Aachen CosmoTools 2018: Statistics exercises

Will Handley

April 24, 2018

Background Theory

The normalised Friedmann equation of the universe can be written as:

$$\left(\frac{H}{H_0}\right)^2 = \Omega_{r,0}\, a^{-4} + \Omega_{m,0}\, a^{-3} + \Omega_{k,0}\, a^{-2} + \Omega_{\Lambda,0}, \qquad a = \frac{1}{1+z} \tag{1}$$

where:

- a is the time-dependent scale-factor of the universe, normalised to 1 today.
- z is the observed redshift of an object located at a previous epoch.
- $H = \frac{d}{dt}\log a$ is the Hubble parameter, with present value H_0.
- $\Omega_i = 8\pi G\rho_i/3H^2$ is the density parameter for component i, with present value $\Omega_{i,0}$.

From these definitions, you should note that by setting $a = 1$ (today), $1 = \Omega_{r,0} + \Omega_{m,0} + \Omega_{k,0} + \Omega_{\Lambda,0}$.

With this, along with a little geometry, we may compute several cosmological distance measures:

$$d_L = \frac{d_M}{1+z} \qquad \text{(Luminosity dist.)} \tag{2}$$

$$d_M = \begin{cases} \frac{d_H}{\sqrt{\Omega_{k,0}}} \sinh \frac{d_C\sqrt{\Omega_{k,0}}}{d_H} & \Omega_{k,0} > 0 \\ d_C & \Omega_{k,0} = 0 \\ \frac{d_H}{\sqrt{|\Omega_{k,0}|}} \sin \frac{d_C\sqrt{|\Omega_{k,0}|}}{d_H} & \Omega_{k,0} < 0 \end{cases} \qquad \text{(Transverse comoving dist.)} \tag{3}$$

$$d_C = d_H \int_0^z \frac{dz}{\sqrt{\Omega_{r,0}a^{-4} + \Omega_{m,0}a^{-3} + \Omega_{\Lambda,0}}} \qquad \text{(Comoving dist.)} \tag{4}$$

$$d_H = \frac{c}{H_0} \qquad \text{(Hubble dist.)} \tag{5}$$

For specific kinds of supernovae/astronomical objects, we can directly measure the magnitude $\mu = m - M$, which is related to their luminosity distance d_L:

$$\mu = 5(\log_{10} d_L - 1) \qquad \text{(obviously...)} \tag{6}$$

We measure the magnitudes μ and redshifts z of N supernovae, and combine them into two data vectors $y = (\mu_1, \ldots, \mu_N)$, $x = (z_1, \ldots, z_N)$. These magnitudes also come with an associated error, or more precisely a covariance matrix Σ.[1]. Given a specific cosmology defined by the parameters $\theta = (H_0, \Omega_{r,0}, \Omega_{m,0}, \Omega_{r,0}, \Omega_{\Lambda,0}, \Omega_{k,0})$, we can compute the luminosity distance (2) as a function of redshift, and thus the theoretical magnitude of the object $\hat{\mu}(z;\theta)$ for any given z, and thus the theoretical data vector $\hat{y}(\theta) = (\hat{\mu}(z_1;\theta), \ldots, \hat{\mu}(z_N;\theta))$. The likelihood of observing this data is then:

$$\mathcal{L}(\theta) = P(y, x|\theta) = \frac{1}{\sqrt{\det 2\pi\Sigma}} \exp\left(-\frac{1}{2}[y - \hat{y}(\theta)]^T \Sigma^{-1} [y - \hat{y}(\theta)]\right) \tag{7}$$

This object forms the centre of our inference. The data are taken from: `http://supernova.lbl.gov/Union/` which you should visit to see some example figures.

1 Plotting the data (5-10 mins)

Set up and download the supernovae likelihoods:

```
mkdir $HOME/stats_examples
cd $HOME/stats_examples
wget www.mrao.cam.ac.uk/~wh260/cosmotools18/SNE.tar.gz
tar -xvf SNE.tar.gz
```

1. Create a file (or workbook) containing the following python code:

```
import matplotlib.pyplot as plt
from SNE.distance import D_Ls, distance_modulus
from SNE.supernova_data import zs, mods, mod_errs

fig, ax = plt.subplots()
ax.errorbar(zs, mods, yerr=mod_errs, fmt='.')
ax.set_xlabel(r'redshift ($z$)')
ax.set_ylabel(r'distance modulus ($\mu=m-M$)')

d_Ls_theory = D_Ls(zs, O_L=0.7, O_m=0.3)
mods_theory = distance_modulus(d_Ls_theory)

ax.plot(zs, mods_theory)
plt.show()
```

[1] These statements are grossly misleading. Much of the research/controversy in this field goes into/comes from (in)correctly modelling the errors in the Hubble distance ladder.

2. Modify the script so that instead of plotting magnitudes, you plot luminosity distances d_L against redshift z (Hint: consider equation (6), and make sure you calculate the errors correctly).

3. Question: Why can't we turn this plot into something more theoretically intuitive, such as scale-factor against cosmic time?

2 Metropolis Hastings (20-30 mins)

I have written up the Gaussian likelihood for you (if you are interested, it's all in `SNE/supernova_data.py`). For now, we will assume that the universe is flat $\Omega_{k,0} = 0$, with only matter and dark energy. An example of a likelihood call is:

```
from SNE.supernova_data import loglikelihood_sys
H_0, O_m = 60, 0.3
O_L, O_k, O_r = 1. - O_m, 0., 0.
loglikelihood_sys(H0, O_r, O_m, O_L, O_k)
```

4. Write a two-dimensional Metropolis-Hastings algorithm to explore the likelihood for flat universes with only dark energy and matter in it. You should choose your parameters to be H_0 and $\Omega_{m,0}$, with the value of dark energy therefore being $\Omega_{\Lambda,0} = 1 - \Omega_{m,0}$. The general algorithm is:

   (a) Start at some initial parameter value θ_0

   (b) at step i, propose a new $\theta_{\rm prop}$ dependent on θ_i, e.g. a Gaussian distribution width d centered on θ_i, where d is chosen by the user (use `numpy.random.normal`) .

   (c) accept the step with probability $\alpha = L(\theta_{\rm prop})/L(\theta_i)$. Note, if $\alpha > 1$, then it is automatically accepted. If accepted, set $\theta_{i+1} = \theta_{\rm prop}$, otherwise $\theta_{i+1} = \theta_i$.

   (d) Go back to step (4b) until $i = n_{\rm max} \sim 10^5$.

   If you haven't managed to debug your code after 20 minutes or so, you can copy the solution in `www.mrao.cam.ac.uk/~wh260/cosmotools18/answers/MH_1.py`

5. Plot your chain of points using `matplotlib`.

6. Question: Comment on the properties of the chain produced.

7. Reduce the number of lines in your code. Talk to your neighbor for advice

8. How do the properties of the chain change if you choose a more (in)sensible starting point? or a different step size? Even if you tune these correctly, do you notice anything funny about the properties of your chains?

9. Plot your chains using getdist (install with `pip install getdist`):

```
import getdist
O_L_array = 1. - np.array(O_m_array)
samples = np.array([H0_array, O_m_array, O_L_array]).T
names = ['H0','O_m',r'O_L*']
labels = ['H_0',r'\Omega_m',r'\Omega_\Lambda']
samples = getdist.MCSamples(samples=samples, names=names, labels=labels)
g = getdist.plots.getSubplotPlotter()
g.triangle_plot(samples,filled=True)
plt.show()
```

10. Is there anything different if you use the likelihood that doesn't take into account systematic errors? (change `loglikelihood_sys` for `loglikelihood_nosys`)

3 PolyChord (20 mins)

Download and install PolyChord:

```
# PolyChord
cd $HOME/stats_examples
wget www.mrao.cam.ac.uk/~wh260/cosmotools18/PolyChord_v1.13.tar.gz
tar -xvf PolyChord_v1.13.tar.gz
cd PolyChord
make FC=mpif90   # replace mpif90 with the name of your fortran mpi compiler
python setup.py install --user   # choose python 3 at this stage
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/stats_examples/PolyChord/lib
ulimit -s unlimited
python run_PyPolyChord.py
# Follow the instructions to modify your LD_PRELOAD and/or LD_LIBRARY_PATH --
```

repeat the above command until no error is produced. You should get some output that finishes with the lines like:

```
 ---------------------------------------------------
|                                                   |
| ndead  =         1092                             |
| log(Z) =          -2.43834 +/-           0.26105 |
|___________________________________________________|
```

You should put the `LD_LIBRARY_PATH` and `LD_PRELOAD` commands into your .bashrc

```
ulimit -s unlimited
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/stats_examples/PolyChord/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/stats_examples/MultiNest/lib
export LD_PRELOAD=<what PolyChord tells you to put in here>
```

for certain versions of mac OS, you may need to specify a number rather than `unlimited`. Set it as high as it allows.

11. Start by using polychord to plot the likelihood you just worked on:

    ```
    wget www.mrao.cam.ac.uk/~wh260/cosmotools18/run_PyPolyChord_sne.py
    mpirun -np 1 python run_PyPolyChord_sne.py
    ```

    You can change `-np 1` to a higher number to increase the number of MPI cores which PolyChord runs on. What is the fundamental difference between the likelihood which includes systematic errors, and the one that does not? Have a look at the script to see how PolyChord is set up. Most of the code is purely interfacing our code with PolyChord's input, but some of the settings can be important.

12. Now for the punch-line. Run:

    ```
    wget www.mrao.cam.ac.uk/~wh260/cosmotools18/run_PyPolyChord_sne_all.py
    mpirun -np 1 python run_PyPolyChord_sne_all.py
    ```

    This will run three models:

    (a) matter + dark energy (flat)

    (b) matter + dark energy + curvature

    (c) matter + curvature (no dark energy).

    After it's done, what can you say about both the evidences of each model, and the comparison of the posteriors with & without curvature?

13. Modify the above script by adding in/removing components of the universe (radiation etc).

4 Bonus Questions/extended investigations

To be attempted in any order

14. Modify code to allow for a dark energy with a variable equation of state parameter w (i.e. change $\Omega_{\Lambda,0} \to \Omega_{\Lambda,0}\, a^{-3(1+w)}$).

15. Modify your MH algorithm to include convergence diagnostics, such as split-Rhat. A good reference can be found in the stan manual:
    `https://github.com/stan-dev/stan/releases/download/v2.14.0/stan-reference-2.14.0.pdf`
    (28.3. Initialization and Convergence Monitoring)

16. How is run-time and evidence accuracy affected by the PolyChord setting `nlive`?

17. How sensitive are your conclusions to the prior widths (qualitatively and quantitatively)?

18. Plot the predictive posterior distribution using `fgivenx`

19. Compare PolyChord with MultiNest:

    Download and install MultiNest with:

```
# MultiNest
cd $HOME/stats_examples
wget www.mrao.cam.ac.uk/~wh260/cosmotools18/MultiNest_v3.11_CMake.tar.gz
tar -xvf $HOME/Downloads/MultiNest_v3.11_CMake.tar.gz
mv MultiNest_v3.11_CMake/multinest MultiNest
rm -r MultiNest_v3.11_CMake
cd MultiNest
cd build
cmake ..
make
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/stats_examples/MultiNest/lib

# pyMultiNest
pip install pymultinest

#or

cd $HOME/stats_examples
git clone https://github\chiom/JohannesBuchner/PyMultiNest/
cd PyMultiNest
python setup.py install --user
```

    you should get some output that finishes with lines like:

```
ln(ev)=   235.74020481292135      +/-  0.12530119876331158     Total
Likelihood Evaluations:         5173
 Sampling finished. Exiting MultiNest
```

    modify the `run_PyPolyChord.py` script to run MultiNest instead.