EE18BTECH11026_A6

February 28, 2022

1 KOIDALA SURYA PRAKASH

2 EE18BTECH11026

2.1 Asst 06

[8]:
```
### imports here !!

import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
from scipy import stats
import corner
import emcee
```

3 Q1

[37]:
```
## constants
ein_pred, new_pred = 1.74, 0.87

## eddington vals
ed_val,ed_err = 1.61, 0.4
## crommelin vals
crom_val, crom_err = 1.98, 0.16

## Bayes factors for respective data assuming gaussinan likelihood
ed_bf =  (stats.norm.pdf(ed_val,loc=ein_pred, scale=ed_err)/stats.norm.
 ↪pdf(ed_val,loc=new_pred,scale=ed_err))
crom_bf = (stats.norm.pdf(crom_val,loc=ein_pred, scale=crom_err)/stats.norm.
 ↪pdf(crom_val,loc=new_pred,scale=crom_err))

print('Bayes factor for Eddington : {}'.format(ed_bf))
print('Bayes factor for Crommelin : {}'.format(crom_bf))
```

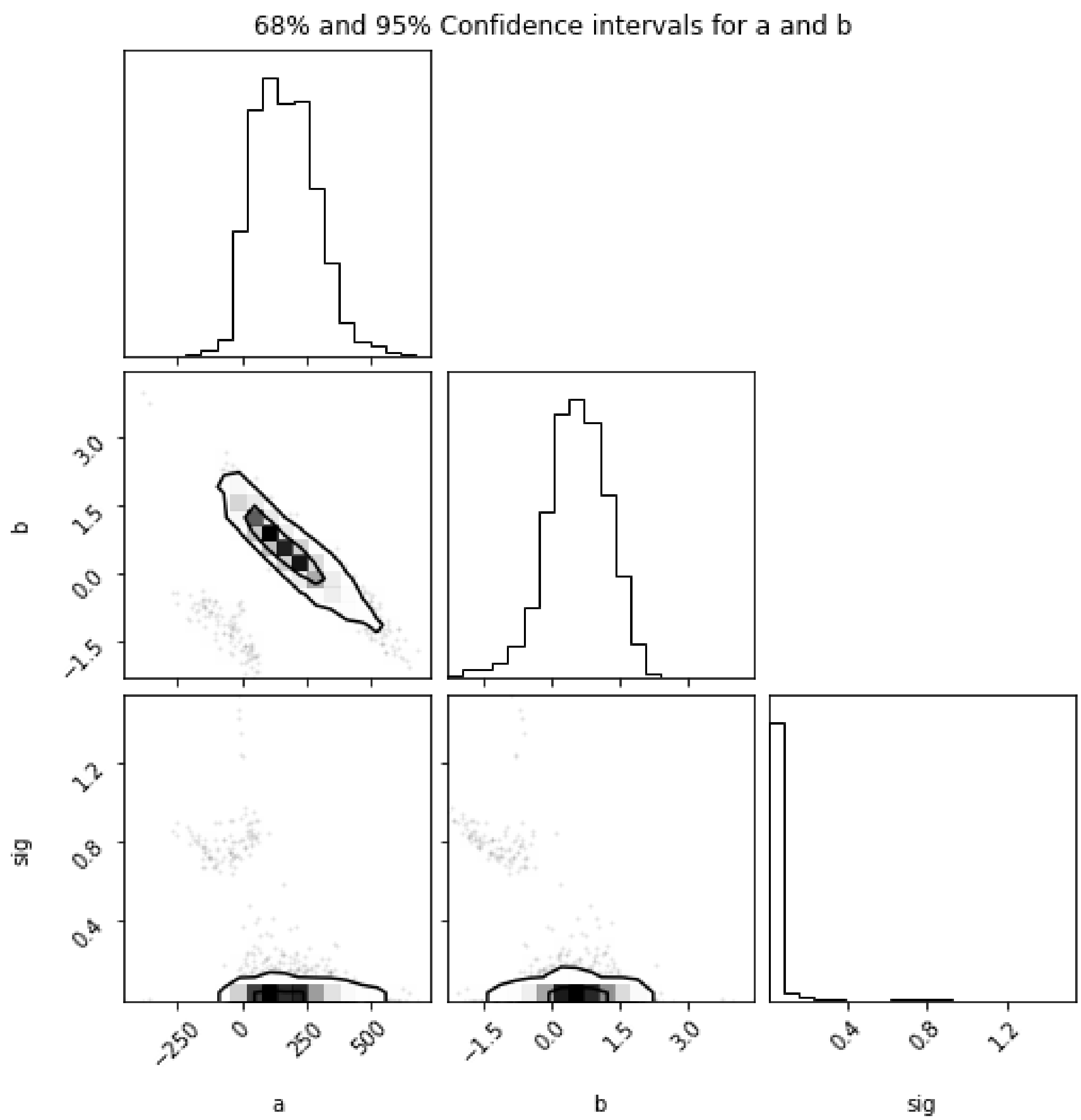
```
print('Strength of evidence is stronger for crommelin when compared to that of␣
↪eddington')
```

```
Bayes factor for Eddington : 5.25109958796716
Bayes factor for Crommelin : 9172292802.960836
Strength of evidence is stronger for crommelin when compared to that of
eddington
```

4 Q2

[25]:
```
'''
num_dim : num of params of the model
nwalkers : no. of MCMC walkers
nburn : burn in period
nsteps : MCMC steps to take
'''
num_dim, num_burn_period, num_steps, num_walkers = 3, 100, 2000, 50

### Data
x_dat =np.array( [203, 58, 210, 202, 198, 158, 165, 201, 157, 131, 166, 160,␣
↪186, 125, 218, 146])
y_dat = np.array([ 495, 173, 479, 504, 510, 416, 393, 442, 317, 311, 400, 337,␣
↪423, 334, 533, 344])
sig_y = np.array([ 21, 15, 27, 14, 30, 16, 14, 25, 52, 16, 34, 31, 42, 26, 16,␣
↪22])

## log post probability
def log_post(theta, x, y, sig):

    c, m, sig = theta
    if(sig >= 0):
        log_prior = -(np.log(sig) + 1.5*np.log(1+pow(m,2)))
    else:
        log_prior = -np.inf

    y_model = (m*x + c)
    sigma_model = pow(sig, 2) + np.exp(2*sig)*pow(y_model, 2)
    log_likeli = -0.5*np.sum((y-y_model)**2 / sigma_model + np.log(sigma_model))
    return log_prior + log_likeli


###### Code
np.random.seed(0)
#initial guesses
starting_guesses = np.random.random((num_walkers, num_dim))
```

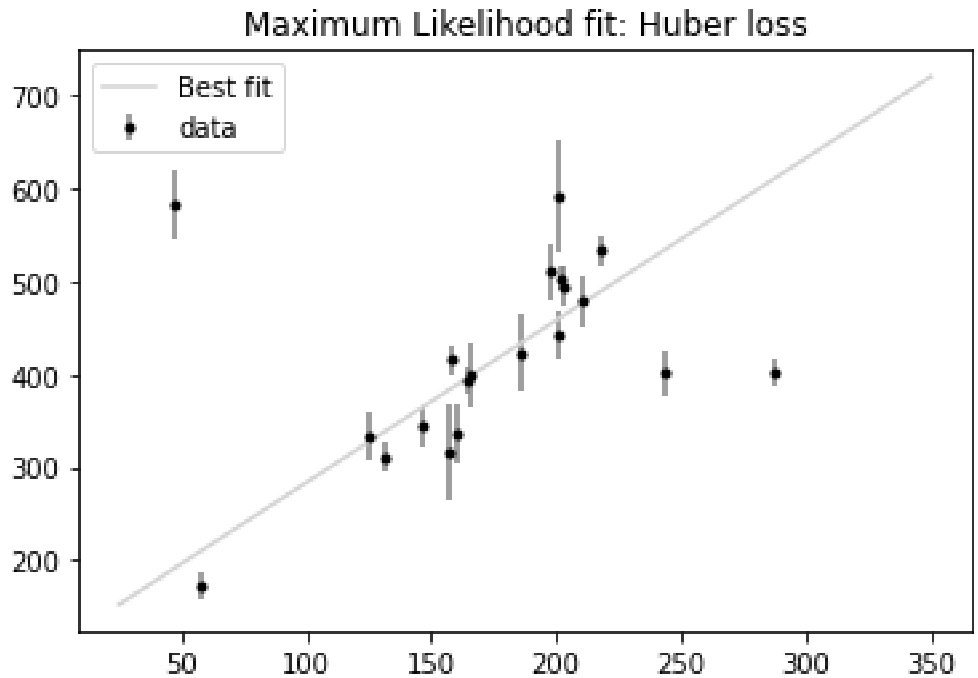
```
sampler = emcee.EnsembleSampler(num_walkers, num_dim, log_post,args=[x_dat,␣
→y_dat, sig_y])
sampler.run_mcmc(starting_guesses, num_steps)

chain = sampler.get_chain(discard=num_burn_period, thin=20, flat=True)
emcee_trace = sampler.chain[:, num_burn_period:, :].reshape(-1, num_dim).T

fig = corner.corner(chain, levels=(0.68,0.95), labels=["a","b","sig"])
fig.suptitle('68% and 95% Confidence intervals for a and b')

plt.show()
```



5 Q3

[39]:
```
### Data

x_dat =np.array( [201, 244, 47, 287, 203, 58, 210, 202, 198, 158, 165, 201,␣
↪157, 131, 166, 160, 186, 125, 218, 146])
y_dat = np.array([592, 401, 583, 402, 495, 173, 479, 504, 510, 416, 393, 442,␣
↪317, 311, 400, 337, 423, 334, 533, 344])
sig_y = np.array([61, 25, 38, 15, 21, 15, 27, 14, 30, 16, 14, 25, 52, 16, 34,␣
↪31, 42, 26, 16, 22])
```

[40]:
```
## Max Likelihood

def huber_loss(t, c=3):
    return ((abs(t) < c) * 0.5 * t ** 2
            + (abs(t) >= c) * -c * (0.5 * c - abs(t)))

def total_huber_loss(theta, x=x_dat, y=y_dat, e=sig_y, c=3):
    return huber_loss((y - theta[0] - (theta[1] * x) ) / e, c).sum()

x = np.linspace(25, 350, 2000)
theta_opt = optimize.fmin(total_huber_loss, [0,0], disp=False)

plt.errorbar(x_dat, y_dat, sig_y, fmt='.k', ecolor='gray', label = 'data')
plt.plot(x, theta_opt[0] + (theta_opt[1]*x), color='lightgray', label = 'Best␣
↪fit')
plt.title('Maximum Likelihood fit: Huber loss')
plt.legend()
plt.show()
```

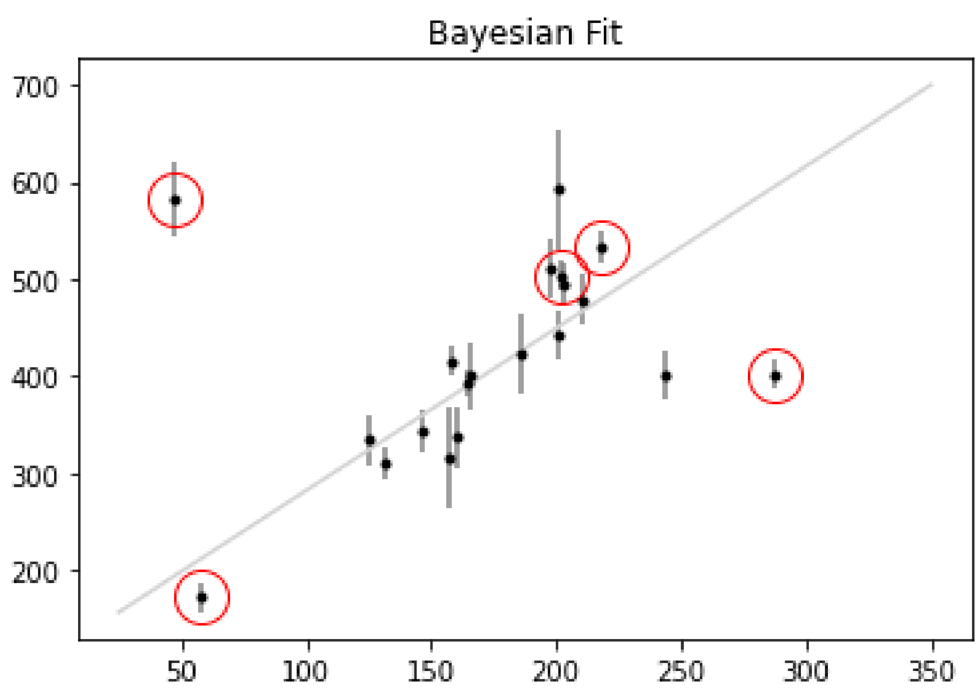


[41]:
```
## Bayesian model

def log_prior(theta):
    #g_i needs to be between 0 and 1
    if (all(theta[2:] > 0) and all(theta[2:] < 1)):
        return 0
    else:
        return -np.inf  # recall log(0) = -inf

def log_likelihood(theta, x, y, e, sigma_B):
    dy = y - theta[0] - theta[1] * x
    g = np.clip(theta[2:], 0, 1)  # g<0 or g>1 leads to NaNs in logarithm
    logL1 = np.log(g) - 0.5 * np.log(2 * np.pi * e ** 2) - 0.5 * (dy / e) ** 2
    logL2 = np.log(1 - g) - 0.5 * np.log(2 * np.pi * sigma_B ** 2) - 0.5 * (dy /
 ↪ sigma_B) ** 2
    return np.sum(np.logaddexp(logL1, logL2))

def log_posterior(theta, x, y, e, sigma_B):
    return log_prior(theta) + log_likelihood(theta, x, y, e, sigma_B)
```

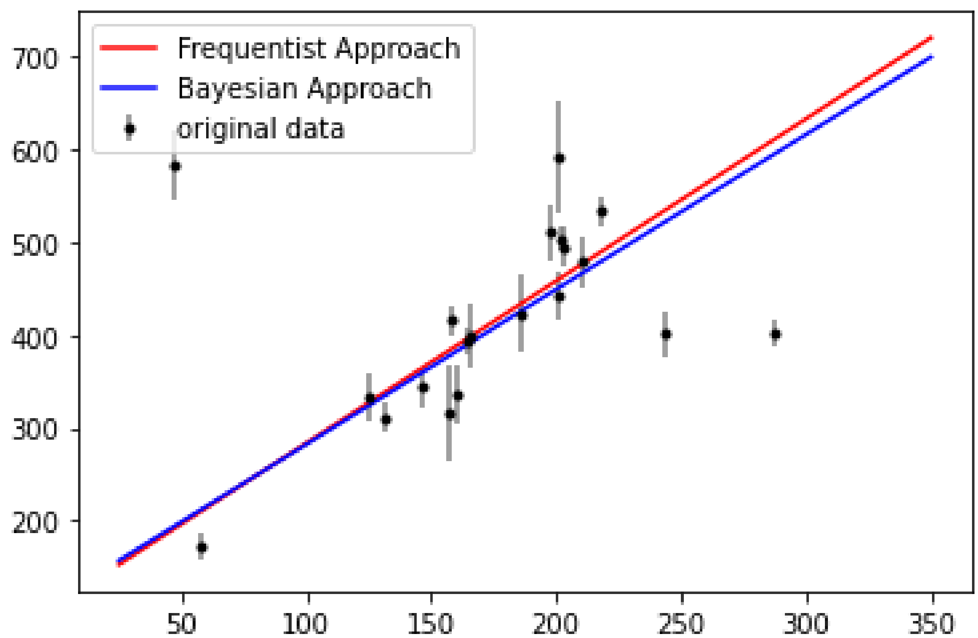
[ ]:
```
ndim = 2 + len(x_dat)  # number of parameters in the model
nwalkers = 50  # number of MCMC walkers
nburn = 10000  # "burn-in" period to let chains stabilize
nsteps = 15000  # number of MCMC steps to take
```

```python
# set theta near the maximum likelihood, with
np.random.seed(0)
starting_guesses = np.zeros((nwalkers, ndim))
starting_guesses[:, :2] = np.random.normal(theta_opt, 1, (nwalkers, 2))
starting_guesses[:, 2:] = np.random.normal(0.5, 0.1, (nwalkers, ndim - 2))

sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=[x_dat,␣
↪y_dat, sig_y, 50])
sampler.run_mcmc(starting_guesses, nsteps)

sample = sampler.chain  # shape = (nwalkers, nsteps, ndim)
sample = sampler.chain[:, nburn:, :].reshape(-1, ndim)
```

[43]:
```python
new_theta = np.mean(sample[:,:2], 0)
g = np.mean(sample[:, 2:], 0)
outliers = (g < 0.41)



plt.errorbar(x_dat, y_dat, sig_y, fmt='.k', ecolor='gray')
plt.plot(x, new_theta[0] + new_theta[1] * x, color='lightgray')
plt.plot(x_dat[outliers], y_dat[outliers], 'go', ms=20, mfc='none', mec='red')
plt.title('Bayesian Fit')
plt.show()
```



[44]:
```
#Plotting the fit obtained from the two approaches
plt.errorbar(x_dat, y_dat, sig_y, fmt='.k', ecolor='gray', label='original␣
 ↪data')
plt.plot(x, theta_opt[0] + theta_opt[1] * x, color='r', label='Frequentist␣
 ↪Approach')
plt.plot(x, new_theta[0] + new_theta[1] * x, color='b',label='Bayesian␣
 ↪Approach')
plt.legend()
# plt.title("")
plt.show()
```



6 THE END