Monte Python likelihoods

Benjamin Audren

École Polytechnique Fédérale de Lausanne

31/10/2014

Design

4 rules, for MyLikelihood

- `Mylikelihood` must be a folder in `montepython/likelihoods/`
- `Mylikelihood` must contain two files,
  `__init__.py` and `Mylikelihood.data`
- `__init__.py` must define a `class` called `Mylikelihood`, inheriting from `Likelihood` (at least)
- `class Mylikelihood` must define a function called `loglkl` that returns the **log likelihood**

Reasoning

- no **hard-coding** of acceptable likelihoods
- no need to touch the **source code** when adding a likelihood
- **object oriented** design (look at the Planck likelihoods)

Available functions

- `self.need_cosmo_arguments(data, {'output': 'mPk'})`
- `data.mcmc_parameters['alpha']['current']`
- `data.get_mcmc_parameters(['cosmo'])`

Practical example: JLA

data file

```
JLA.data_directory = os.path.join(data.path['data'], 'JLA')
JLA.settings       = 'jla.dataset'
JLA.conflicting_experiments = ['JLA_simple']

JLA.use_nuisance = ['alpha', 'beta', 'M', 'Delta_M']
```

Practical example: JLA

source file

```python
class JLA(Likelihood_sn):

    def __init__(self, path, data, command_line):

        Likelihood_sn.__init__(self, path, data, command_line)

        # Load matrices from text files, whose names were read in the
        # configuration file
        self.C00 = self.read_matrix(self.mag_covmat_file)
        self.C11 = self.read_matrix(self.stretch_covmat_file)
        self.C22 = self.read_matrix(self.colour_covmat_file)
        self.C01 = self.read_matrix(self.mag_stretch_covmat_file)
        self.C02 = self.read_matrix(self.mag_colour_covmat_file)
        self.C12 = self.read_matrix(self.stretch_colour_covmat_file)

        # Reading light-curve parameters from self.data_file (jla_lcparams.txt)
        self.light_curve_params = self.read_light_curve_parameters()
```

Practical example: JLA

source file

```
class JLA(Likelihood_sn):
```

```
    def loglkl(self, cosmo, data):
        """
        Compute negative log-likelihood (eq.15 Betoule et al. 2014)

        """
```

...

```
        redshifts = self.light_curve_params.zcmb
        size = redshifts.size

        moduli = np.empty((size, ))
        for index, row in self.light_curve_params.iterrows():
            z_cmb = row['zcmb']
            moduli[index] = cosmo.luminosity_distance(z_cmb)
        moduli = 5 * np.log10(moduli) + 25
```

Practical example: JLA

source file

```
class JLA(Likelihood_sn):
```

```
    def loglkl(self, cosmo, data):
        """
        Compute negative log-likelihood (eq.15 Betoule et al. 2014)

        """
```

...

```
        # Compute the residuals (estimate of distance moduli - exact moduli)
        residuals = np.empty((size,))
        sn = self.light_curve_params
        # This operation loops over all supernovae!
        # Compute the approximate moduli
        residuals = sn.mb - (
            M - alpha*sn.x1 + beta*sn.color + Delta_M*(
                sn.thirdvar > self.scriptmcut))
        # Remove from the approximate moduli the one computed from CLASS
        residuals -= moduli

        # Update the diagonal terms of the covariance matrix with the
        # statistical error
        cov += np.diag(sn.dmb**2 + (alpha*sn.dx1)**2 + (beta*sn.dcolor)**2
                       + 2.*alpha*sn.cov_m_s
                       - 2.*beta*sn.cov_m_c
                       - 2.*alpha*beta*sn.cov_s_c)
```

Practical example: JLA

source file

```python
class JLA(Likelihood_sn):
```

```python
    def loglkl(self, cosmo, data):
        """
        Compute negative log-likelihood (eq.15 Betoule et al. 2014)

        """
```

...

```python
        # Whiten the residuals, in two steps
        # 1) Compute the Cholesky decomposition of the covariance matrix, in
        # place. This is a time expensive (0.015 seconds) part
        cov = la.cholesky(cov, lower=True, overwrite_a=True)

        # 2) Solve the triangular system, also time expensive (0.02 seconds)
        residuals = la.solve_triangular(cov, residuals, lower=True, check_finite=False)

        # Finally, compute the chi2 as the sum of the squared residuals
        chi2 = (residuals**2).sum()
```

List of existing likelihoods in Monte Python

ls montepython/likelihoods

- `Planck_highl`, `Planck_lowl`
- `Planck_actstp`, `Planck_lensing`
- `Planck_SZ`, `lowlike`
- `clik_wmap_full` and `lowl`
- `bicep`, `bicep2`, `acbar`
- `boomerang`, `cbi`, `quad`
- `spt` and `spt_2500`
- `WiggleZ`, `sdss_lrgDR4`
- `euclid_lensing`, `euclid_pk`
- `sn`, `hst`, `timedelay`
- `bao`, `bao_boss`, ...

Exercise: Implementing a new likelihood

Simplest example

I) HST-like likelihood

An experiment called `hubble_2013` measured

$$h = 0.712 \pm 0.012$$

Create this likelihood and use it in a run.

Hints for Exercice I

- create all the needed files/folder first
- define the data associated to the measurement in the `.data` file
- inherit from `Likelihood`
- get inspiration from `hst`