Bayesian Statistical Methods: Homework 2

Due on Feb 14, 2013

Andrew Kurzawski

Problem 1

Reading Assignment.

Problem 2

I began by doing a test run of 1000 iterations with no burn in or thinning. From the traces of μ and σ, I saw that both had converged within a few iterations. So, I burned 100 iterations because running the model is not computationally expensive. Then, I ran the model again and observed significant autocorrelation of μ within about 15 itterations, so I selected to thin the results by 15. Fig. 1 and Fig. 2 are the resulting posterior summaries after 10,000 iterations.

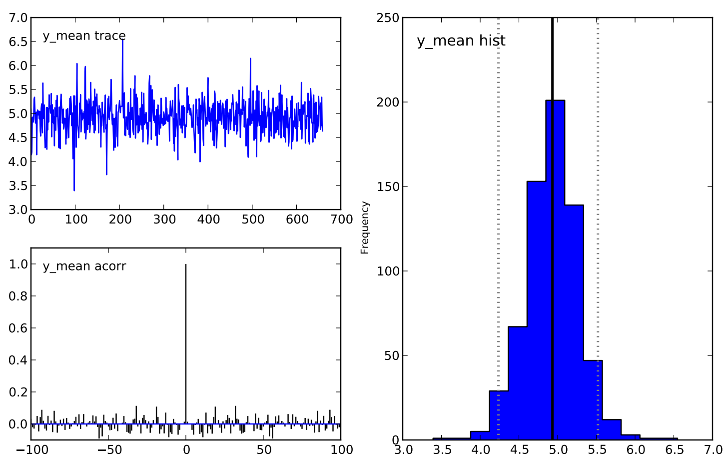


Figure 1: Posterior summary for μ using PyMC

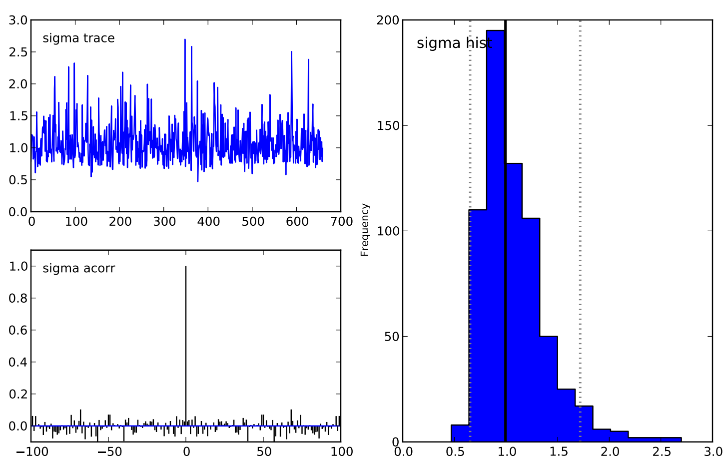


Figure 2: Posterior summary for σ using PyMC

Problem 3

In a standard slice sampler, you would pick a starting value for θ. However, because in this case we are able to sample off of $h_1(\theta)$, we can choose the starting value for our chain by sampling from this distribution. This in turn cuts down on the burn in time. From here, we would proceed with slice sampling as follows:

Use θ_0 generated from $h_1(\theta)$ to calculate $h(\theta_0) = h_1(\theta_0)h_2(\theta_0)$.
Sample U from $Uniform(0, h(\theta_0))$.
Search for the intercepts of U and $h(\theta_0)$, denoted as θ_L and θ_R.
Sample θ_1 from $Uniform(\theta_L, \theta_R)$.
Repeat.

Problem 4

The following code is a Metropolis-Hastings algorithm written in Python for the data given in Problem 2 with $\sigma = 1$. A poster summary for μ is given in Fig. 3 for 10000 iterations. Roughly 10% of the samples were accepted.

```
from __future__ import division
import numpy as np
import scipy.stats
import matplotlib as plt
from pylab import *

def mh_sample(n, data, mu, walk_sig):
    # n is the number of iterations
    # data is what we are conditioning on
    # mu is an initial value for the chain of samples
    # walk_sig is the variance of the random walk

    # Initialize arrays
    x_list = np.zeros(n)
    n_list = np.zeros(n)
    x_list[0] = mu
    n_list[0] = 1

    # Variance is 1 for now
    sigma = 1.0

    # Pointer for last accepted value
    acc = 0

    for i in range(n):
        # Random Walk to generate next value of x
        z = x_list[acc] + np.random.normal(0.0,walk_sig**2,1)

        # Compute Denominator of MH alogrithm
        den = 1
        for j in range(0,data.shape[0]):
            den = den * np.exp((-(data[j]-x_list[acc])**2)/(2.0*sigma**2))
```

```
        # Compute Numerator of MH alogrithm
        num = 1
        for j in range(0,data.shape[0]):
            num = num * np.exp((-(data[j]-z)**2)/2.0)

        # Compute the probability of a move, alpha
        alpha = min(1, num/den)

        # Accept / Reject
        if alpha == 1:
            x_list[acc+1] = z
            n_list[acc+1] = i+1
            acc = acc + 1
        elif alpha < 1:
            p_a = np.random.binomial(1, alpha, 1)
            if p_a == 1:
                x_list[acc+1] = z
                n_list[acc+1] = i+1
                acc = acc + 1

    return x_list, n_list, acc
```

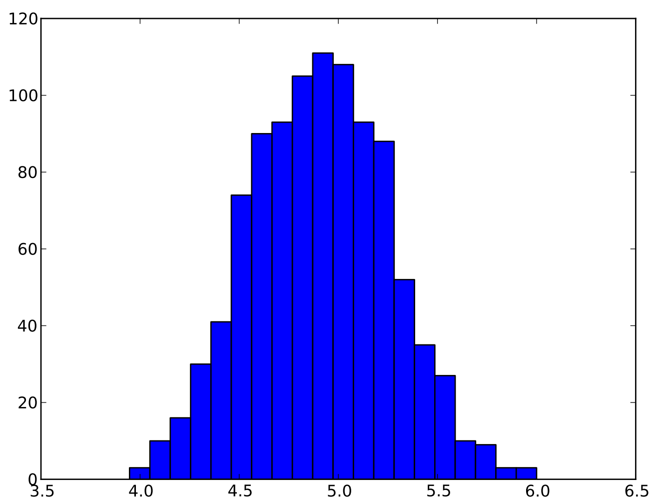


Figure 3: Posterior summary for μ using Metropoils-Hastings

Problem 5

The following code is a Slice Sampling algorithm written in Python for the data given in Problem 2 with $\sigma = 1$. A poster summary for μ is given in Fig. 4 for 10000 iterations.

```
from __future__ import division
import numpy as np
import scipy.stats
import matplotlib as plt
from pylab import *

def slice_sample(n, data, mu):
    # n is the number of iterations
    # data is what we are conditioning on
    # mu is an initial value for the chain of samples

    # Initialize arrays
    x_list = np.zeros(n+1)
    x_list[0] = mu
    sigma = 1.0

    width = 1.0

    for i in range(n):
        # Calculate h(x) at data
        h_x = 1
        for j in range(0,data.shape[0]):
            h_x = h_x * np.exp((-(data[j]-x_list[i])**2)/(2.0*sigma**2))

        # Sample from 0 to h(x) to get U
        U = np.random.uniform(0, h_x, 1)

        # Find bounds x_L and x_R
        x_L = x_list[i] - width
        x_R = x_list[i] + width
        left_bound = False
        right_bound = False
        while left_bound == False:
            h_x_L = 1.0
            for j in range(0,data.shape[0]):
                h_x_L = h_x_L * np.exp((-(data[j]-x_L)**2)/(2.0*sigma**2))
            if h_x_L < U:
                left_bound = True
            elif h_x_L > U:
                x_L = x_L - width

        while right_bound == False:
            h_x_R = 1.0
            for j in range(0,data.shape[0]):
                h_x_R = h_x_R * np.exp((-(data[j]-x_R)**2)/(2.0*sigma**2))
            if h_x_R < U:
                right_bound = True
            elif h_x_R > U:
                x_R = x_R + width
```

```
        # Sample from uniform on interval x_L to x_R
        x_list[i+1] = np.random.uniform(x_L, x_R, 1)

    return x_list
```

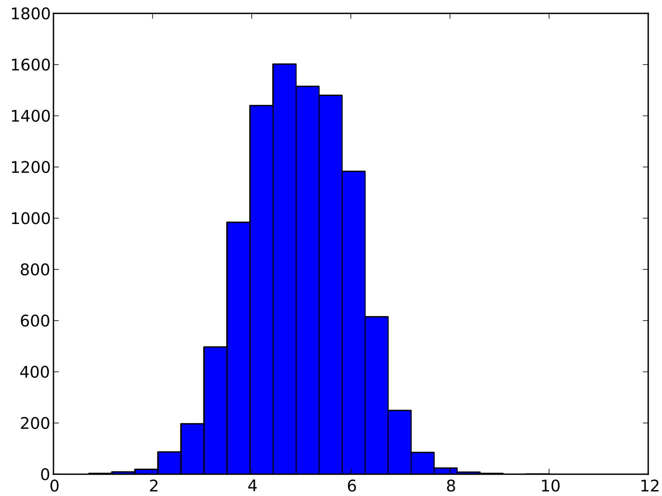


Figure 4: Posterior summary for μ using Slice Sampling

Problem 6

I used the same methods stated in Problem 2 to choose the Burn In and Thinning for each case. The main differences to take note of between the posteriors (Fig. 5-7) are that the means sampled from the Informed and Uniform priors are both slightly higher than that of the diffuse Gamma prior. Additionally, the posterior generated from the Uniform prior has a larger tail on the right hand side, due to the fact that the Uniform prior was given the range from 0 to 20.

Table 1: Comparison of Specified Priors

Prior	Iterations	Burn In	Thinning
Diffuse Gamma(0.001,0.001)	50,000	10,000	20
Informed Gamma(1.11,1.61)	30,000	10,000	20
Uniform(0,20)	30,000	4,000	15

The effect of assigning a Uniform prior becomes much more apparent when we examine the probability that θ is greater than 0.5 (Tab. 2).

Table 2: Probablity that θ is greater than 0.5

Prior	Probability
Diffuse Gamma(0.001,0.001)	0.0065
Informed Gamma(1.11,1.61)	0.006
Uniform(0,20)	0.0179

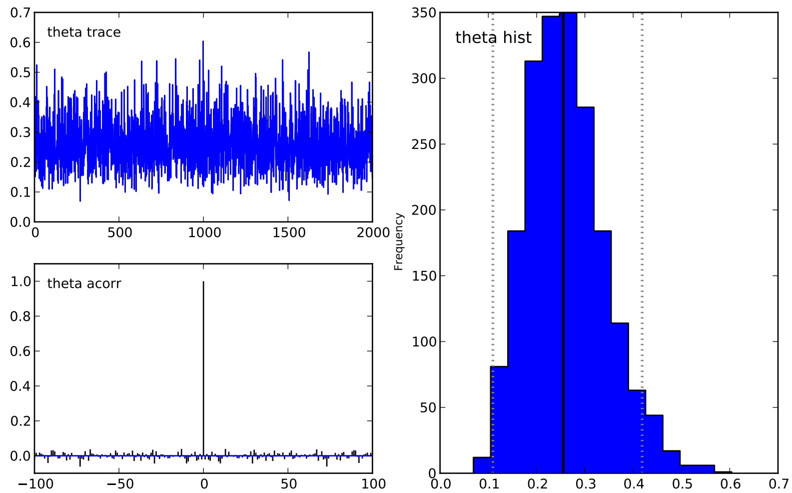


Figure 5: Posterior summary for θ using a diffuse gamma prior

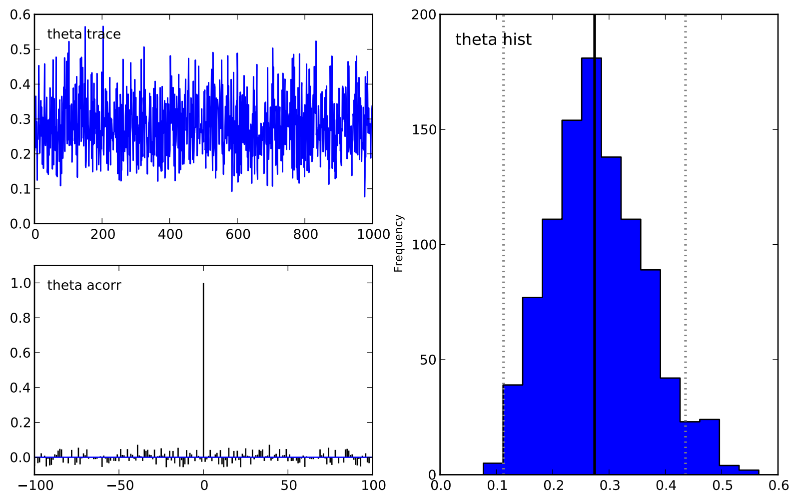


Figure 6: Posterior summary for θ using an informed gamma prior

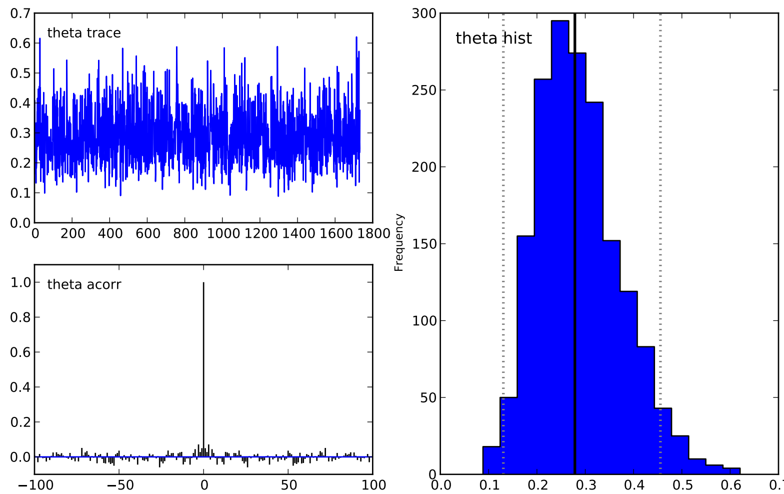


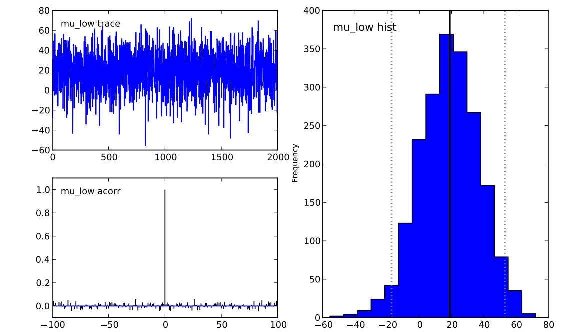
Figure 7: Posterior summary for θ using a uniform prior

Problem 7

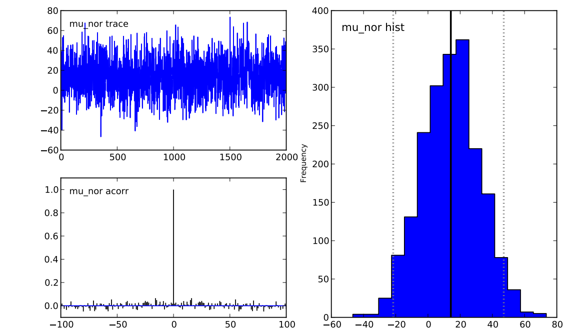
Once again, Burn In and Thinning were determined in the same fashion as Problem 2, with the main difference being that I had to examine multiple posterior distributions for convergence and autocorrelation. Upon looking at the means of μ, I realized that the multivariate normal distribution was not working properly. I am probably specifying something incorrectly. Including any further analysis at this point would be useless, however I will run the model again in my free time once I figure out how to properly implement a multivariate normal distribution in PyMC.

Table 3: Burn In and Thinning for Informed and Uninformed Priors

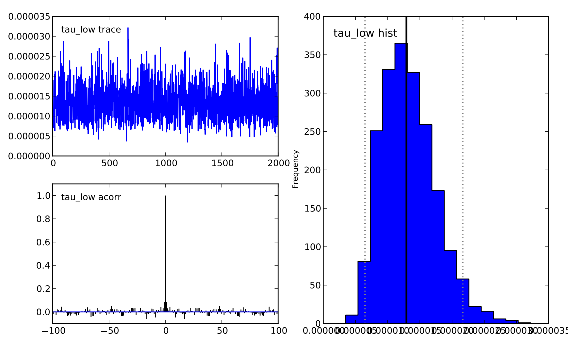
Priors	Burn In	Thinning
Informed	20,000	15
Uninformed	10,000	25



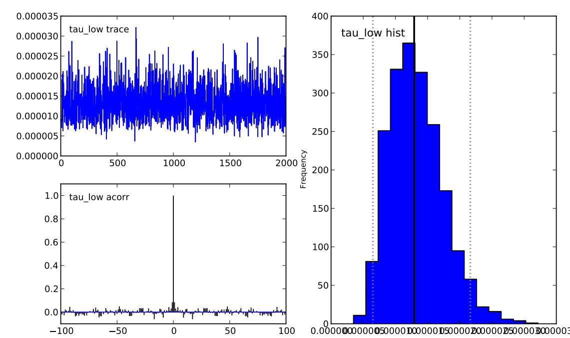
μ_1



μ_2

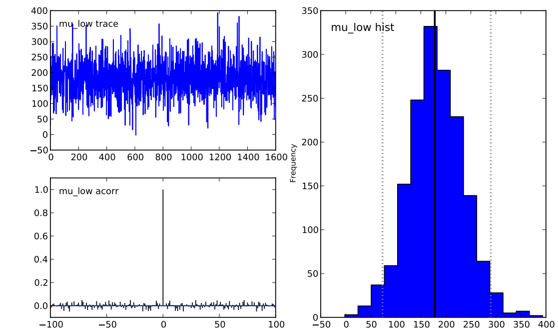


τ_1

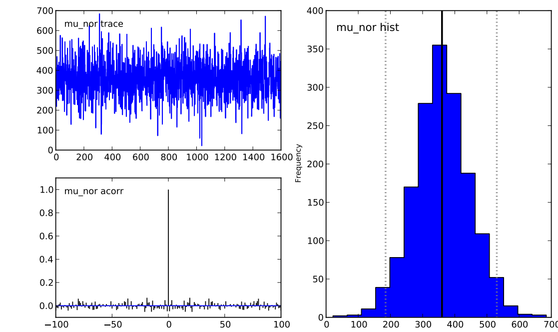


τ_2

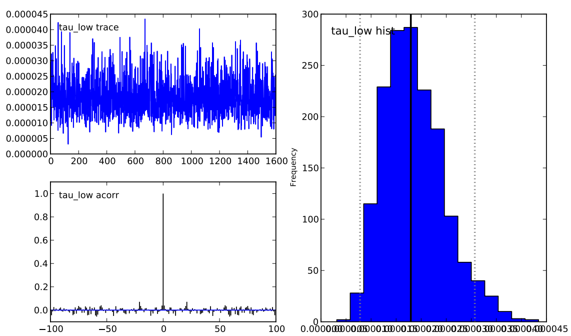
Table 4: Posteriors with Informed Priors



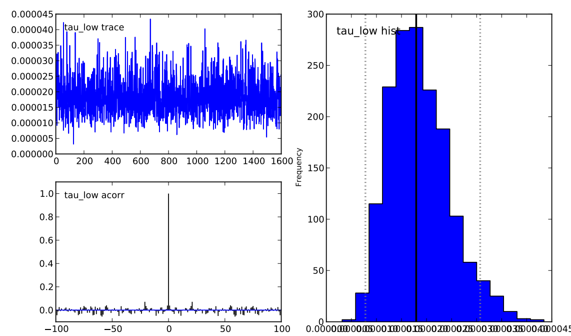
μ_1



μ_2



τ_1



τ_2

Table 5: Posteriors with Uninformed Priors