Supernova MCMC/HMC exercise

Alan Heavens, ICIC, Imperial College London

ICIC Data Analysis Workshop September 2021 (online)

1 Cosmology with Supernovae Ia

The aim of this exercise is to write an MCMC code to infer cosmological parameters from supernova Ia data. Supernova Ia are standard candles (or can be made so), so can be used to measure the contents of the Universe.

As standard candles, the apparent brightness (or faintness, represented by a quantity μ) should depend on distance (or redshift z) in a parameter-dependent way, and is illustrated with the SNLS dataset in Fig. 1.

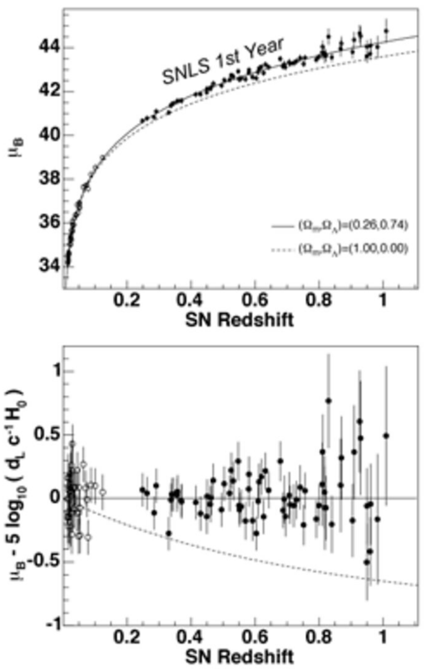

Figure 1: Supernova faintness-distance relation from SNLS.

2 Theory and parameters

The flux from a supernova of luminosity L is given by

$$f = \frac{L}{4\pi D_L^2}$$

where D_L is the *Luminosity Distance*. In Big Bang cosmology it is given by

$$D_L = \frac{(1+z)c}{H_0\sqrt{|1-\Omega|}} S_k(r),$$

where

$$r(z) = \sqrt{|1-\Omega|}\int_0^z \frac{dz'}{\sqrt{\Omega_\mathrm{m}(1+z')^3 + \Omega_\mathrm{v} + (1-\Omega)(1+z')^2}}.$$

and $S_k(r) = \sin r, r, \sinh r$, depending on whether $\Omega \equiv \Omega_\mathrm{m} + \Omega_\mathrm{v}$ is $> 1, = 1$, or < 1, and z is the observed redshift of the supernova. $\Omega_\mathrm{m}, \Omega_\mathrm{v}$ and H_0 are the density parameters (today) in matter, vacuum energy, and the Hubble constant. It is beyond the scope of these notes to derive this, but it is standard material for an undergraduate cosmology course.

For a flat Universe ($\Omega = 1$), this simplifies to

$$D_L(z) = 3000h^{-1}(1+z)\int_0^z \frac{dz'}{\sqrt{\Omega_\mathrm{m}(1+z')^3 + 1 - \Omega_\mathrm{m}}}\ \mathrm{Mpc},$$

where $H_0 = 100h\mathrm{km\,s^{-1}\ Mpc^{-1}}$. To avoid evaluating integrals to calculate D_L, we can use an accurate fitting formula (valid for flat universes only), given by U.-L. Pen, ApJS, 120, 49 (1999):

$$D_L(z) = \frac{c}{H_0}(1+z)\left[\eta(1,\Omega_m) - \eta\left(\frac{1}{1+z}, \Omega_m\right)\right]$$

where

$$\eta(a,\Omega_m) = 2\sqrt{s^3+1}\left[\frac{1}{a^4} - 0.1540\frac{s}{a^3} + 0.4304\frac{s^2}{a^2} + 0.19097\frac{s^3}{a} + 0.066941 s^4\right]^{-1/8}$$

and $s^3 \equiv (1-\Omega_m)/\Omega_m$. This is accurate to better than 0.4% for $0.2 \le \Omega_m \le 1$.

Fluxes are usually expressed in magnitudes, where $m = -2.5\log_{10} F$+constant. The distance modulus is $\mu = m - M$, where M is the absolute magnitude, which is the value of m if the source is at a distance 10pc. With D_L in Mpc[1], this is

$$\mu = 25 - 5\log_{10} h + 5\log_{10}\left(\frac{D_L^*}{\mathrm{Mpc}}\right)$$

The Hubble constant has been factored out of D_L: $D_L^* \equiv D_L(h=1)$.

If we have measurements of μ, then we can use Bayesian arguments to infer the parameters $\Omega_\mathrm{m}, \Omega_\mathrm{v}, h$. For anyone unfamiliar with cosmology, these numbers are somewhere between 0 and 1.

3 Data

The data file (from the 'JLA' sample - see `http://supernovae.in2p3.fr/sdss_snls_jla/ReadMe.html` for more detail) consists of data from about 700 supernovae, which are averaged together in 31 narrow bins of redshift, to give a distance modulus μ for each bin.

The sample file (`jla_mub.txt` from Discord), contains $n = 31$ pairs of (z, μ), corresponding to bins containing supernovae with $z < 1.3$.

4 Exercise

Write an MCMC code to infer h and Ω_m from the supernova dataset, assuming the Universe is flat and the errors are gaussian[2], i.e. assume that the likelihood is

$$L \propto \exp\left[-\frac{1}{2}\sum_{i,j=1}^{n}[\mu_i - \mu_\mathrm{th}(z_i)]C_{ij}^{-1}[\mu_j - \mu_\mathrm{th}(z_j)]\right]$$

[1]There is a simplification in the exercise here; we assume we know what the absolute magnitude (or luminosity) of type I supernovae are, but in fact unless we have supernovae with known distances, we don't. In fact M and h are degenerate, since M is set from low-redshift supernovae where we assume Hubble's law to give us the distance. For the purpose of this exercise, we will cheat.

[2]This is largely justified by the averaging over large numbers of supernovae, and using the central limit theorem, but is a simplification

where μ_{th} is the theoretical value of the distance modulus, for which you will need to compute the integral for D_L^* numerically, using the fitting formula (for a flat Universe). For clarity, we have not written the full dependence of $\mu_{\rm th}$; we should write $\mu_{\rm th}(z; \Omega_{\rm m}, h)$, and indeed it also depends on the (LCDM) model M.

C is the 31×31 covariance matrix of the data, provided as a list of numbers in an obvious order, from the website, in the file `jla_mub_covmatrix.txt`.

- h and Ω_m are positive, and have values of the rough order of unity
- Assume uniform priors on the parameters (so you compute the likelihood)
- You might like to start with a very simple 'top-hat' proposal distribution, where the new point is selected from a rectangular region centred on the old point. For this you will need a simple random number generator. Or use a gaussian for each parameter.
- Explore visually the chain when you have (a) a very small proposal distribution, and (b) a very large proposal distribution, for a maximum of 1000 trials. What do you conclude?
- Show how the acceptance probability changes as you change the size of the proposal distribution from very small (say 0.001) to very large (say 100).
- Once you have settled on a 'reasonable' proposal distribution, compute the average value of the parameters under the posterior distribution, and their variances and covariance.
- Optionally, generalise to non-flat Universes and include $\Omega_{\rm v}$ as an independent parameter. You will need to perform the integral for D_L^* numerically.

4.1 Tips

If you are estimating h, $\Omega_{\rm m}$ and $\Omega_{\rm v}$, you can precalculate D_L for $h = 1$ as a function of $\Omega_{\rm m}$ and $\Omega_{\rm v}$, and do a bilinear interpolation when you are running the chains (and divide by h). You only need to do this at the 31 redshifts of the JLA sample. This will be much faster than computing D_L every time you change parameters. This is *not* necessary if your parameters are h and Ω_m only.

5 Optional Extensions

- Importance sampling. Consider a non-flat prior, so the target distribution is the posterior, not the likelihood. We can still sample from the likelihood (as you have been doing), and construct the posterior by weighting the points with the prior to get the target. This is an example of *importance sampling*, where we sample from a different distribution from the one we eventually want. Apply a prior on the Hubble constant to your chain, assuming a gaussian prior with mean 0.738 and standard deviation 0.024. Now plotting all the points in the chain will give a graph which looks the same as your previous graphs, so what should you do? Compute the mean h, Ω_m with and without the prior.
- Write and apply a Gelman-Rubin convergence test, and deduce roughly how long the chains should be for convergence.
- Extend to perform Hamiltonian Monte Carlo. You might like to try to compare the performance of MCMC and HMC; you will need to decide what the right criterion is.

For HMC, the algorithm is (from Hajian 2006):

Hamiltonian Monte Carlo

```
1: initialize $\mathbf{x}_{(0)}$
2: for i = 1 to $N_{samples}$
3:         $\mathbf{u} \sim \mathcal{N}(0,1)$ (Normal distribution)
4:         $(\mathbf{x}^*_{(0)}, \mathbf{u}^*_{(0)}) = (\mathbf{x}_{(i-1)}, \mathbf{u})$
5:         for j = 1 to N
6:                 make a leapfrog move: $(\mathbf{x}^*_{(j-1)}, \mathbf{u}^*_{(j-1)}) \to (\mathbf{x}^*_{(j)}, \mathbf{u}^*_{(j)})$
7:         end for
8:         $(\mathbf{x}^*, \mathbf{u}^*) = (\mathbf{x}_{(N)}, \mathbf{u}_{(N)})$
9:         draw $\alpha \sim$ Uniform(0,1)
10:         if $\alpha < \min\{1, e^{-(H(\mathbf{x}^*,\mathbf{u}^*)-H(\mathbf{x},\mathbf{u}))}\}$
11:                 $\mathbf{x}_{(i)} = \mathbf{x}^*$
12:         else
13:                 $\mathbf{x}_{(i)} = \mathbf{x}_{(i-1)}$
14: end for
```

$H = -\ln L + K$, where $K = \mathbf{u}\cdot\mathbf{u}/2$. The exact derivative of $U = -\ln L$ is easy to compute for h, but the derivative with respect to Ω_m is complicated. Sympy (https://www.sympy.org/en/index.html) is one way to differentiate U automatically and produce (quite a few lines of!) python code, and there are other possibilities (Jax, autograd). Speak to demonstrators for suggestions.

Aside: an alternative approach is to approximate U by a bivariate gaussian with covariances estimated from the MCMC code:

$$U = \frac{1}{2}(\theta - \theta_0)_\alpha C^{-1}_{\alpha\beta}(\theta - \theta_0)_\beta.$$

Since it's approximate, H will not be conserved, but the Metropolis step sorts everything out. The derivatives are then easy and analytic (but approximate).

You should use the leapfrog algorithm (which is forward-backward symmetric, as required for detailed balance)

$$\begin{aligned} u_i\left(t+\frac{\epsilon}{2}\right) &= u_i(t) - \frac{\epsilon}{2}\left(\frac{\partial U}{\partial x_i}\right)_{\mathbf{x}(t)} \\ x_i(t+\epsilon) &= x_i(t) + \epsilon u_i\left(t+\frac{\epsilon}{2}\right) \\ u_i(t+\epsilon) &= u_i\left(t+\frac{\epsilon}{2}\right) - \frac{\epsilon}{2}\left(\frac{\partial U}{\partial x_i}\right)_{\mathbf{x}(t+\epsilon)}. \end{aligned} \tag{1}$$

Issues to consider are how many integration steps per point in the chain, and how big those steps should be. For some discussion, see Hajian (2006), astroph/0608679.
Alan Heavens