Example of an advanced computing package: `PyMC`

ESOpy 2.0 – January 10, 2018– 12:15 - 13:00

Jan Bolmer

Goals - Fitting an absorption line using PyMC

(The real goal: start programming in Python!
+ demonstrate the power of Python!)

- Load data from a .csv file using Pandas (loading data from an external file)
- Defining the model (Voigt absorption profile)
- Defining and choosing priors (PDFs)
- Starting the MCMC and extracting the best fit (saving, storing results ..)
- Plotting the results (matplotlib, seaborn ...)

Outline

PyMC - Version 2.3.6

Purpose

`https://pymc-devs.github.io/pymc/`

> PyMC *is a python module that implements Bayesian statistical models and fitting algorithms, including Markov chain Monte Carlo. Its flexibility and extensibility make it applicable to a large suite of problems. Along with core sampling functionality,* PyMC *includes methods for summarizing output, plotting, goodness-of-fit and convergence diagnostics.*

Marcov Chain Monte Carlo, Bayesian Statistics

class of algorithms used to efficiently sample posterior distributions

Monte Carlo:

Generation of random Numbers (sample from a distribution)

Marcov Chain:

chain of numbers, with each number depending on the previous number

$$\theta_{t+1} = \text{Normal}\left(\theta_t, \sigma\right)$$

Marcov Chain Monte Carlo, Bayesian Statistics

class of algorithms used to efficiently sample posterior distributions

Monte Carlo:

Generation of random Numbers (sample from a distribution)

Marcov Chain:

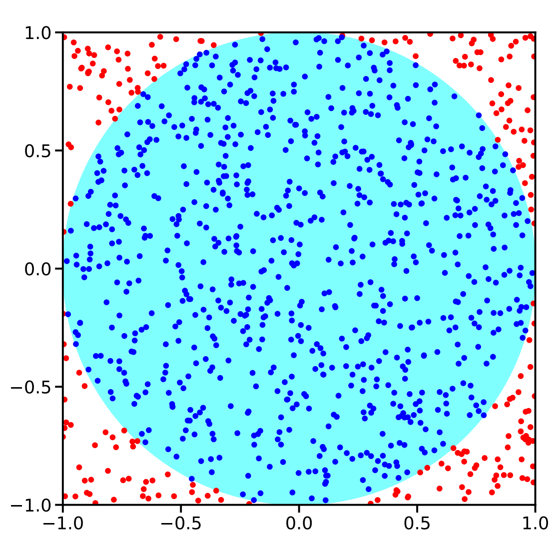
chain of numbers, with each number depending on the previous number

$$\theta_{t+1} = \text{Normal}\left(\theta_t, \sigma\right)$$

Bayesian Statistics: We are interested in the Probability/Posterior Distribution of a (set of) parameter(s) θ, which we want to sample

$$P\left(\theta|D, M\right) = \frac{P\left(D|\theta, M\right) P\left(\theta\right)}{P\left(D\right)} \text{ (Bayes Theorem)}$$

Example - Calculating π using Monte Carlo



Metropolis-Hastings Algorithm

Algorithm to decide weather a new value should be accepted or not, e.g. the Metropolis Hastings Algorithm

$$\theta_{t+1} = \mathrm{Normal}\left(\theta_t, \sigma\right)$$

$$a = \frac{P\left(\theta_{t+1}|D, M\right)}{P\left(\theta_t|D, M\right)} \overset{\text{Bayes Theorem}}{=} \frac{\frac{P(D|\theta_{t+1}, M)P(\theta_{t+1})}{P(D)}}{\frac{P(D|\theta_t, M)P(\theta_t)}{P(D)}} = \frac{\mathscr{L}\left(\theta_{t+1}\right)P\left(\theta_{t+1}\right)}{\mathscr{L}\left(\theta_t\right)P\left(\theta_t\right)}$$

Metropolis-Hastings Algorithm

Algorithm to decide weather a new value should be accepted or not, e.g. the Metropolis Hastings Algorithm

$$a = \frac{P(\theta_{t+1}|D,M)}{P(\theta_t|D,M)} \overset{\text{Bayes Theorem}}{=} \frac{\frac{P(D|\theta_{t+1},M)P(\theta_{t+1})}{P(D)}}{\frac{P(D|\theta_t,M)P(\theta_t)}{P(D)}} = \frac{\mathscr{L}(\theta_{t+1})P(\theta_{t+1})}{\mathscr{L}(\theta_t)P(\theta_t)}$$

$$P(D) = \int_\theta P(x,\theta)\,d\theta$$

hard to compute!

Likelihood function (assumption of Gaussian errors):

$$\mathscr{L}(\theta) = \prod_i l_i(\theta) = \prod_i \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma_i^2}}$$

Metropolis-Hastings Algorithm

Algorithm to decide weather a new value should be accepted or not, e.g. the Metropolis Hastings Algorithm

$$a = \frac{P(\theta_{t+1}|D, M)}{P(\theta_t|D, M)} \overset{\text{Bayes Theorem}}{=} \frac{\frac{P(D|\theta_{t+1}, M)P(\theta_{t+1})}{P(D)}}{\frac{P(D|\theta_t, M)P(\theta_t)}{P(D)}} = \frac{\mathscr{L}(\theta_{t+1})P(\theta_{t+1})}{\mathscr{L}(\theta_t)P(\theta_t)}$$

$$\theta_{t+1} = \begin{cases} \theta_{t+1}, & \text{if } a > 1 \\ \theta_t, & \text{otherwise} \end{cases}$$

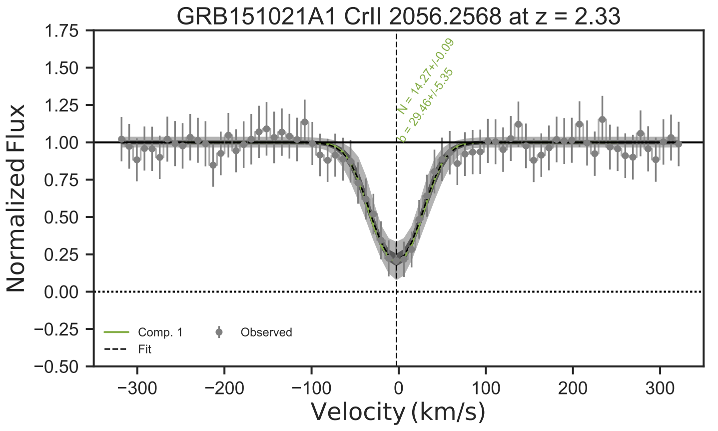
(Animation!)

Outline

Absorption Line Fitting

Fitting Voigt profile to a (normalized) GRB afterglow spectrum

- Voigt Profile: N, b, v_0 + (Continuum, Background)



The Voigt Profile in Python

(in velocity space)

$$F_{\text{model}} = F_{\text{cont}} \cdot e^{-\tau}$$

$$\tau = \frac{\pi e^2}{m_e c} f_{ij} \lambda_{ij} \cdot 10^{\boxed{N}} \cdot \phi_v \left(v - \boxed{v_0}, \frac{\boxed{b}}{\sqrt{2}}, \Gamma \right) (*10e^{-13})$$

The Voigt Profile in Python

(in velocity space)

$$F_{\text{model}} = F_{\text{cont}} \cdot e^{-\tau}$$

$$\tau = \frac{\pi e^2}{m_e c} f_{ij} \lambda_{ij} \cdot 10^{\boxed{N}} \cdot \phi_v \left(v - \boxed{v_0}, \frac{\boxed{b}}{\sqrt{2}}, \Gamma \right) (*10e^{-13})$$

```
def add_abs_velo(v, N, b, gamma, f, l0):
    # Add an absorption line in velocity space
    A = (((np.pi*e**2)/(m_e*c))*f*l0*1E-13) * (10**N)
    tau = A * voigt_profile(v,b/np.sqrt(2.0),gamma)
    return np.exp(-tau)
```

The Voigt Profile in Python

(in velocity space)

```
def add_abs_velo(v, N, b, gamma, f, l0):
    # Add an absorption line in velocity space
    A = (((np.pi*e**2)/(m_e*c))*f*l0*1E-13) * (10**N)
    tau = A * voigt_profile(v,b/np.sqrt(2.0),gamma)
    return np.exp(-tau)

from scipy.special import wofz #Faddeeva function
def voigt_profile(x, sigma, gamma):
    #gamma: HWHM of the Lorentzian profile
    #sigma: the standard deviation of the Gaussian profile
    z = (x + 1j*gamma) / (sigma * np.sqrt(2.0))
    V = wofz(z).real / (sigma * np.sqrt(2.0*np.pi))
    return V
```

Outline

Implementation in PyMC

General Structure

```
import pymc
def model(velocity, flux, flux_err, *args, **kwargs):
    #Priors on unknown parameters
    v0 = pymc.Uniform('v0',lower=-400,upper=400, doc='v0')
    def add_voigt(priors, *args, **kwargs):
        return F * np.exp(-t)
    data = pymc.Normal('y_val',mu=add_voigt, tau=1.0/(
        flux_err**2),value=flux,observed=True) #likelihood
    return locals()

def mcmc(velocity, flux, flux_err):
    MDL = pymc.MCMC(model(velocity, flux, flux_err))
    MDL.sample(iterations=20000, burn_in=15000)

    return fit_parameters
```

Defining the Priors, The Stochastic Class

Variables whose values are not determined by its parents

PyMC Build in distributions:

`https://pymc-devs.github.io/pymc/distributions.html`

```
def model(velocity, flux, flux_err, *args, **kwargs):
    v0 = pymc.Uniform('v0',lower=-400,upper=400, doc='v0')
    N  = pymc.Normal('N',mu=15.0,tau=1.0/(10**2), doc='N')
    b  = pymc.Normal('b',mu=15.0,tau=1.0/(10**2), doc='b')

    ...
```

The Physical Model, The Deterministic class

A variable that is entirely determined by its parents

```
@pymc.deterministic(plot=False) #Deterministic Decorator
def add_voigt(velocity,f,gamma,l0,N,b,v0):

    f = np.ones(len(velocity)) #Background, Continuum
    v = velocity - v0
    f *= add_abs_velo(v, N, b, gamma, f, l0)
    return f

#Data with Gaussian errors, for Likelihood
y_val = pymc.Normal('y_val',mu=add_voigt,tau=1/(flux_err**2),
    value=flux,observed=True)

return locals()
```

Start the MCMC - The MCMC Class

```
def mcmc(velocity, flux, flux_err):

    MDL = pymc.MCMC(model(velocity,fluxv,fluxv_err),
        db='pickle',dbname='results.pickle')

    MDL.db
    MDL.sample(iterations, burn_in)
    MDL.db.close()

    y_min = MDL.stats()['add_voigt']['quantiles'][2.5]
    y_max = MDL.stats()['add_voigt']['quantiles'][97.5]
    y_fit = MDL.stats()['add_voigt']['mean']

    return y_fit, y_min, y_max
```

```
$git clone https://github.com/Jan91/esopy
```