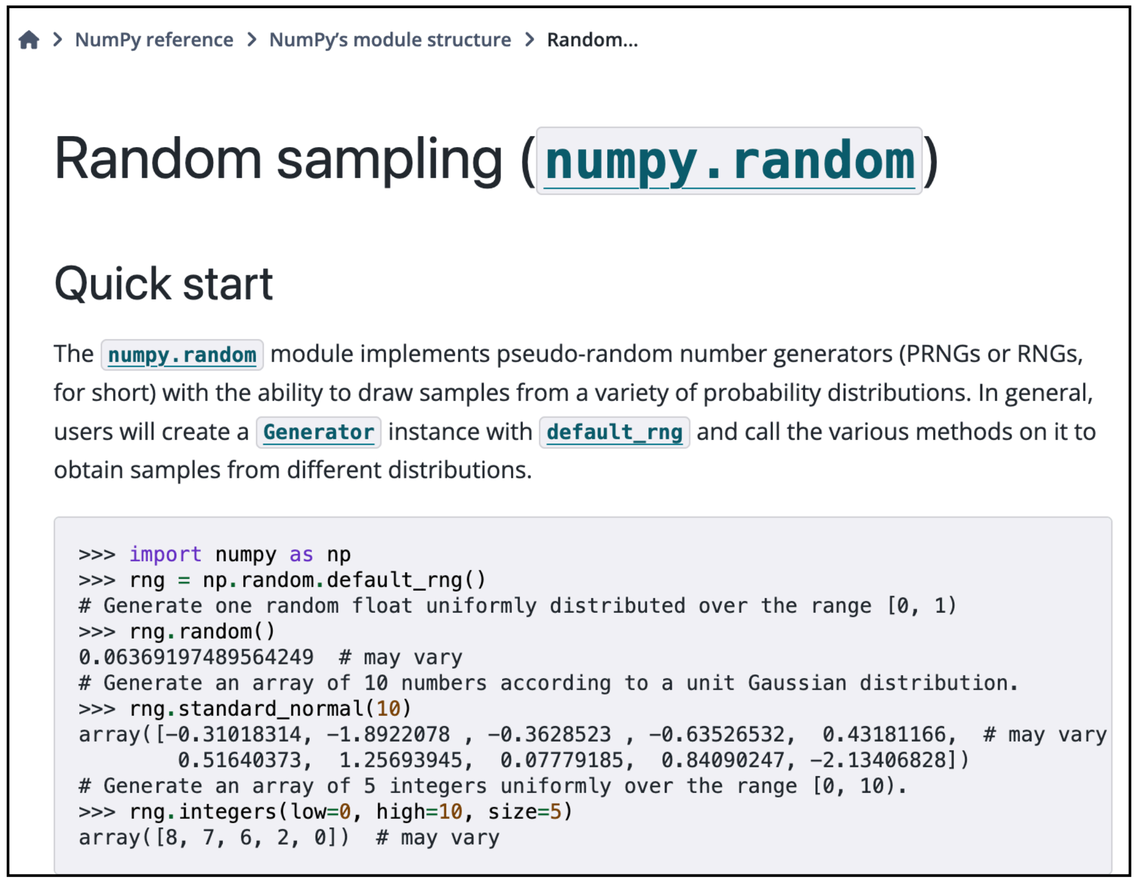
Samples drawn from the distribution

By Central Limit Theorem, this estimate tends to $\mathcal{N}(\langle \theta_1 \rangle, \sigma_1^2/N_s)$ for $N_s \gg 1$

Simple Distributions

Can analytically integrate over uniform or Gaussian distributions

Methods to directly sample from these distributions also exist — pseudorandom number generation
e.g. uniform distribution



NumPy reference › NumPy's module structure › Random...

Random sampling (`numpy.random`)

Quick start

The `numpy.random` module implements pseudo-random number generators (PRNGs or RNGs, for short) with the ability to draw samples from a variety of probability distributions. In general, users will create a `Generator` instance with `default_rng` and call the various methods on it to obtain samples from different distributions.

```
>>> import numpy as np
>>> rng = np.random.default_rng()
# Generate one random float uniformly distributed over the range [0, 1)
>>> rng.random()
0.06369197489564249  # may vary
# Generate an array of 10 numbers according to a unit Gaussian distribution.
>>> rng.standard_normal(10)
array([-0.31018314, -1.8922078 , -0.3628523 , -0.63526532,  0.43181166,  # may vary
        0.51640373,  1.25693945,  0.07779185,  0.84090247, -2.13406828])
# Generate an array of 5 integers uniformly over the range [0, 10).
>>> rng.integers(low=0, high=10, size=5)
array([8, 7, 6, 2, 0])  # may vary
```

Direct Sampling

We can sample from P using Q, $x \in Q$ can be converted to $\theta \in P$ by solving $Q(\theta) = x$

A simple example: $P(\theta) = e^{-\theta/2}$ has $Q(\theta) = 1 - e^{-\theta/2}$

$$Q(\theta) = \int_0^\theta d\theta' \, P(\theta')$$

CDF $\in [0,1]$

PDF

Parameter transformations can also help e.g.
the Box-Muller transform takes uniform x_1, x_2 to
standard normal z_1, z_2

$$z_1 = \sqrt{-2 \ln x_1} \cos(2\pi x_2)$$
$$z_2 = \sqrt{-2 \ln x_1} \sin(2\pi x_2)$$

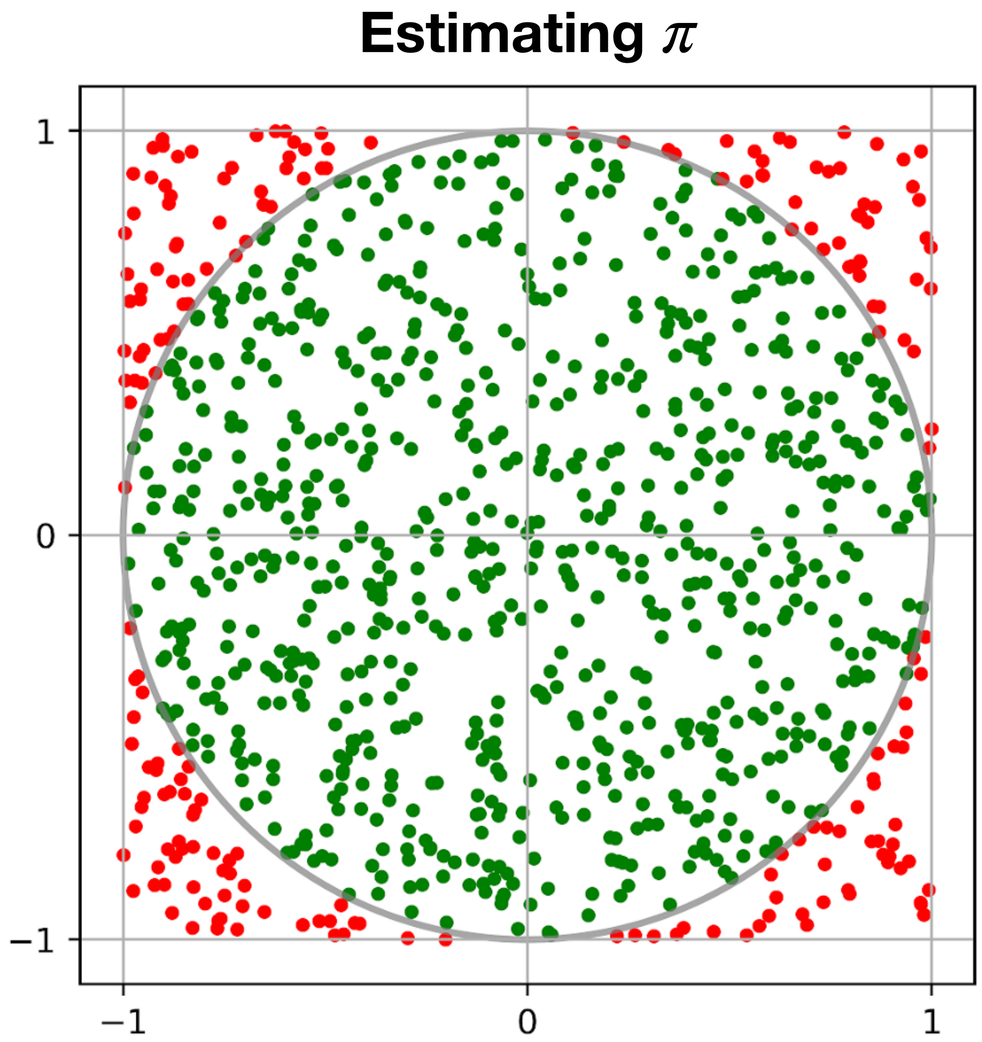
However, this will not always be possible :(

Monte Carlo Methods

Randomness to the rescue!

Stanislaw Ulam

Rejection Sampling



Accept-Reject Sampling with an envelope distribution (Von Neumann)

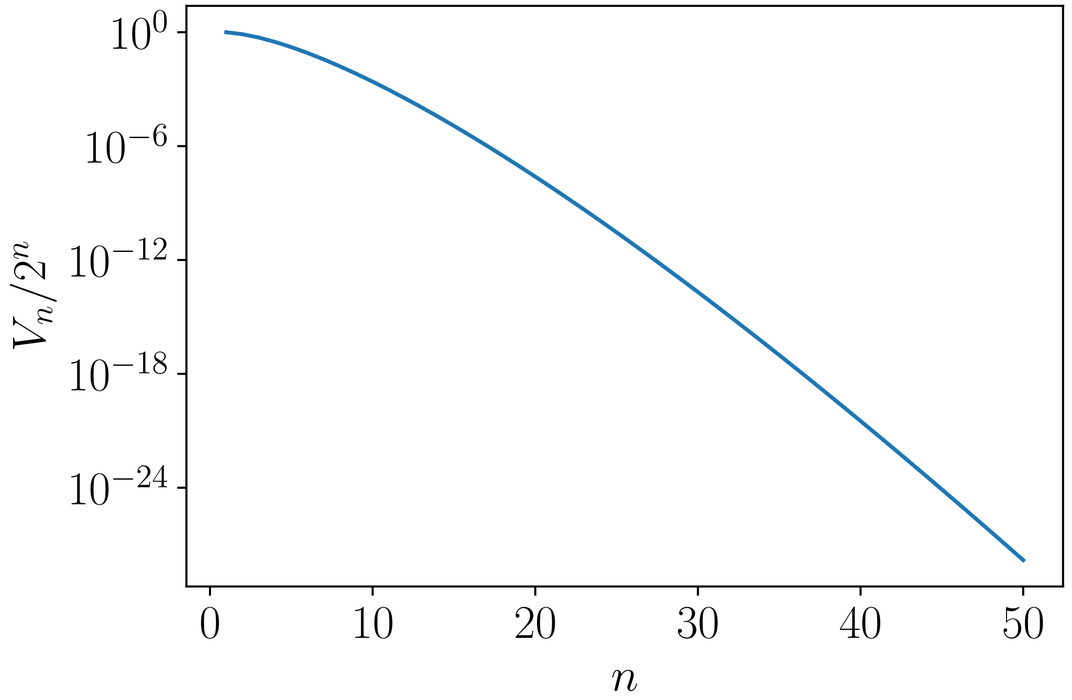
$$\frac{\pi}{4} = \frac{N(\text{accept})}{N(\text{total})}$$

Curse of dimensionality

In high dimensions the acceptance fraction can be extremely low

$$\frac{V_n}{2^n} = \left(\frac{\sqrt{\pi}}{2}\right)^n \frac{1}{\Gamma(1+n/2)}$$

‘Interesting’ region in high dimensions is tiny compared to full volume



Importance Sampling

Suppose we want to sample $p(x)$ but only have methods to sample $q(x)$

$$E_p[f(x)] = \int dx\, q(x) \left[\frac{p(x)}{q(x)} f(x)\right] = E_q \left[\frac{p(x)}{q(x)} f(x)\right]$$

$$E_p[f(x)] \approx \frac{1}{n_s} \sum_{i=1}^{n_s} \frac{p(x)}{q(x)} f(x)$$

Samples from $q(x)$

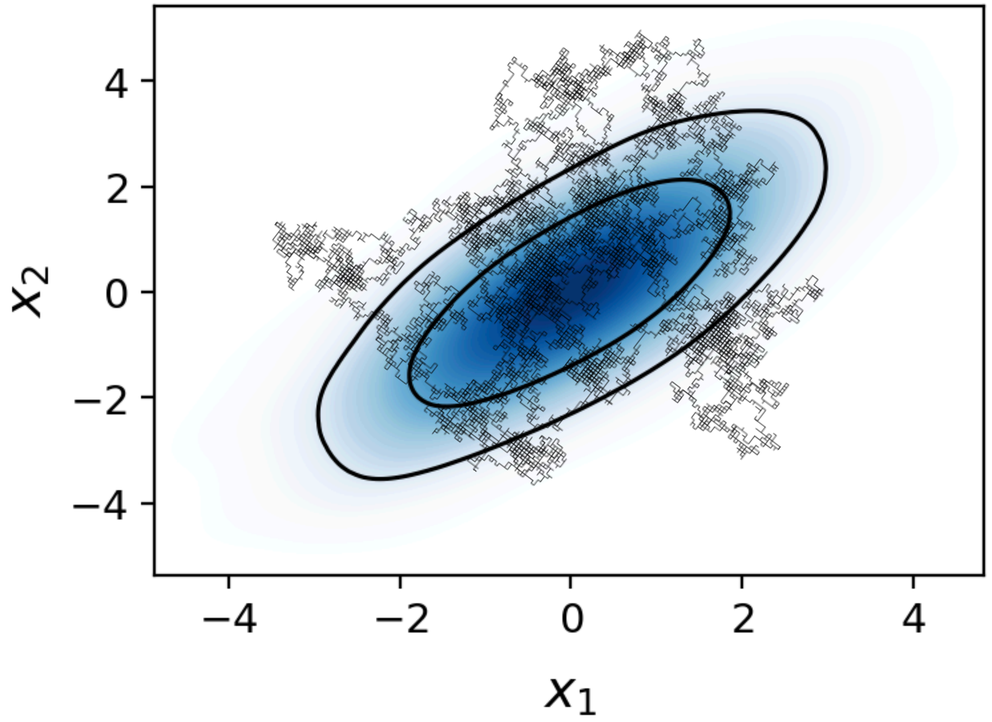
Works best when $p(x)$ and $q(x)$ are somewhat similar

Closely related to Monte Carlo Integration

Markov Chain Monte Carlo

Using a random walk, we can generate independent points in a way that the density of points $\propto P(\vec{\theta}\,|\,D)$

A method to ‘compress’ the full posterior distribution into samples



Markov Chain Monte Carlo

A Markov chain is a sequence of points (or random variables) such that the move

$$\vec{\theta}_i \rightarrow \vec{\theta}_{i+1}$$

depends only on θ_i (no memory beyond previous point)

The goal of Markov Chain Monte Carlo is to construct a sequence of points such that asymptotically, the probability of having a point $\vec{\theta}$ is $\propto P(\vec{\theta})$

Markov Chain Monte Carlo

Transition probabilities $\pi(x_2; x_1)$ should satisfy ***detailed balance***

$$P(x_1)\pi(x_2; x_1) = P(x_2)\pi(x_1; x_2)$$

Probability of being at x_1 ×
Probability of going to x_2 from x_1

If x_1 and x_2 occur at rates proportional to P, then the overall transition rates in each direction are the same —> stationary of the distribution

Markov Chain Monte Carlo

When detailed balance is satisfied and the chains have converged to the equilibrium/true distribution, it means that if x_1 is drawn from P, then so is x_2

$$\int P(x_1)\pi(x_2; x_1)dx_1 = P(x_2) = \int P(x_2)\pi(x_1; x_2)dx_1$$

(L.H.S) of detailed balance eq.

(R.H.S) of detailed balance eq.

Markov Chain Monte Carlo

Will chain converge to true distribution? Yes, eventually…

There is a unique stationary distribution for the Markov chain

The Markov chain eventually converges to the stationary distribution

See Neal (1993) for proof of convergence of MCMC.

Note — this does not say anything about the rate of convergence

Metropolis-Hastings algorithm

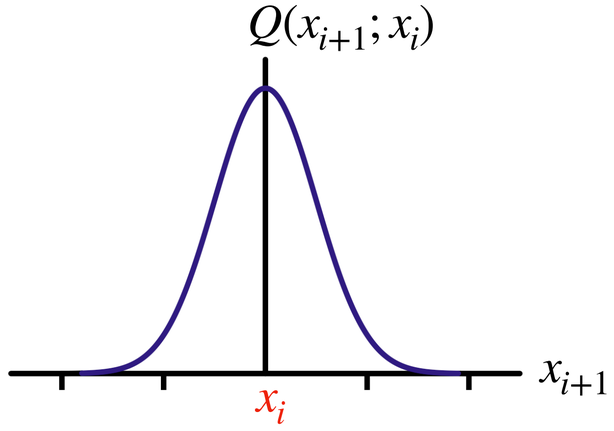
1. Start at some (random) location in parameter space
2. Take a random step according to **proposal** distribution $Q(x_{i+1}; x_i)$
3. Compute **acceptance** probability $= \min\left(1, \frac{P(x_{i+1})Q(x_i; x_{i+1})}{P(x_i)Q(x_{i+1}; x_i)}\right)$
4. **Accept/reject** move according to this probability
5. Repeat 2-4 until **convergence** criterion reached

Nice Animation of Metropolis-Hastings in action

Proposal distribution

Should be a function of current position only, e.g Gaussian

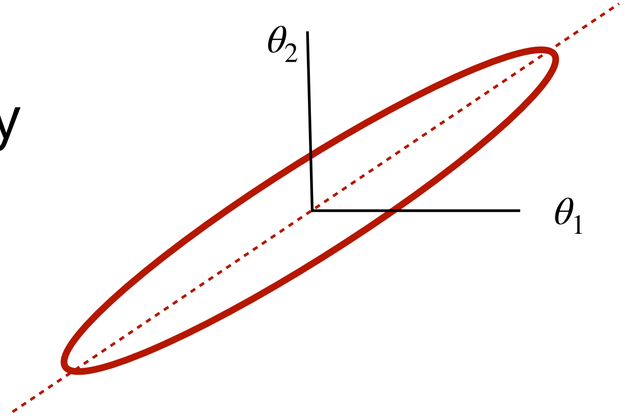
$$Q(x_{i+1}; x_i) \propto \exp\left[-\frac{(x_{i+1} - x_i)^2}{2\sigma^2}\right]$$



Example proposal distribution

Not necessary that it resembles the true posterior but helpful.

Smart choice of proposal can make the MCMC very efficient: important when parameters are strongly correlated



Proposal and Transition

Once we have a new state from the proposal, compute the acceptance probability

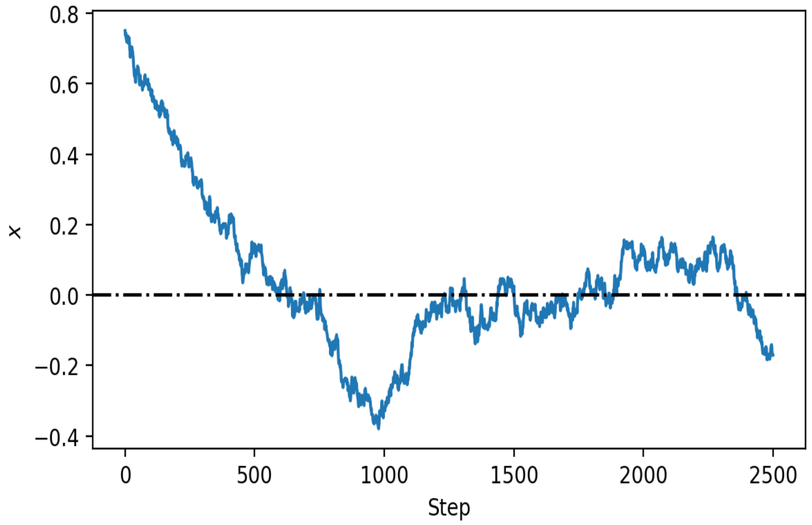
$$a = \frac{P(x_{i+1})Q(x_i; x_{i+1})}{P(x_i)Q(x_{i+1}; x_i)}$$

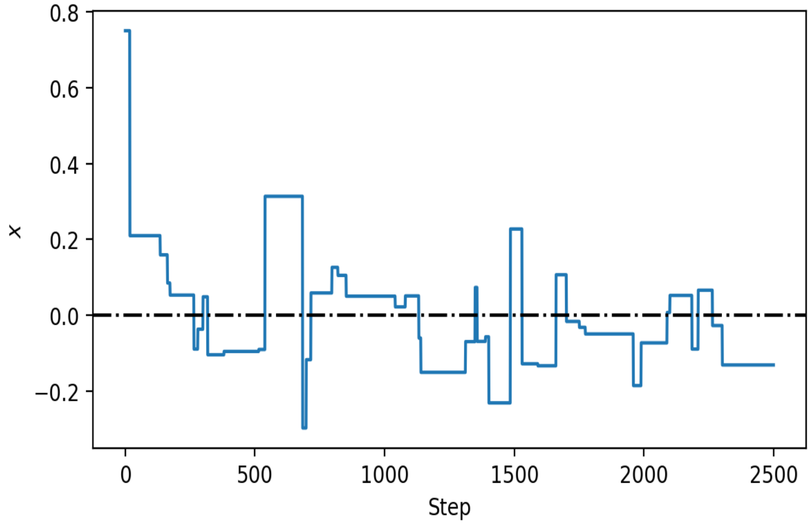
If $a > 1$, do the move

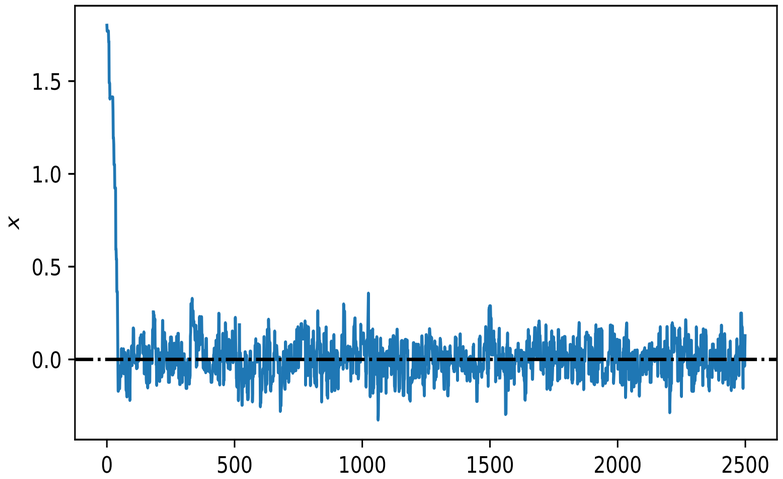
Otherwise generate $s \in U[0,1]$, if $s > a$ reject the move and add x_i to the samples, if $s < a$ do the move and add x_{i+1} to the sample

Typically proposal is symmetric so proposal ratio cancels out

Proposal and Transition







Which one has a good proposal distribution?

Convergence

When do we stop sampling?

Good practice to run 4-8 independent chains with different initial points, then compute Gelman-Rubin $(R-1)$ statistic

$$R-1 = \frac{\text{Variance of the means of the chains}}{\text{Mean of the variance of the individual chains}} \approx 10^{-2}$$

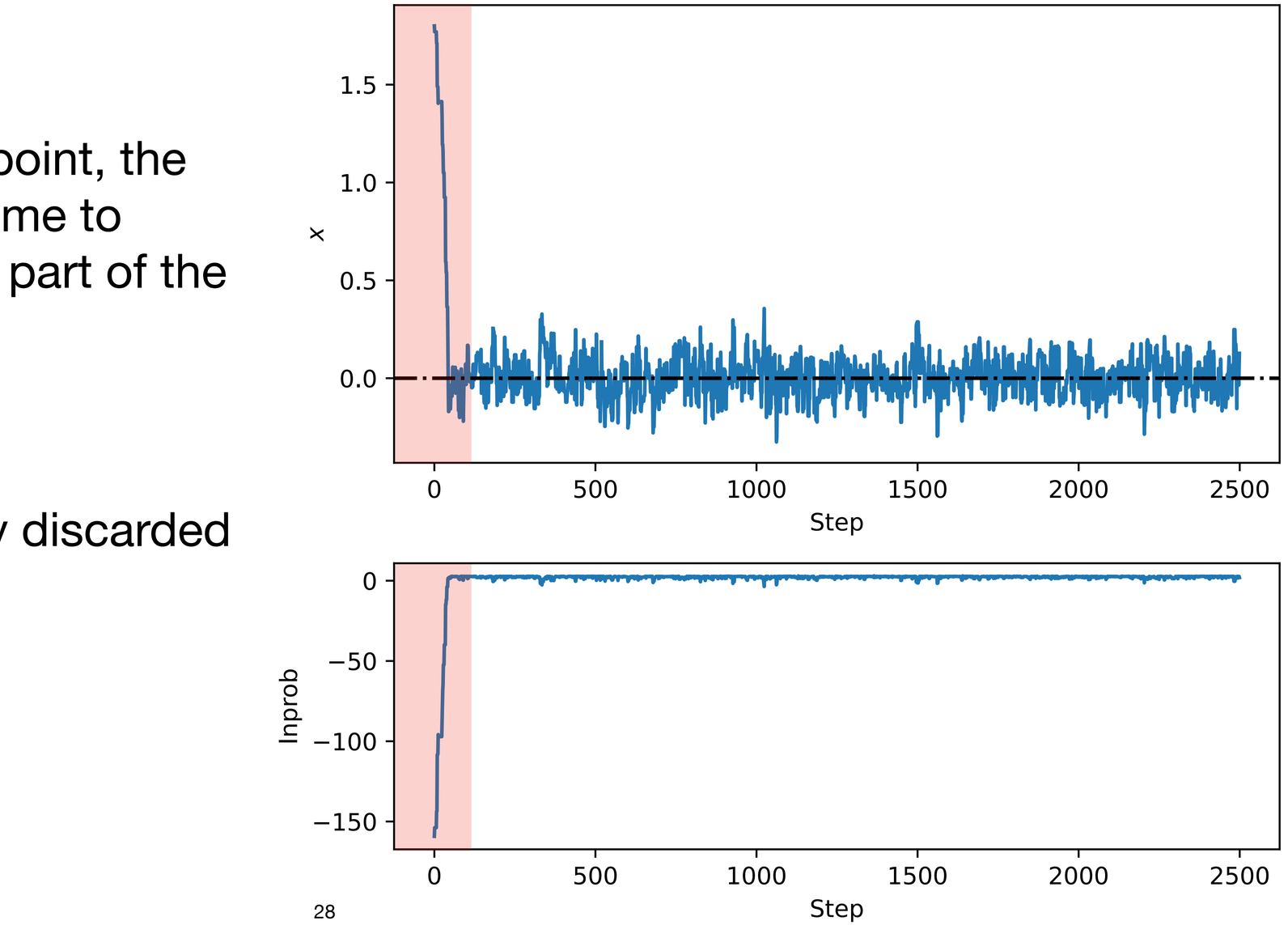
In other words, the means estimated from the different chains should not be too different

Many other useful statistics — all convergence statistics are merely **heuristics**

Burn-in

Depending on the starting point, the chains may take some time to equilibrate around the ‘good’ part of the distribution

This ‘burn-in’ phase is usually discarded



Inference using the MCMC sample

Easy to compute marginalised probability distributions for single parameter or combinations — just get the θ_i or $\{\theta_i, \ldots \theta_n\}$ values at the samples

Use these to get summary statistics — mean, credible intervals etc.

Can also importance sample to change likelihood, prior etc.

Not easy/useful to get conditional probabilities e.g. $P(H_0 \,|\, \Omega_m = 0.3)$ — there may not even be a single sample at $\Omega_m = 0.3$

Practical tips for MCMC

Smart choice of parameters/transformations can greatly speed up the process, ideally uncorrelated parameters with simple marginal distributions

If you don’t have a good proposal distribution initially, a short preliminary run can be used to estimate a covariance matrix for a new run

Most importantly, do not blindly trust the MCMC results — make sure things are sensible, a model can be bad but MCMC can still be converged

Always some trial and error involved!

Exercise 1a

MCMC theory

Check that the Metropolis-Hastings transition probability $\pi = aQ$ satisfies the detailed balance condition.

Some other question to think about…

What are the conditions under which the Markov chain may **not** converge?

Are there distributions which might be hard to sample with MCMC?

How does the number of MCMC samples required for inference scale with dimensions?

Exercise 1b

Writing your own MCMC code

Sample the 1D Gaussian distribution with $\mu = 0$, $\sigma = 0.1$ and a uniform prior in the range $[-1,1]$.

Take a Gaussian proposal distribution, run the mcmc for three values of the proposal jump size $\sigma = 0.01, 0.24, 10$. Experiment with different initial points of the mcmc (tails vs close to mean).

How do these choices affect the mcmc performance, burn-in, convergence?

(Python notebook with code structure provided)

Questions?

MCMC in Cosmology

- **Cobaya** (Python) [previously CosmoMC (Fortran90)]
- MontePython (C++ and Python)
- Emcee (Python)
- PyMC (Python)
- PTMCMC (Python)
- + more

Using Cobaya

Theory Block

Specify cosmological Boltzmann code (**CLASS** or **CAMB**) and relevant settings

```yaml
theory:
  camb:
    path: 'global'
    stop_at_error: False
    extra_args:
      halofit_version: mead
      bbn_predictor: PArthENoPE_880.2_standard.dat
      lens_potential_accuracy: 1
      num_massive_neutrinos: 1
      dark_energy_model: 'fluid'
      nnu: 3.046
      theta_H0_range:
      - 10
      - 100
```

Using Cobaya

Likelihood Block

Specify Likelihoods to use and relevant settings

```yaml
likelihood:
  planck_2018_lowl.TT: null
  planck_2018_lowl.EE: null
  planck_2018_highl_plik.TTTEEE_lite_native: null
  planck_2018_lensing.native: null
```

Using Cobaya

Params Block

Specify parameters, priors and reference distributions for initial point of MCMC

```yaml
params:
# Baseline cosmological parameters
  omch2:
    latex: \Omega_\mathrm{c} h^2
    prior:
      min: 0.08
      max: 0.16
    ref:
      dist: norm
      loc: 0.12
      scale: 0.001
    proposal: 0.0005
  logA:
    prior:
      min: 2.9
      max: 3.2
    ref:
      dist: norm
      loc: 3.05
      scale: 0.001
    proposal: 0.001
    latex: \log(10^{10} A_\mathrm{s})
```

Using Cobaya

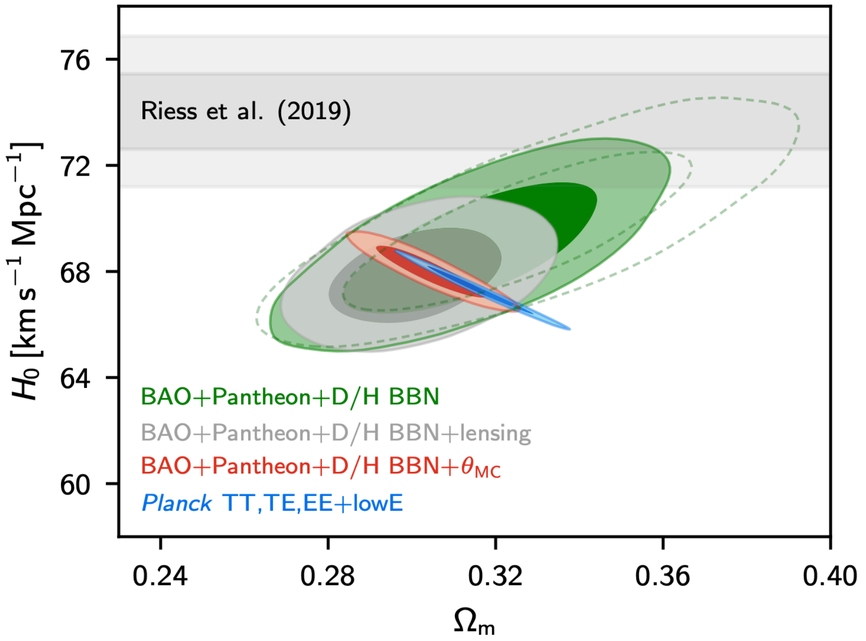
Sampler Block

Specify MCMC settings, output settings

```yaml
sampler:
  mcmc:
    drag: False
    oversample_power: 0.4
    proposal_scale: 1.9
    covmat: auto # replace with covmat for 2nd run
    Rminus1_stop: 0.01
    Rminus1_cl_stop: 0.2
output: chains/LCDM
```

Analysing MCMC output

- **GetDist** (example notebooks provided)
- Corner
- Anaesthetic



These codes perform smart binning and smoothing of the MCMC samples to produce such plots, also have methods to get summary statistics, convergence etc. from the samples

Exercise 2a (for laptop)

Background density and DE parameter estimation using SN data

Constrain $\Omega_m, \Omega_{\rm DE}$ and $w_{\rm DE}$ using PantheonPlus and Union3 supernovae data (only 1 supernovae dataset at a time)

How many samples does it take for convergence? Can you make it run faster?

Plot the contours for Ω_m vs $\Omega_{\rm DE}$ and Ω_m vs $w_{\rm DE}$. Explain the correlations in terms of the underlying physics. Is there evidence for $w_{\rm DE} \neq -1$?

(`Cobaya` input files, `GetDist` analysis notebook provided, takes ~ few min on your laptop)

Exercise 2b (for cluster)

Parameter estimation for the LCDM model using CMB data

Run the mcmc using `Cobaya`, analysing the LCDM model using the *Planck* CMB likelihoods. Sample the base LCDM parameters $\{\Omega_b h^2, \Omega_c h^2, A_s, n_s, \tau, \theta_{\mathrm{MC}}\}$

Quantify the deviation from a scale invariant power spectrum ($n_s = 1$). Plot the 1D and 2D contours for 6 parameters of the LCDM model. What kind of correlations exist between the different parameters and how do they relate to the underlying physics?

(`Cobaya` input files, `GetDist` analysis notebook and cluster job scripts are provided, MCMC will take about **4 to 8 hours** on the cluster)

Use the obtained covariance matrix (.covmat file) for a new mcmc run. Does it converge faster?

Cobaya on the cluster

See the GitHub readme

Load necessary modules (python,compiler,mpi)

Recommend to use python/conda virtual environments with python>=3.10

Install cobaya following the installation instructions

References

- Press et al., ***Numerical Recipes: The Art of Scientific Computing***
- Hobson et al., ***Bayesian Methods in Cosmology,*** (2010)
- Trotta R, ***Bayesian Methods in Cosmology,*** [arXiv:1701.01467]
- D. Hogg and D. Foreman-Mackey, ***Data analysis recipes: Using Markov Chain Monte Carlo***, Astrophys. J. Suppl. 236, no.1, 11 (2018), [arXiv:1710.06068].