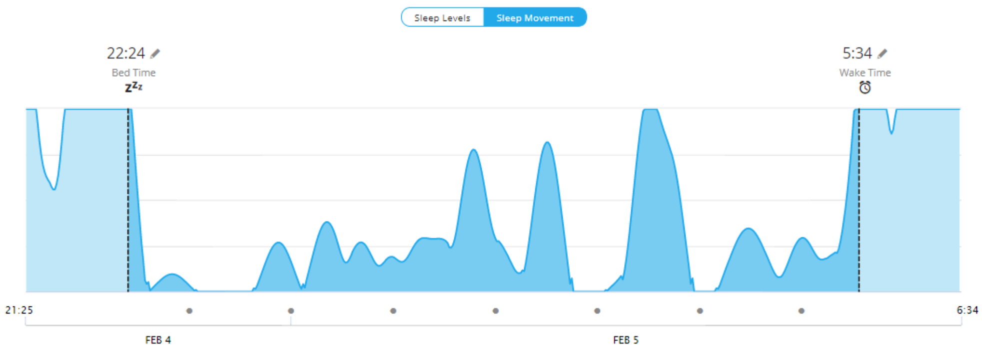
Bayesian Inference on Sleep Data

Will Koehrsen

February 5, 2018

Contents

1 Problem Description



My Garmin Vivosmart tracks the time I fall asleep and wake up each day. Using this data, my first objective is to create a model that returns the probability I am asleep at a given time. Once we have a basic model that calculates the sleep probabilty based on time, we can make it more accurate by including additional information such as if my bedroom light is on or if my phone is charging. This information will be incorporated in a Bayesian method, updating our posterior estimate based on further evidence. The final goal is a model that determines the posterior probabilty I am asleep at a given time with the knowledge of my bedroom light and phone status. Mathematically, this is expressed as:

$$P(\text{sleep}|\text{time}, \text{bedroom light}, \text{phone charging})$$

.

1.1 Approach

The general method is as follows, with additional details provided in the respective sections.

1. Format the data (done in separate notebook) and visualize
2. Choose function to represent posterior probabilty of sleep given the time
3. Use Markov Chain Monte Carlo and the data to find most likely parameters for the selected posterior distribution
4. Use the posterior probability of sleep at a given time as the prior for Bayesian Inference using additional data
5. Build a model to perform Bayesian Inference and find the probabilty of sleep given the time, light condition, and phone charging info
6. Interpret and visualize model

I will treat the falling asleep and waking up data separately and build a model for each. I make extensive use of the PyMC3 library in this report.

2 Wake and Sleep Data Exploration

The wake and sleep data contains a little more than two months of information. The Garmin tries to record when I fall asleep and wake up based on motion and heart rate. It is not 100% accurate and often will think I'm sleeping if I turn off notifications and am quietly reading in bed. Sometimes we have to deal with imperfect data, and there are more truthful observations than false observations, so we can expect the correct data to have a larger effect on the model.

First, we will import the libraries, and then we can visual both the sleep data and the waking data.

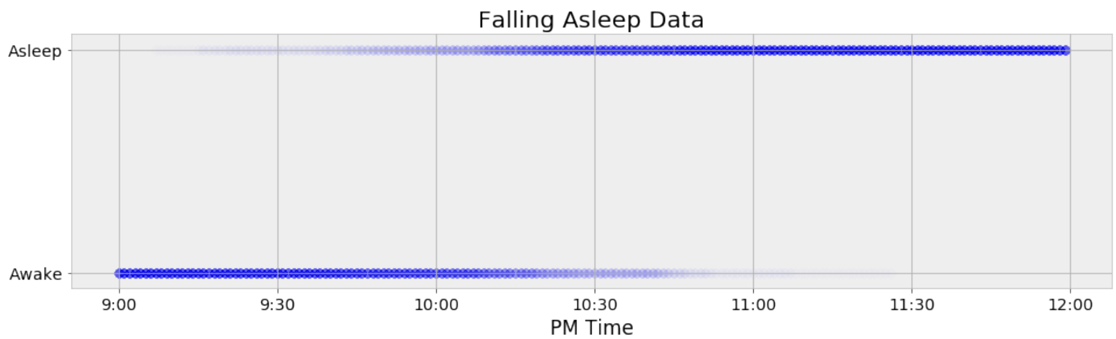
In [56]:
```
# pandas and numpy for data manipulation
import pandas as pd
import numpy as np

# scipy for algorithms
import scipy
from scipy import stats

# pymc3 for Bayesian Inference, pymc built on t
import pymc3 as pm
import theano.tensor as tt
import scipy

# matplotlib for plotting
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.core.pylabtools import figsize
import matplotlib

import json
s = json.load(open('style/bmh_matplotlibrc.json'))
matplotlib.rcParams.update(s)
matplotlib.rcParams['figure.figsize'] = (10, 3)
matplotlib.rcParams['font.size'] = 14

# Number of samples for Markov Chain Monte Carlo
N_SAMPLES = 5000
```

In [57]:
```
# Data formatted in other notebook
sleep_data = pd.read_csv('data/sleep_data.csv')
wake_data = pd.read_csv('data/wake_data.csv')

# Labels for plotting
sleep_labels = ['9:00', '9:30', '10:00', '10:30', '11:00', '11:30', '12:00']
wake_labels = ['5:00', '5:30', '6:00', '6:30', '7:00', '7:30', '8:00']
```

2.1 Falling Asleep Data

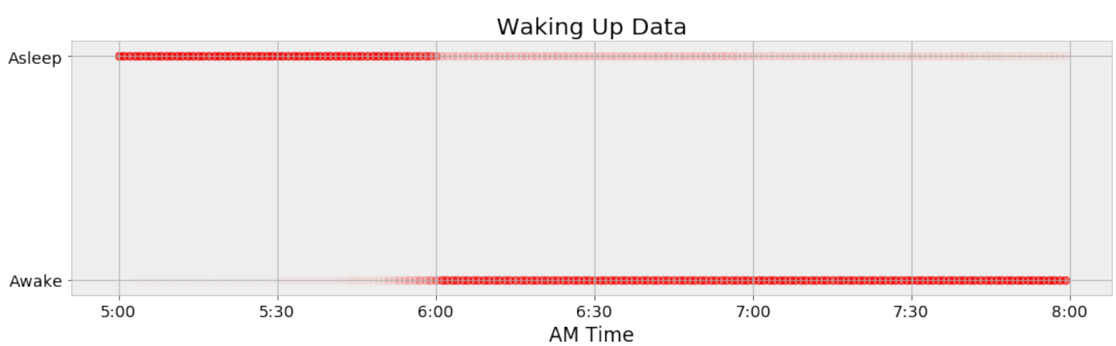
Each dot represents one observations. We can see that I tend to fall asleep a little after 10:00 PM and wake up around 6:00 AM.

In [59]:
```
figsize(16, 4)

# Sleep data
plt.scatter(sleep_data['time_offset'], sleep_data['indicator'],
            s= 60, alpha=0.01, facecolor = 'b', edgecolors='b')
plt.yticks([0, 1], ['Awake', 'Asleep']); plt.xlabel('PM Time');
plt.title('Falling Asleep Data')
plt.xticks([-60, -30, 0, 30, 60, 90, 120], sleep_labels);
```



2.2 Waking Up Data

In [60]:
```
# Wake data
plt.scatter(wake_data['time_offset'], wake_data['indicator'],
            s= 50, alpha = 0.01, facecolor='r', edgecolors =  'r');
plt.yticks([0, 1], ['Awake', 'Asleep']); plt.xlabel('AM Time');
plt.title('Waking Up Data')
plt.xticks([-60, -30, 0, 30, 60, 90, 120], wake_labels);
```



3 Logistic Function to Represent Transition

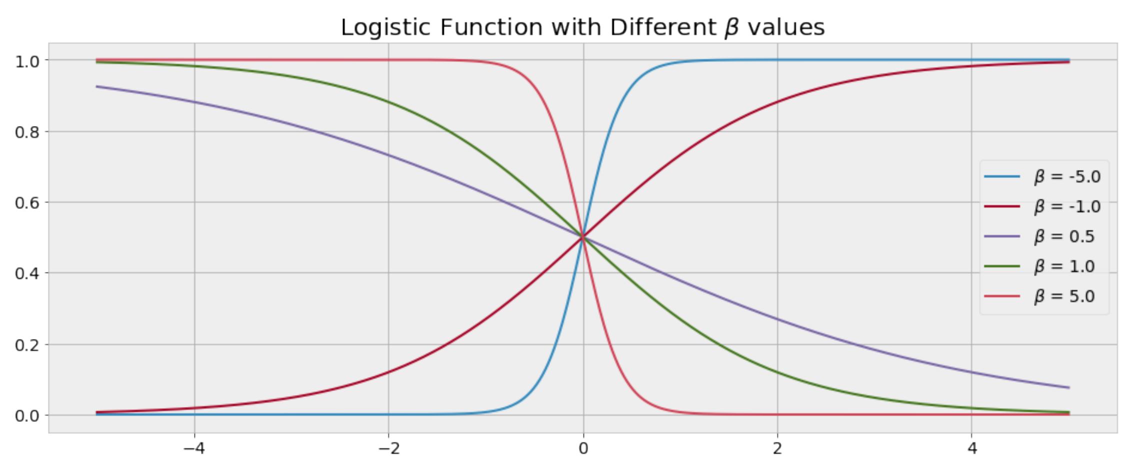
We now need to decide on a function to represent the probability of being asleep at a given time. We will assume this transition can be modeled as a logistic function. A logistic function (also called a sigmoid) is a non-linear function bounded between 0 and 1. The expression for a logistic probability distribution as a function of time is:

$$p(t) = \frac{1}{1 + e^{\beta t}}$$

The β parameter is unknown and will be found using Markov Chain Monte Carlo (MCMC) sampling. MCMC samples from the posterior and tries to to maximize the probabilty of the parameter given the data.

Several logistic functions with various β parameters are shown below:

```
In [61]: figsize(16, 6)

         # Logistic function with only beta
         def logistic(x, beta):
             return 1. / (1. + np.exp(beta * x))

         # Plot examples with different betas
         x = np.linspace(-5, 5, 1000)
         for beta in [-5, -1, 0.5, 1, 5]:
             plt.plot(x, logistic(x, beta), label = r"$\beta$ = %.1f" % beta)

         plt.legend();
         plt.title(r'Logistic Function with Different $\beta$ values');
```



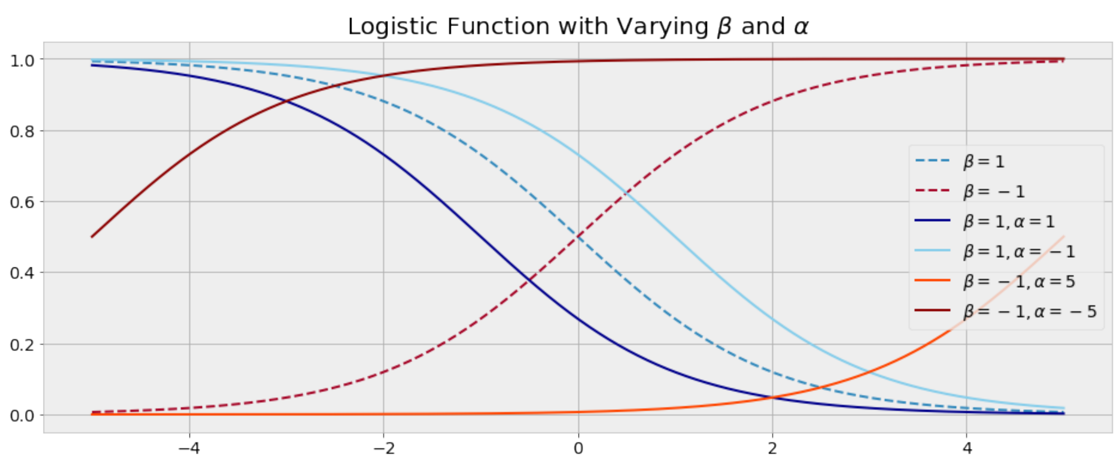
There is one problem with the basic logistic function: the transition is not centered at 0. Instead, it is around 10:00 pm for sleeping and 6:00 am for waking. We adress this by adding an offset, called a bias, to adjust the location of the logistic function. The logistic function is now:

$$p(t) = \frac{1}{1 + e^{\beta t + \alpha}}$$

This introduces another unknown parameter, α, which we will also find from Markov Chain Monte Carlo, again sampling from the posterior to maximize the likelihood of the parameter given the data.

The logistic function with various α and β parameters is shown below.

In [62]:
```
# Logistic function with both beta and alpha
def logistic(x, beta, alpha=0):
    return 1.0 / (1.0 + np.exp(np.dot(beta, x) + alpha))

x = np.linspace(-5, 5, 1000)

plt.plot(x, logistic(x, beta=1), label=r"$\beta = 1$", ls="--", lw=2)
plt.plot(x, logistic(x, beta=-1), label=r"$\beta = -1$", ls="--", lw=2)

plt.plot(x, logistic(x, 1, 1),
         label=r"$\beta = 1, \alpha = 1$", color="darkblue")
plt.plot(x, logistic(x, 1, -1),
         label=r"$\beta = 1, \alpha = -1$",color="skyblue")
plt.plot(x, logistic(x, -1, 5),
         label=r"$\beta = -1, \alpha = 5$", color="orangered")
plt.plot(x, logistic(x, -1, -5),
         label=r"$\beta = -1, \alpha = -5$", color="darkred")
plt.legend();
plt.title(r'Logistic Function with Varying $\beta$ and $\alpha$');
```



β shifts the direction and steepness of the curve, while α changes the location. We will need to use the data to find the most likely value of these parameters. The most likely parameters can then be used in the logistic function to derive a posterior probabilty for being asleep at a given time.

4 Prior Distribution for β and α

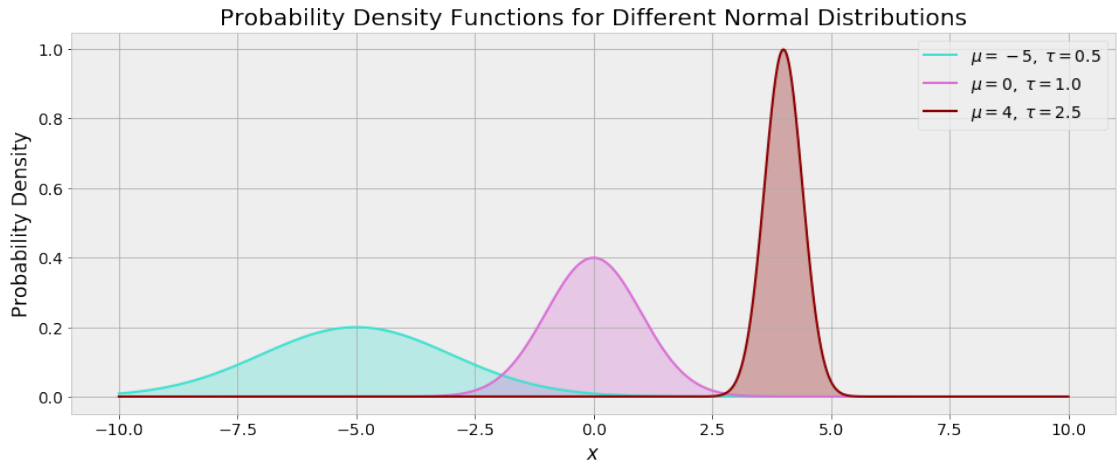
We have no idea what the prior distributions for the model parameters β and α are ahead of time. Therefore, we can model them as if they came from a normal distribution. The normal, or Gaussian, distribution is defined by the mean, μ, and the precision, τ. The precision is the reciprocal of the standard deviation, σ. The mean defines the location of the distribution and the precision shows the spread. A larger value of τ indicates the data is less spread out (it is more precise) and hence the variation is smaller. The mean can be either positive or negative, but the precision will always be positive. A normal distribution is represented as:

$$f(x|\mu,\tau) = \sqrt{\frac{\tau}{2\pi}} \exp\left(-\frac{\tau}{2}(x-\mu)^2\right)$$

When we perform MCMC, we tell the model to sample β and α from a normal distribution with a μ and τ that we can set ahead of modeling.

Probability density functions for three normal distributions are shown below.

```
In [63]: # Set up the plotting parameters
         nor = stats.norm
         x= np.linspace(-10, 10, 1000)
         mu = (-5, 0, 4)
         tau = (0.5, 1, 2.5)
         colors = ("turquoise", "orchid", "darkred")

         # Plot 3 pdfs for different normal distributions
         params = zip(mu, tau, colors)
         for param in params:
             y = nor.pdf(x, loc = param[0], scale = 1 / param[1])
             plt.plot(x, y,
                      label="$\mu = %d,\;\\tau = %.1f$" % (param[0], param[1]),
                      color = param[2])
             plt.fill_between(x, y, color = param[2], alpha = 0.3)

         plt.legend();
         plt.xlabel("$x$")
         plt.ylabel("Probability Density")
         plt.title("Probability Density Functions for Normal Distributions");
```



The expected value of a normal distribution is the mean.

$$E[X|\mu, \tau] = \mu$$

The variance of a normal distribution equal to:

$$Var[X|\mu, \tau) = \frac{1}{\tau}$$

We have no assumptions about the value for either μ or τ in the priors for α and β. When we initialize the model, we can use $\mu = 0$ and a relatively large variance such as $\tau = 0.05$

5 Creating the Posterior Model

We have all the pieces and it is time to put them together. The logistic function describes the transition from awake to asleep, but we do not konw the parameters β and α. The aim is to find the parameters of the logistic function which maximize the likelihood of the observed data. The parameters are assumed to come from a normal distribution defined by a mean, $\mu = 0.0$ and a variance, $\tau = 0.05$. The data is connected to the parameters through a Bernoulli Variable

5.1 Bernoulli Variable

A bernoulli variable is a discrete random variable that is either 0 or 1. In our example, we can model asleep or awake as a Bernoulli variable where awake is 0 and asleep is 1. The variable is a function of the time, so we need a model that outputs a probability for being asleep given the time. The final model looks like

$$\text{Sleep Probability, } S_i \sim \text{Ber}(\ p(t_i)\ ),\quad i = 1..N$$

$p(t_i)$ is the logistic function with the independent variable time, so this becomes:

$$P(\text{sleep}|t_i) = \text{Ber}(\frac{1}{1+e^{(\beta t_i+\alpha)}})$$

Again, we have to find the β and α parameters using MCMC and assuming they come from a normal distribution.

5.1.1 PyMC3 Model

We are using a powerful Bayesian Inference library in Python called PyMC3. This library has features for running Markov Chain Monte Carlo and other inference algorithms. This report does not detail PyMC3, but a great book for getting started is *Probabilistic Programming and Bayesian Methods for Hackers* by Cameron Davidson-Pilon which is available for free on GitHub

The following code creates the model and performs MCMC, drawing `N_SAMPLES` number of samples from the posterior for β and α. The specific sampling algorithm is Metropolic Hastings. We feed in the observed data to the Bernoulli variable in order to maximize the likeilood of the β and α parameters from the data.

```
In [64]: # Sort the values by time offset
         sleep_data.sort_values('time_offset', inplace=True)

         # Time is the time offset
         time = np.array(sleep_data.loc[:, 'time_offset'])

         # Observations are the indicator
         sleep_obs = np.array(sleep_data.loc[:, 'indicator'])
```

```
In [65]: with pm.Model() as sleep_model:
             # Create the alpha and beta parameters
             alpha = pm.Normal('alpha', mu=0.0, tau=0.05, testval=0.0)
             beta = pm.Normal('beta', mu=0.0, tau=0.05, testval=0.0)

             # Create the probability from the logistic function
             p = pm.Deterministic('p', 1. / (1. + tt.exp(beta * time + alpha)))

             # Create the bernoulli parameter which uses the observed data
             observed = pm.Bernoulli('obs', p, observed=sleep_obs)

             # Starting values are found through Maximum A Posterior estimation
             # start = pm.find_MAP()

             # Using Metropolis Hastings Sampling
             step = pm.Metropolis()
```

```
        # Sample from the posterior using the sampling method
        sleep_trace = pm.sample(N_SAMPLES, step=step);
```

```
Multiprocess sampling (2 chains in 2 jobs)
CompoundStep
>Metropolis: [beta]
>Metropolis: [alpha]
The number of effective samples is smaller than 10% for some parameters.
```

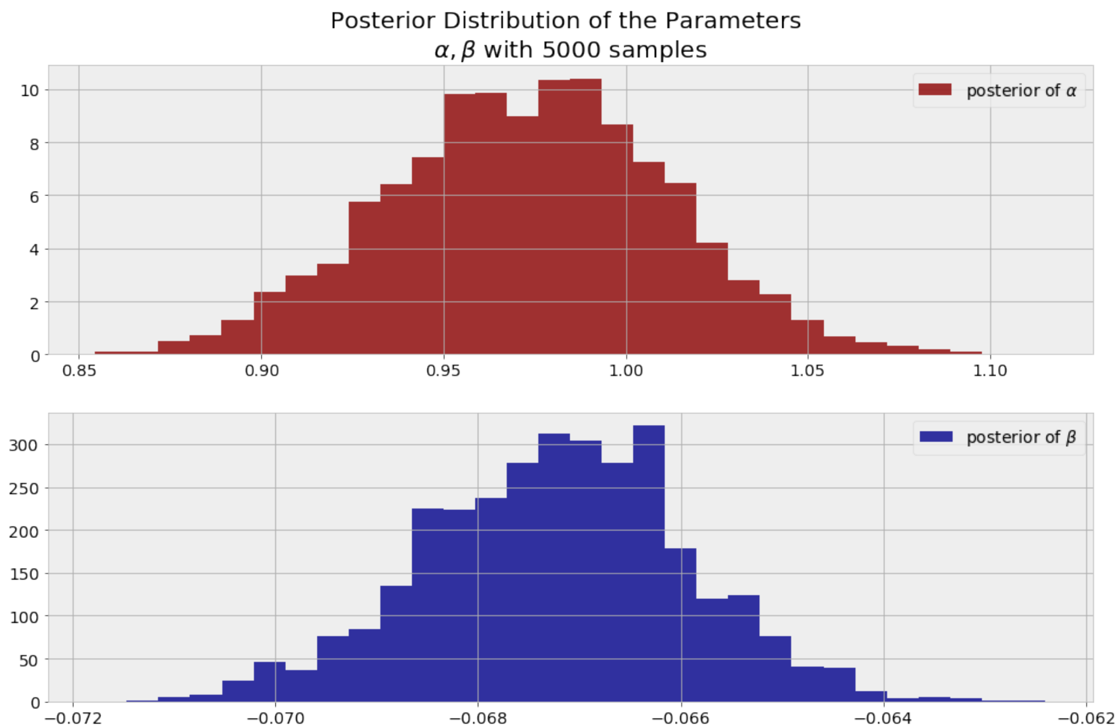
The `trace` variable contains all of the samples drawn from the posterior for β and α. We can graph these samples to explore how they change over the course of sampling. The idea of MCMC is that the samples will get more likely given the data as the algorithm continues. Therefore, we expect the latter values drawn from the posterior to be more accurate than the earlier values. In Markov Chain Monte Carlo, it is common practice to discard a portion of the samples, usually about 50%, which are known as the burn-in samples. For this report I am not discarding any samples, but in a real application, we would run the model for many more steps and discard the initial samples.

5.2 Visualize Posteriors for β and α

The values returned in the `trace` are all the samples drawn from the posterior. They do not represent the true posterior for the parameters, but an approximation. We can visually inspect these values in histograms.

```
In [66]: # Extract the alpha and beta samples
         # Currently using all, including the burn-in period
         alpha_samples = sleep_trace["alpha"][:, None]
         beta_samples = sleep_trace["beta"][:, None]
```

```
In [67]: figsize(16, 10)

         plt.subplot(211)
         plt.title(r"""Posterior Distribution of the Parameters
         $\alpha, \beta$ with %d samples""" % N_SAMPLES)

         plt.hist(alpha_samples, histtype='stepfilled',
                  color = 'darkred', bins=30, alpha=0.8,
                  label=r"posterior of $\alpha$", normed=True);
         plt.legend()

         plt.subplot(212)

         plt.hist(beta_samples, histtype='stepfilled',
                  color = 'darkblue', bins=30, alpha=0.8,
                  label=r"posterior of $\beta$", normed=True)
         plt.legend();
```



If the β values were centered around 0 that would indicate the time has no effect on the probability of being asleep. The α values also are not at 0, indicating that there is an offset from 10:00 pm in terms of being asleep.

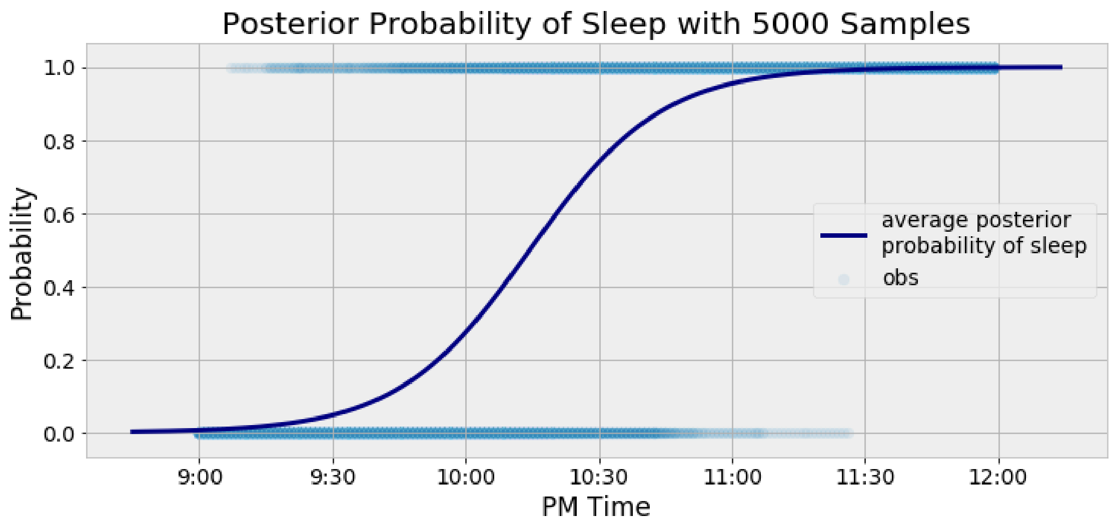
The spread of the data gives us a measure of uncertainty about the data. A larger spread indicates more uncertainty. As there is some overlap between awake and asleep, the uncertainty is expected to be large.

To show the results of the model, we can create the logistic function with the most likely values of α and β which we can take to be the mean of the posterior samples.

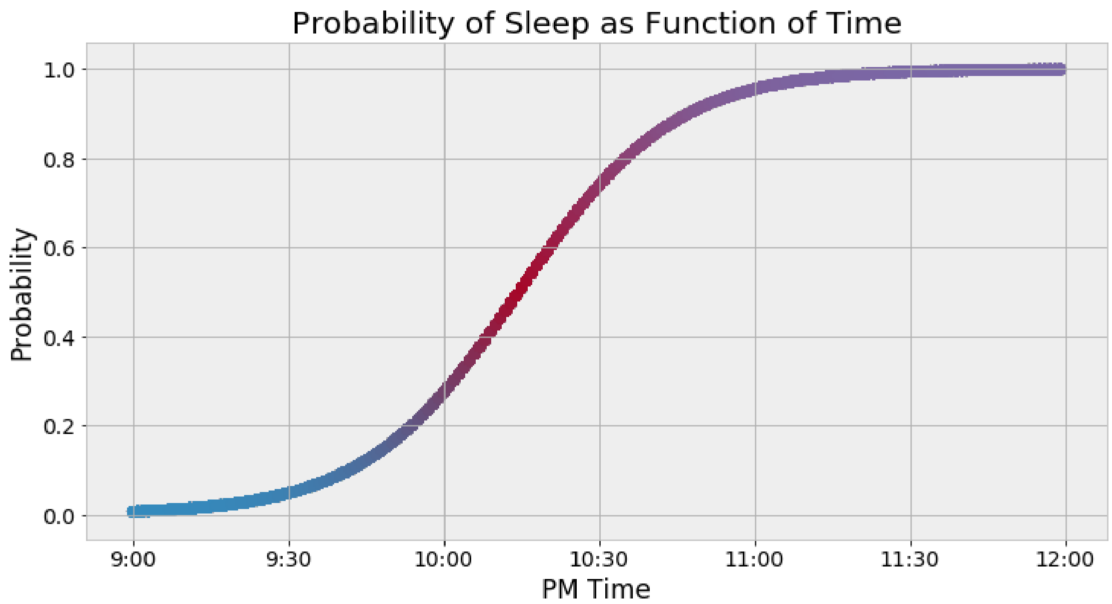
In [68]:
```
# Time values for probability prediction
time_est = np.linspace(time.min()- 15, time.max() + 15, 1e3)[:, None]

# Take most likely parameters to be mean values
alpha_est = alpha_samples.mean()
beta_est = beta_samples.mean()

# Probability at each time using mean values of alpha and beta
sleep_est = logistic(time_est, beta=beta_est, alpha=alpha_est)
```

In [69]:
```
figsize(12, 5)

plt.plot(time_est, sleep_est, color = 'navy',
         lw=3, label="average posterior \nprobability of sleep")
```

```
plt.scatter(time, sleep_obs, edgecolor = 'skyblue',
            s=50, alpha=0.1, label='obs')
plt.title('Posterior Probability of Sleep with %d Samples' % N_SAMPLES);
plt.legend(prop={'size':14})
plt.ylabel('Probability')
plt.xlabel('PM Time');
plt.xticks([-60, -30, 0, 30, 60, 90, 120], sleep_labels);
```




```
In [70]: colors = ["#348ABD", "#A60628", "#7A68A6"]
         cmap = matplotlib.colors.LinearSegmentedColormap.from_list("BMH", colors)
         figsize(12, 6)
         probs = sleep_trace['p']

         plt.scatter(time, probs.mean(axis=0), cmap = cmap,
                     c = probs.mean(axis=0), s = 50);
         plt.title('Probability of Sleep as Function of Time')
         plt.xlabel('PM Time');
         plt.ylabel('Probability');
         plt.xticks([-60, -30, 0, 30, 60, 90, 120], sleep_labels);
```



This is helpful, and now we can pass in any time and receive an estimated probability that I am asleep. The time is passed in as an offset from 10:00 pm.

In [71]:
```
print('10:00 PM probability of being asleep: {:.2f}%.'.
      format(100 * logistic(0, beta_est, alpha_est)))
print('9:30  PM probability of being asleep: {:.2f}%.'.
      format(100 * logistic(-30, beta_est, alpha_est)))
print('10:30 PM probability of being asleep: {:.2f}%.'.
      format(100 * logistic(30, beta_est, alpha_est)))
```

```
10:00 PM probability of being asleep: 27.42%.
9:30  PM probability of being asleep: 4.79%.
10:30 PM probability of being asleep: 73.95%.
```

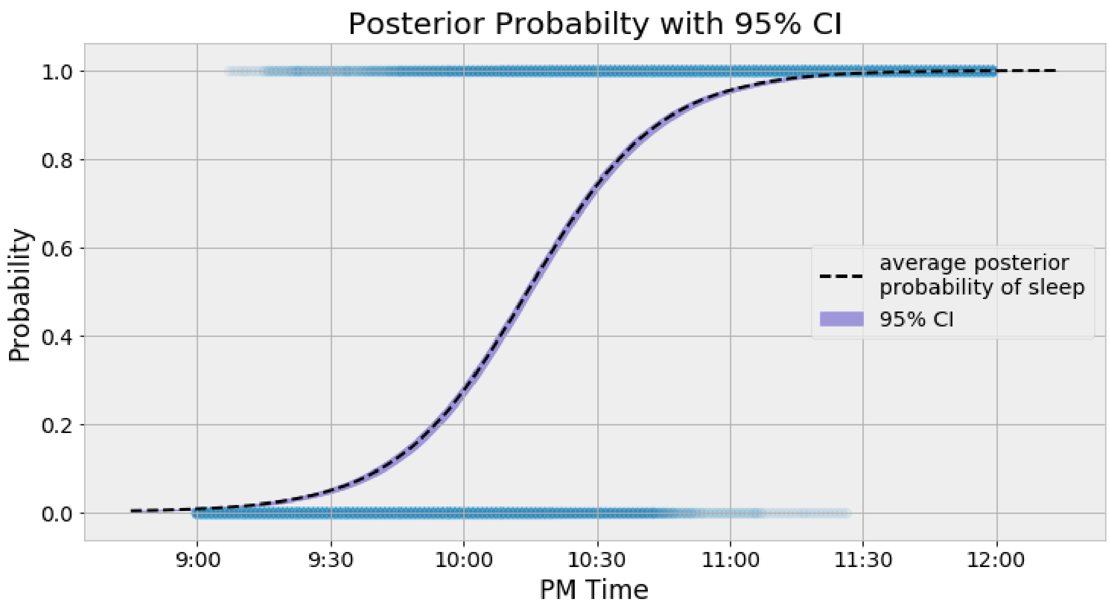
5.2.1 Confidence Interval

There are many other diagnostics of the model that we can perform. For example, we know there is a considerable amount of uncertainty in our parameter estimates. A better way to show the graph therefore would be to include the 95% confidence interval at each time.

In [72]:
```
sleep_all_est = logistic(time_est.T, beta_samples, alpha_samples)
quantiles = stats.mstats.mquantiles(sleep_all_est, [0.025, 0.975], axis=0)
```

In [73]:
```
plt.fill_between(time_est[:, 0], *quantiles, alpha=0.6,
                 color='slateblue', label = '95% CI')
```

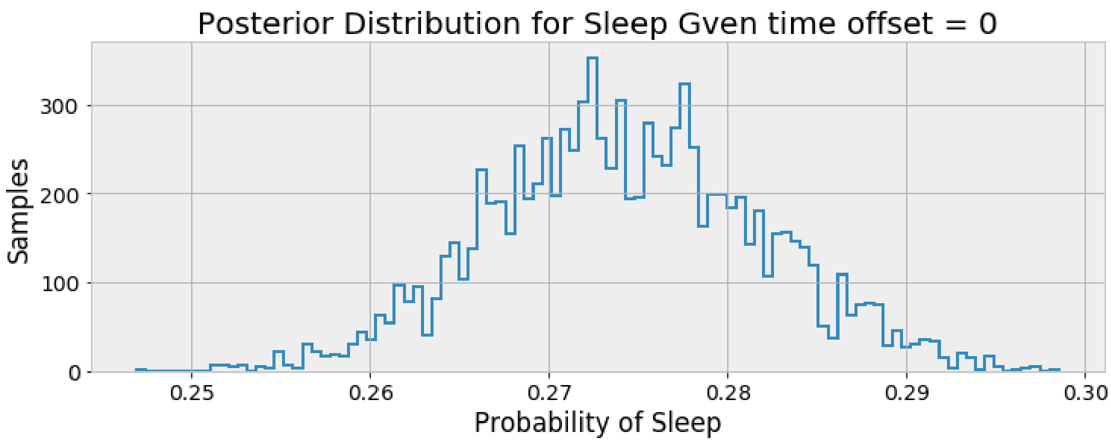
```
plt.plot(time_est, sleep_est, lw=2, ls='--',
         color='black', label="average posterior \nprobability of sleep")
plt.xticks([-60, -30, 0, 30, 60, 90, 120], sleep_labels);
plt.scatter(time, sleep_obs, edgecolor = 'skyblue', s=50, alpha=0.1);
plt.legend(prop={'size':14})
plt.xlabel('PM Time'); plt.ylabel('Probability');
plt.title('Posterior Probabilty with 95% CI');
```



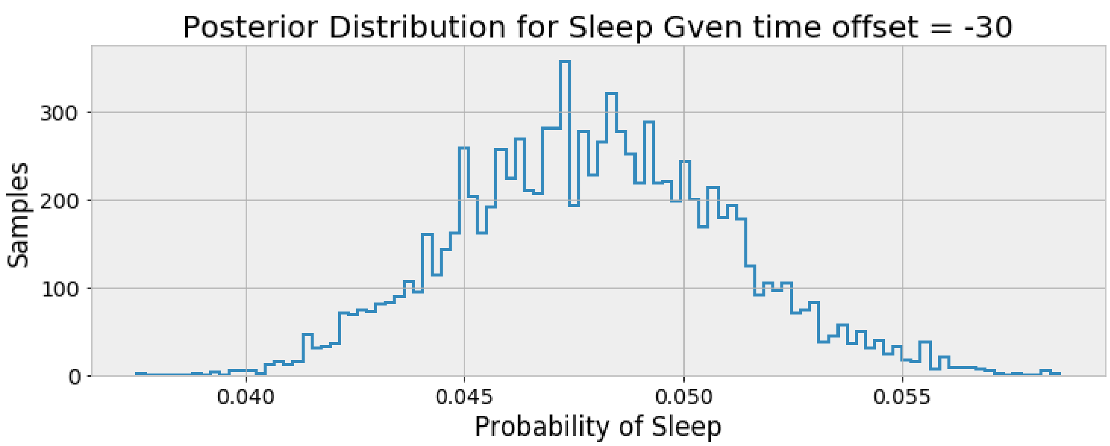
5.3 Posterior Probability for Specific Time

We can plot the posterior distribution for time where each estimate is taken using all the samples from the Markov Chain Monte Carlo. This gives us a look at the uncertainty in the model. Each estimate is made using different values from the sampling.

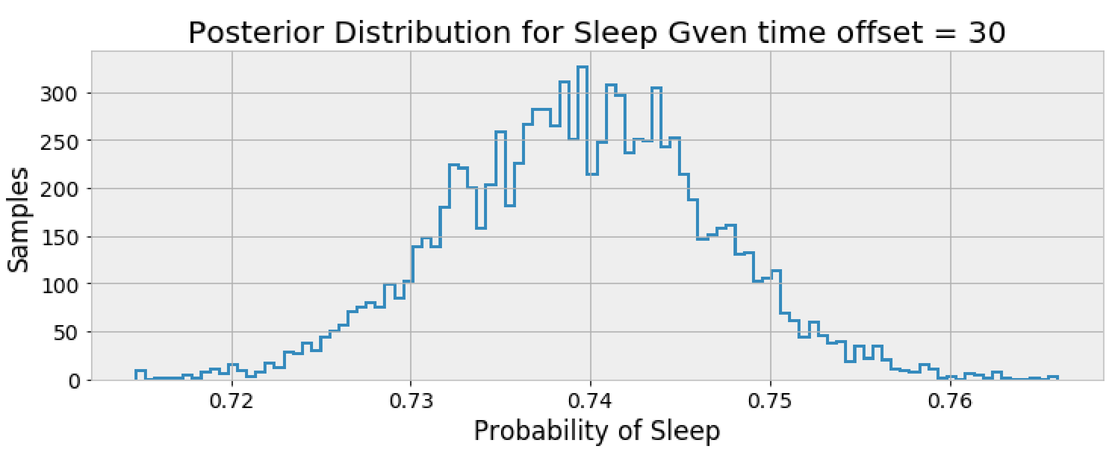
```
In [74]: def sleep_posterior(time_offset):
             figsize(12, 4)
             prob = logistic(time_offset, beta_samples, alpha_samples)
             plt.hist(prob, bins=100, histtype='step', lw=2)
             plt.title('Posterior Distribution for Sleep Gven time offset = %d' %
                       time_offset)
             plt.xlabel('Probability of Sleep'); plt.ylabel('Samples')
             plt.show();

In [75]: sleep_posterior(0)
```



In [76]: sleep_posterior(-30)



In [77]: sleep_posterior(30)



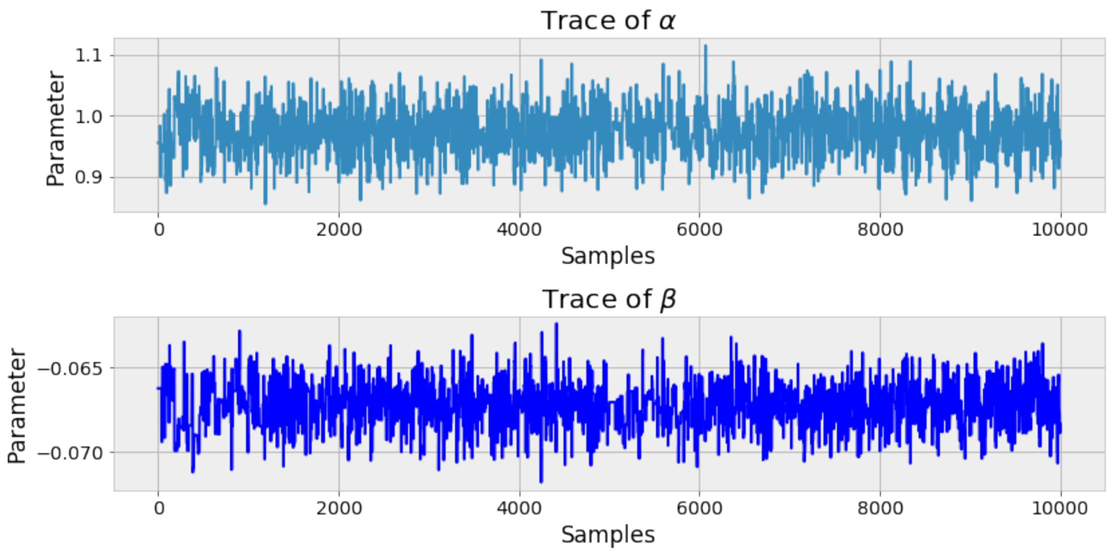
6 Convergence in Markov Chain Monte Carlo

How can we know if the model converged? We can look at the trace, or the path of the values over sampling. Another option is to look at the auto-correlation of the samples. In Markov Chain modeling, the samples are correlated with themselves because the next value depends on the current state (or the current state and past states based on the order). Initially, the algorithm tends to wander about the search space and will have a high auto-correlation. As the algorithm converges, the samples will settle down around a value and one measure of convergence is a low auto-correlation.

6.1 Trace Plots

We can start off making traces of α and β parameters as the algorithm progresses. There is a burn in period, and usually this is removed from the samples because it is not indicative of the final converged values.

```
In [78]: figsize(12, 6)

         # Plot alpha trace
         plt.subplot(211)
         plt.title(r'Trace of $\alpha$')
         plt.plot(alpha_samples)
         plt.xlabel('Samples'); plt.ylabel('Parameter');

         # Plot beta trace
         plt.subplot(212)
         plt.title(r'Trace of $\beta$')
         plt.plot(beta_samples, color='b')
         plt.xlabel('Samples'); plt.ylabel('Parameter');
         plt.tight_layout(h_pad=0.8)
```



7 Incorporating Additional Information

The model we have currently returns the probability I am asleep given only the time. There are additional pieces of information such as if my bedroom light is on, or if my phone is charging, that would improve the model. The way to incorporate additional evidence in Bayesian inference is by using Bayes Rule. We can use the posterior probability for if I am asleep as the prior in the new model. I don't have data on the likelihoods for my light and phone, but I can provide estimates based on habits. Following are the likelihoods for these pieces of evidence:

$$P(L = 1|s) = 0.01$$
$$P(L = 1|\bar{s}) = 0.90$$

and

$$P(C = 1|s) = 0.95$$
$$P(C = 1|\bar{s}) = 0.40$$

Where $L = 1$ indicates my bedroom light is on and $C = 1$ indicates my phone is charging.

Applying Baye's Equation, the probability I am asleep at a given time knowing the condition of my light and phone is:

$$P(s|L, C) = \frac{P(L, C|s)P(s)}{P(L, C)}$$

I will assume that my bedroom light and phone charging are conditionally independent of one another with the knowledge of whether or not I am asleep. This means that the probabilty of my light being on and my phone charging are not dependent if we know I am asleep or not.

$$P(L, C|s) = P(L|s) * P(C|s)$$

Baye's Equation then becomes:

$$P(s|L, C) = \frac{P(L|s)P(C|s) * P(s)}{P(L|s)P(C|s) * P(s) + P(L|\bar{s})P(C|\bar{s}) * P(\bar{s})}$$

Where $P(s)$ is the prior probability of my being asleep from the Markov Chain Monte Carlo Bayesian Inference.

The first step is to write a simple function which will compute the prior of sleep given the parameters from the MCMC.

```
In [79]: alpha_sleep_est = alpha_est
         beta_sleep_est = beta_est
```

```
In [80]: # Returns the prior based on the average parameters from MCMC
         def sleep_prior(time_offset):
             return logistic(time_offset,
                             beta=beta_sleep_est,
                             alpha=alpha_sleep_est)
```

```
In [82]: # Returns the posterior probability of sleep given the time and condition of th
         def sleep_posterior(time_offset, light, charge,
                             light_likelihood_sleep=0.01,
                             light_likelihood_awake=0.90,
                             charge_likelihood_sleep=0.95,
                             charge_likelihood_awake=0.40):
             prior_sleep = sleep_prior(time_offset)
             prior_awake = 1 - prior_sleep

             # Change the priors based on the evidence
             if not light:
                 light_likelihood_sleep = 1 - light_likelihood_sleep
                 light_likelihood_awake = 1 - light_likelihood_awake
             if not charge:
                 charge_likelihood_sleep = 1 - charge_likelihood_sleep
                 charge_likelihood_awake = 1 - charge_likelihood_awake

             # Calculate the numerator and denominator in Baye's Equation
             numerator = light_likelihood_sleep * charge_likelihood_sleep * prior_sleep
             denominator = (numerator +
                            (light_likelihood_awake * charge_likelihood_awake * prior_awa
```

```
        # Return the posterior probability
        posterior_sleep = numerator / denominator
        return posterior_sleep
```

We can now use this model to make predictions with more information.

Let's try a few and compare them to the prior. First up, what is the probability I am asleep at 10:00 pm if my light is off and my phone charging?

```
In [83]: print('The prior for this case is     {:.2f}%.'.
                format(100 * sleep_prior(0)))
         print('The posterior for this case is {:.2f}%.'.
                format(100 * sleep_posterior(0, 0, 1)))
```

```
The prior for this case is     27.42%.
The posterior for this case is 89.88%.
```

How about if it is 10:30 pm, my light is on, and my phone is charging?

```
In [84]: print('The prior for this case is     {:.2f}%.'.
                format(100 * sleep_prior(30)))
         print('The posterior for this case is {:.2f}%.'.
                format(100 * sleep_posterior(30, 1, 1)))
```

```
The prior for this case is     73.95%.
The posterior for this case is 6.97%.
```

One more situation. It is 9:45 pm, my light is off, and my phone is not charging.

```
In [85]: print('The prior for this case is     {:.2f}%.'.
                format(100 * sleep_prior(-15)))
         print('The posterior for this case is {:.2f}%.'.
                format(100 * sleep_posterior(-15, 0, 0)))
```

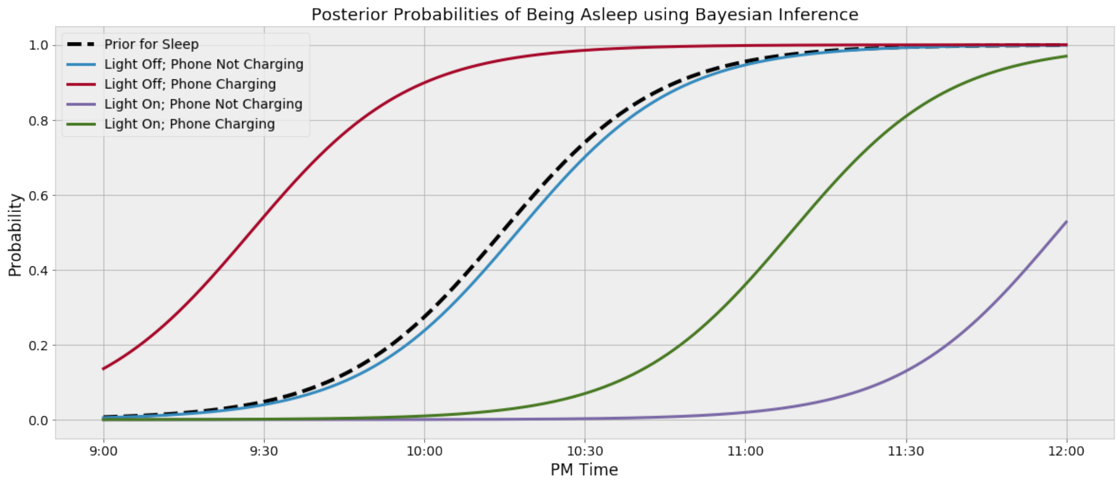
```
The prior for this case is     12.11%.
The posterior for this case is 10.21%.
```

We can see how incorporating more information changes the estimate. With more quality data, the model would be more accurate. This is a great example of Bayesian Inference, where we update our beliefs to incorporate more information. Bayesian Inference means becoming less wrong about the world!

7.1 Visualize Shifts

To show how this information changes, we can make the same plots as before but with the additional information. There are four additional situations that shift the curve.

```
In [86]: figsize(20, 8)

         x = np.linspace(-60, 120, 1e3)
         priors = logistic(x, beta=beta_sleep_est, alpha=alpha_sleep_est )

         evidence = [[0, 0], [0, 1], [1, 0], [1, 1]]
         labels = ['Light Off; Phone Not Charging', 'Light Off; Phone Charging',
                   'Light On; Phone Not Charging', 'Light On; Phone Charging']

         plt.plot(x, priors, label='Prior for Sleep', ls = '--', color = 'k', lw = 4)

         for i, obs in enumerate(evidence):
             posteriors = []
             for time_offset in x:
                 posterior = sleep_posterior(time_offset, obs[0], obs[1])
                 posteriors.append(posterior)

             plt.plot(x, posteriors, label=labels[i], lw = 3)


         plt.legend(prop={'size':14}); plt.xticks([-60, -30, 0, 30, 60, 90, 120], sleep_
         plt.xlabel('PM Time'); plt.ylabel('Probability')
         plt.title("Posterior Probabilities of Being Asleep using Bayesian Inference",
                   size = 18);
```



The graph shows the impact of incorporating more knowledge. If we know the light is on, the probabilty I am asleep at a given time decreases, and if the light is off and the phone charging, the probability I am sleeping at a given time increases! Bayesian Inference is very neat!

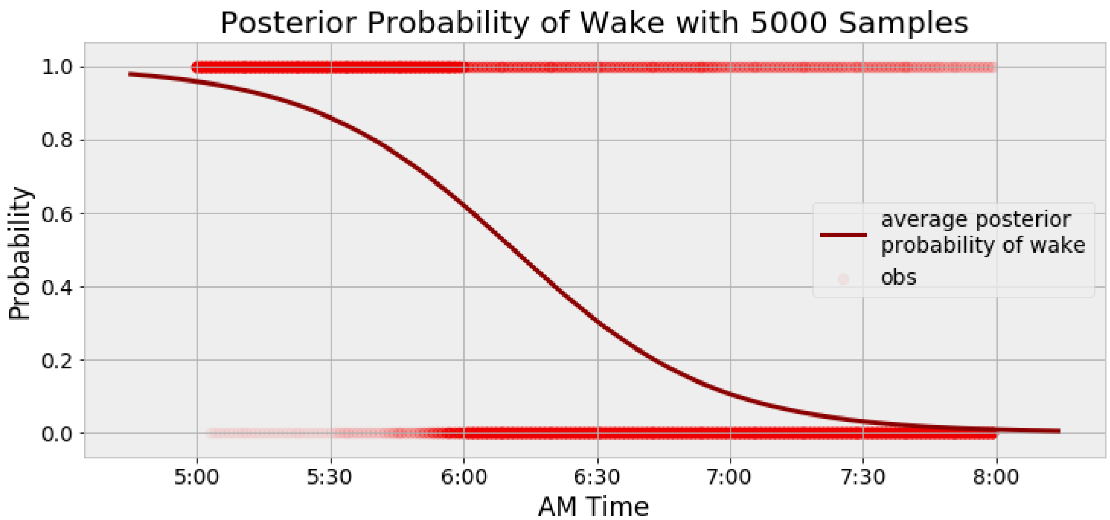
8 Wake Model

We can repeat the same procedure with the wake data. The process is exactly the same we are just using different data.

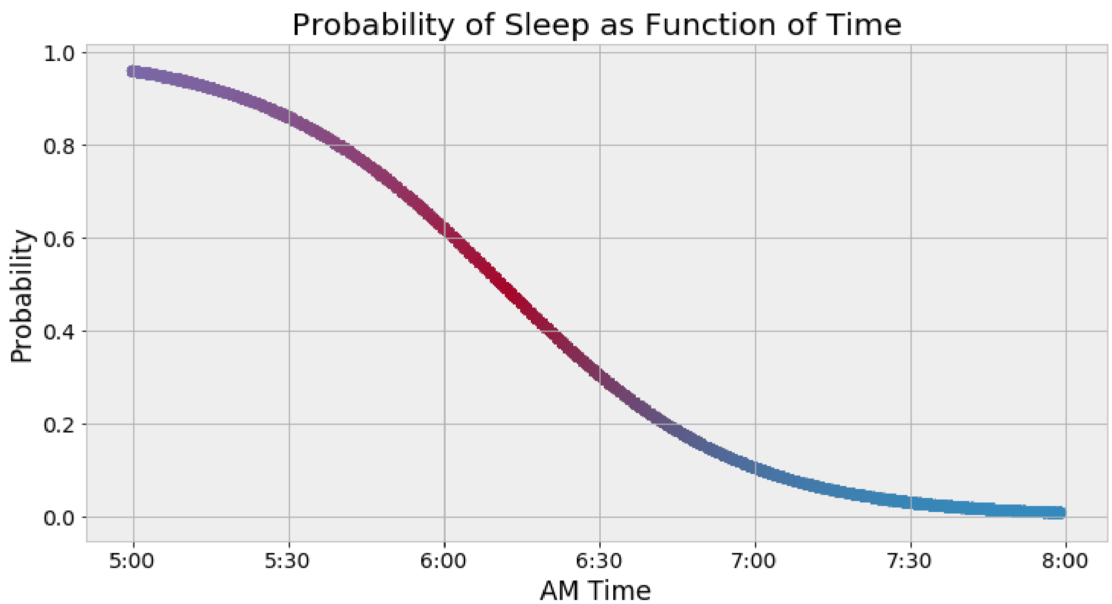
```
In [87]: # Sort the values by time offset
         wake_data.sort_values('time_offset', inplace=True)

         # Time is the time offset
         time = np.array(wake_data.loc[:, 'time_offset'])

         # Observations are the indicator
         wake_obs = np.array(wake_data.loc[:, 'indicator'])

         with pm.Model() as wake_model:
             # Create the alpha and beta parameters
             alpha = pm.Normal('alpha', mu=0.0, tau=0.05, testval=0.0)
             beta = pm.Normal('beta', mu=0.0, tau=0.05, testval=0.0)

             # Create the probability from the logistic function
             p = pm.Deterministic('p', 1. / (1. + tt.exp(beta * time + alpha)))

             # Create the bernoulli parameter which uses the observed data
             observed = pm.Bernoulli('obs', p, observed=wake_obs)

             # Starting values are found through Maximum A Posterior estimation
             # start = pm.find_MAP()

             # Using Metropolis Hastings Sampling
             step = pm.Metropolis()

             # Sample from the posterior using the sampling method
             wake_trace = pm.sample(N_SAMPLES, step=step);
```

```
Multiprocess sampling (2 chains in 2 jobs)
CompoundStep
>Metropolis: [beta]
>Metropolis: [alpha]
The number of effective samples is smaller than 10% for some parameters.
```

```
In [88]: # Extract the alpha and beta samples
         # Currently using all, including the burn-in period
         alpha_samples = wake_trace["alpha"][:, None]
         beta_samples = wake_trace["beta"][:, None]

         # Time values for probability prediction
         time_est = np.linspace(time.min()- 15, time.max() + 15, 1e3)[:, None]

         # Take most likely parameters to be mean values
         alpha_est = alpha_samples.mean()
         beta_est = beta_samples.mean()

         # Probability at each time using mean values of alpha and beta
         wake_est = logistic(time_est, beta=beta_est, alpha=alpha_est)

         figsize(12, 5)

         plt.plot(time_est, wake_est, color = 'darkred',
                  lw=3, label="average posterior \nprobability of wake")
         plt.scatter(time, wake_obs, edgecolor = 'r', facecolor = 'r',
                     s=50, alpha=0.05, label='obs')
         plt.title('Posterior Probability of Wake with %d Samples' % N_SAMPLES);
         plt.legend(prop={'size':14})
         plt.ylabel('Probability')
         plt.xlabel('AM Time');
         plt.xticks([-60, -30, 0, 30, 60, 90, 120], wake_labels);
```



```
In [89]: colors = ["#348ABD", "#A60628", "#7A68A6"]
         cmap = mpl.colors.LinearSegmentedColormap.from_list("BMH", colors)
         figsize(12, 6)
         probs = wake_trace['p']

         plt.scatter(time, probs.mean(axis=0), cmap = cmap,
                     c = probs.mean(axis=0), s = 50);
         plt.title('Probability of Sleep as Function of Time')
         plt.xlabel('AM Time');
         plt.ylabel('Probability');
         plt.xticks([-60, -30, 0, 30, 60, 90, 120], wake_labels);
```



8.1 Investigate the Model

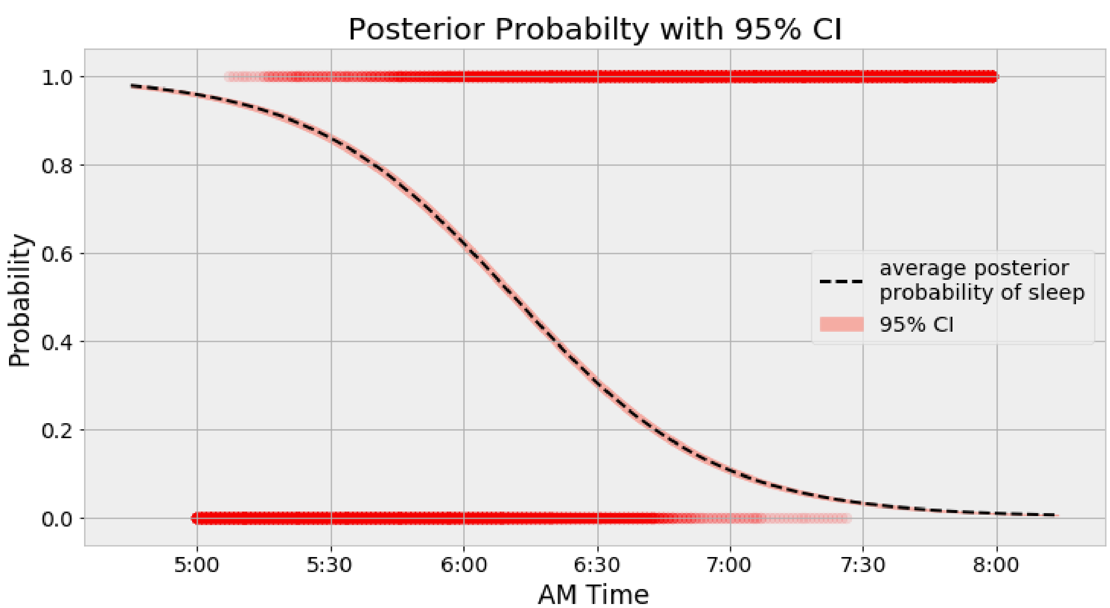
```
In [90]: print('Probability of being awake at 5:30 AM: {:.2f}%.'.
               format(100 - (100 * logistic(-30, beta=beta_est, alpha=alpha_est))))
         print('Probability of being awake at 6:00 AM: {:.2f}%.'.
               format(100 - (100 * logistic(0, beta=beta_est, alpha=alpha_est))))
         print('Probability of being awake at 6:30 AM: {:.2f}%.'.
               format(100 - (100 * logistic(30, beta=beta_est, alpha=alpha_est))))
```

```
Probability of being awake at 5:30 AM: 14.10%.
Probability of being awake at 6:00 AM: 37.94%.
Probability of being awake at 6:30 AM: 69.50%.
```

8.2 Confidence Intervals (95%)

```
In [91]: wake_all_est = logistic(time_est.T, beta_samples, alpha_samples)
         quantiles = mquantiles(wake_all_est, [0.025, 0.975], axis=0)
```

```
In [92]: plt.fill_between(time_est[:, 0], *quantiles,
                          alpha=0.6, color='salmon', label = '95% CI')
         plt.plot(time_est, wake_est, lw=2, ls='--',
                  color='black', label="average posterior \nprobability of sleep")
         plt.xticks([-60, -30, 0, 30, 60, 90, 120], wake_labels);
         plt.scatter(time, sleep_obs, edgecolor = 'red',
                     facecolor = 'red', s=50, alpha=0.1);
         plt.legend(prop={'size':14})
         plt.xlabel('AM Time'); plt.ylabel('Probability');
         plt.title('Posterior Probabilty with 95% CI');
```

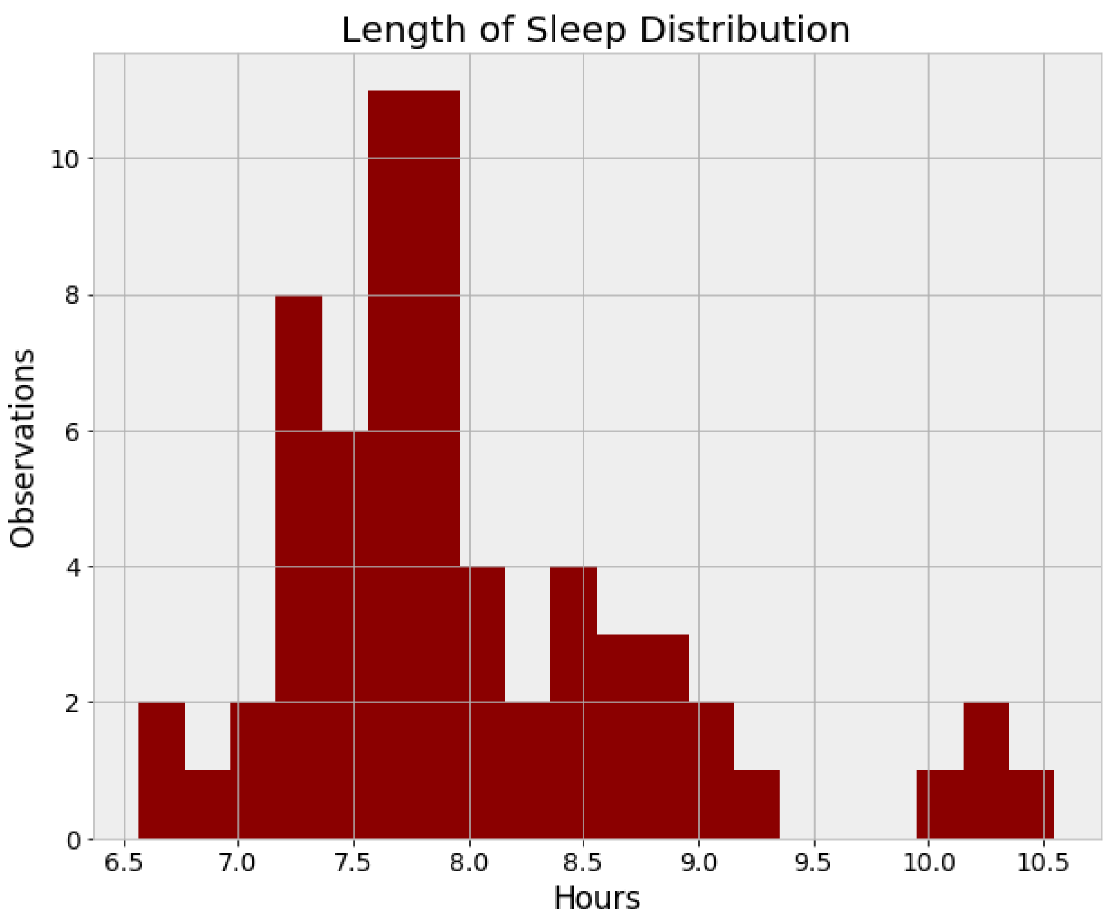


9 Length of Sleep

We can also form a model to estimate the most likely length of time I am asleep. We can first look at the data and then determine which distribution fits best.

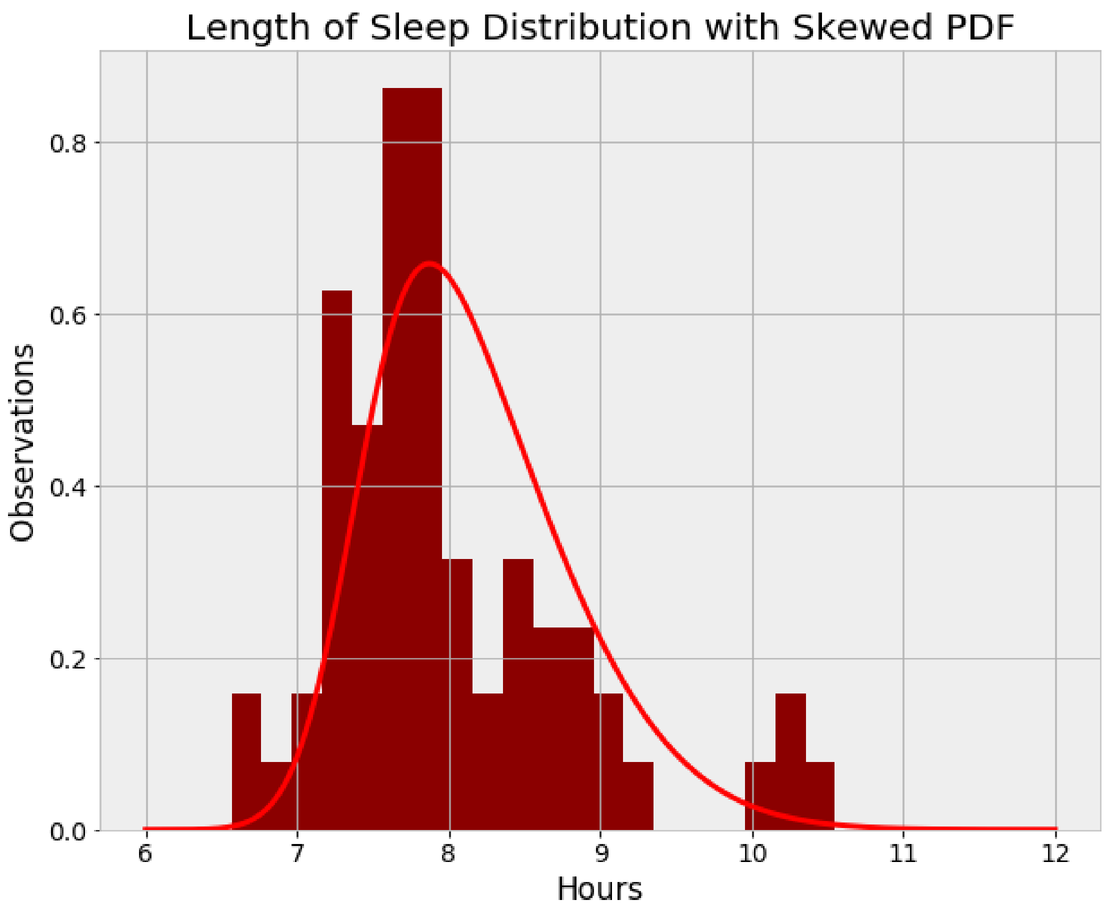
```
In [93]: raw_data = pd.read_csv('data/sleep_wake.csv')
         raw_data['length'] = 8 - (raw_data['Sleep'] / 60) + (raw_data['Wake'] / 60)
         duration = raw_data['length']
```

```
In [94]: figsize(10, 8)
         plt.hist(duration, bins = 20, color = 'darkred')
         plt.xlabel('Hours'); plt.title('Length of Sleep Distribution');
         plt.ylabel('Observations');
```



The distribution is skewed to the right. Therefore, we can used a skewed distribution to model the length of sleep.

```
In [95]: a = 3
         fig, ax = plt.subplots(1, 1)
         x = np.linspace(6, 12, 1e3)


         figsize(10, 8)
         plt.hist(duration, bins = 20, color = 'darkred', normed=True)
         plt.xlabel('Hours'); plt.title('Length of Sleep Distribution with Skewed PDF');
         plt.ylabel('Observations');
         plt.plot(x, stats.skewnorm.pdf(x, a, loc = 7.4, scale=1), 'r-',
                  lw=3, label='skewnorm pdf');
```




```
In [96]: with pm.Model() as duration_model:
             # Three parameters to sample
             alpha_skew = pm.Normal('alpha_skew', mu=0, tau=0.5, testval=3.0)
             mu_ = pm.Normal('mu', mu=0, tau=0.5, testval=7.4)
             tau_ = pm.Normal('tau', mu=0, tau=0.5, testval=1.0)

             # Duration is a deterministic variable
             duration_ = pm.SkewNormal('duration', alpha = alpha_skew, mu = mu_,
                                       sd = 1/tau_, observed = duration)

             # Metropolis Hastings for sampling
             step = pm.Metropolis()
             duration_trace = pm.sample(N_SAMPLES, step=step)
```

```
Multiprocess sampling (2 chains in 2 jobs)
CompoundStep
```

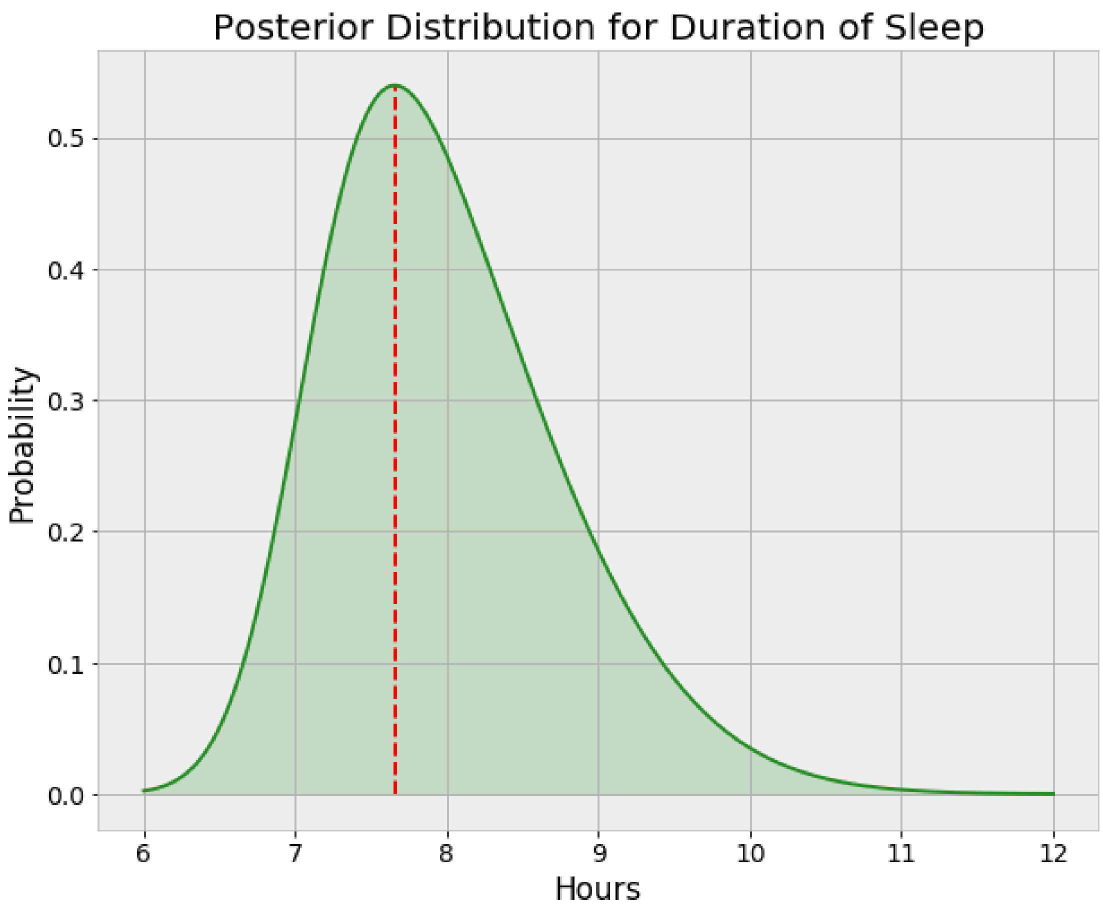
```
>Metropolis: [tau]
>Metropolis: [mu]
>Metropolis: [alpha_skew]
The number of effective samples is smaller than 10% for some parameters.
```

```
In [97]: # Extract the most likely estimates from the sampling
         alpha_skew_samples = duration_trace['alpha_skew']
         mu_samples = duration_trace['mu']
         tau_samples = duration_trace['tau']

         alpha_skew_est = alpha_skew_samples.mean()
         mu_est = mu_samples.mean()
         tau_est = tau_samples.mean()
```

9.1 Visualize Posterior Distribution

```
In [98]: x = np.linspace(6, 12, 1000)
         y = stats.skewnorm.pdf(x, a = alpha_skew_est, loc=mu_est, scale=1/tau_est)
         plt.plot(x, y, color = 'forestgreen')
         plt.fill_between(x, y, color = 'forestgreen', alpha = 0.2);
         plt.xlabel('Hours'); plt.ylabel('Probability');
         plt.title('Posterior Distribution for Duration of Sleep');
         plt.vlines(x = x[np.argmax(y)], ymin=0, ymax=y.max(),
                    linestyles='--', linewidth=2, color='red',
                    label = 'Most Likely Duration');

         print('The most likely duration of sleep is {:.2f} hours.'.format(x[np.argmax(y
```

```
The most likely duration of sleep is 7.66 hours.
```



9.1.1 Query the Posterior Model

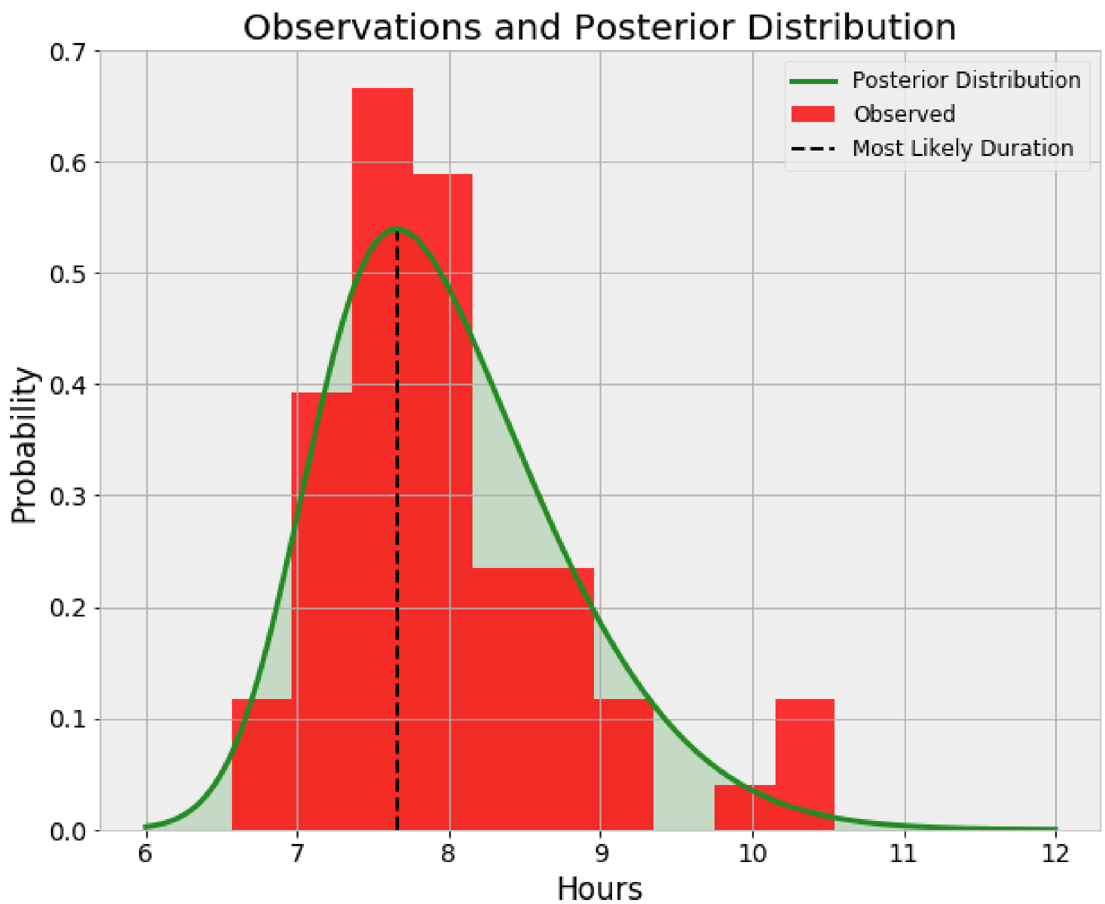
```
In [99]: print('Probability of at least 6.5 hours of sleep = {:.2f}%.'.
               format(100 * (1 - stats.skewnorm.cdf(6.5, a = alpha_skew_est, loc = mu_es
         print('Probability of at least 8.0 hours of sleep = {:.2f}%.'.
               format(100 * (1 - stats.skewnorm.cdf(8.0, a = alpha_skew_est, loc = mu_es
         print('Probability of at least 9.0 hours of sleep = {:.2f}%.'.
               format(100 * (1 - stats.skewnorm.cdf(9.0, a = alpha_skew_est, loc = mu_es
```

```
Probability of at least 6.5 hours of sleep = 99.10%.
Probability of at least 8.0 hours of sleep = 44.31%.
Probability of at least 9.0 hours of sleep = 11.10%.
```

9.2 Visualize the Posterior and the Data

```
In [100]: x = np.linspace(6, 12, 1000)
          y = stats.skewnorm.pdf(x, a = alpha_skew_est, loc=mu_est, scale=1/tau_est)
```

```
# Plot the posterior distribution
plt.plot(x, y, color = 'forestgreen',
         label = 'Posterior Distribution', lw = 3)
plt.fill_between(x, y, color = 'forestgreen', alpha = 0.2);

# Plot the observed values
plt.hist(duration, bins=10, color = 'red', alpha=0.8,
         label='Observed', normed=True)
plt.xlabel('Hours'); plt.ylabel('Probability');
plt.title('Observations and Posterior Distribution');
plt.vlines(x = x[np.argmax(y)], ymin=0, ymax=y.max(),
           linestyles='--', linewidth=2, color='k',
           label = 'Most Likely Duration');
plt.legend(prop={'size':12});
```



The posterior skewed normal distribution looks to fit the data well. However, the data actually may be better modeled as two separate distributions given the second mode to the right. The second mode is not captured in a single skewed normal distribution.

10 Conclusions

This report looked at a number of techniques for implementing Bayesian Inference on real data. A number of models were developed:

```
* Posterior probability I am asleep in the evening at a given time
* Posterior probability I am asleep at a given time with more info using Bayes Rule
* Posterior probability I am asleep in the morning at a given time
* Posterior distribution of the duration of my sleep
```

The results may not be entirely useful, but they could be combined with additional factors such as the day of the week or the daily activities and caffience intake. Nonetheless, this was a great start towards analysis real-world data from a Bayesian viewpoint. I had never used Markov Chain Monte Carlo before this project, and having applied it, I now feel confident extending this to other domains. There were many topics I learned from this project, and I am appreciate the individuals who contributed to my thought process including Robin Cole and the author of *Probabilistic Programming and Bayesian Methods for Hackers*. Learning in the technical age is changing, and this project is a great example because I got the idea from a friend working on an open-source project, and learned how to use all the methods from a free book. The project itself may not be that significant, but what I have learned will be of great use and I look forward to using more Bayesian inference methods.