```
import numpy as np
from math import log, sqrt
import scipy.stats
import pandas as pd
import random
import matplotlib.pyplot as plt

def log_likelihood(n1, n2, a, W): #P(DIW)
    # this function takes a numpy array for n1, n2, and the accuracy (0/1), whether they
answered correctly
    # as well as W (the hypothesis)
    # and returns the *log* likelihood of the responses, log P(accuracy I n1, n2, W)

    assert(len(n1) == len(n2) == len(a))

    p = 1.0-scipy.stats.norm.cdf(0, loc=np.abs(n1-n2), scale=W*np.sqrt(n1**2 + n2**2)) #
the probability of answering correctly
    return np.sum(np.where(a, np.log(p), np.log(1.0-p)))

df = pd.read_csv('Assignment9-data.csv')
n1 = df['n1'].values #behavioral stimuli
n2 = df['n2'].values #behavioral stimuli
a = df['correct'].values

"""Problem 1:
    We will implement functions that compute the logarithm of the prior and the logarithm
    posterior, as opposed to the unlogged versions. Explain why in 1-2 sentences.

    ANSWER:
        The unlogged versions of the prior and posterior may be very small (close to 0).
        Python will round these small values down to zero making them useless for
        our calculations. Using the log function will make the values large negatives
        allowing for computations to be made and then converting the numbers back to
        a linear scale. """


"""Problem 2:
    Explain in a 1-2 sentences why the form of the acceptance ratio P(W'ID)/P(WID) is
    convenient for us – that is, why it saves us from doing an integral.

    ANSWER:
        The number of samples (as a time series) we take on H is proportional to P(WI
D),
        that is a discrete series rather than a continuous one that would require an
        integral. We can just sum over the samples, cancelling out the intregral in
        the denominator of the original sampling algorighm.
        """
```

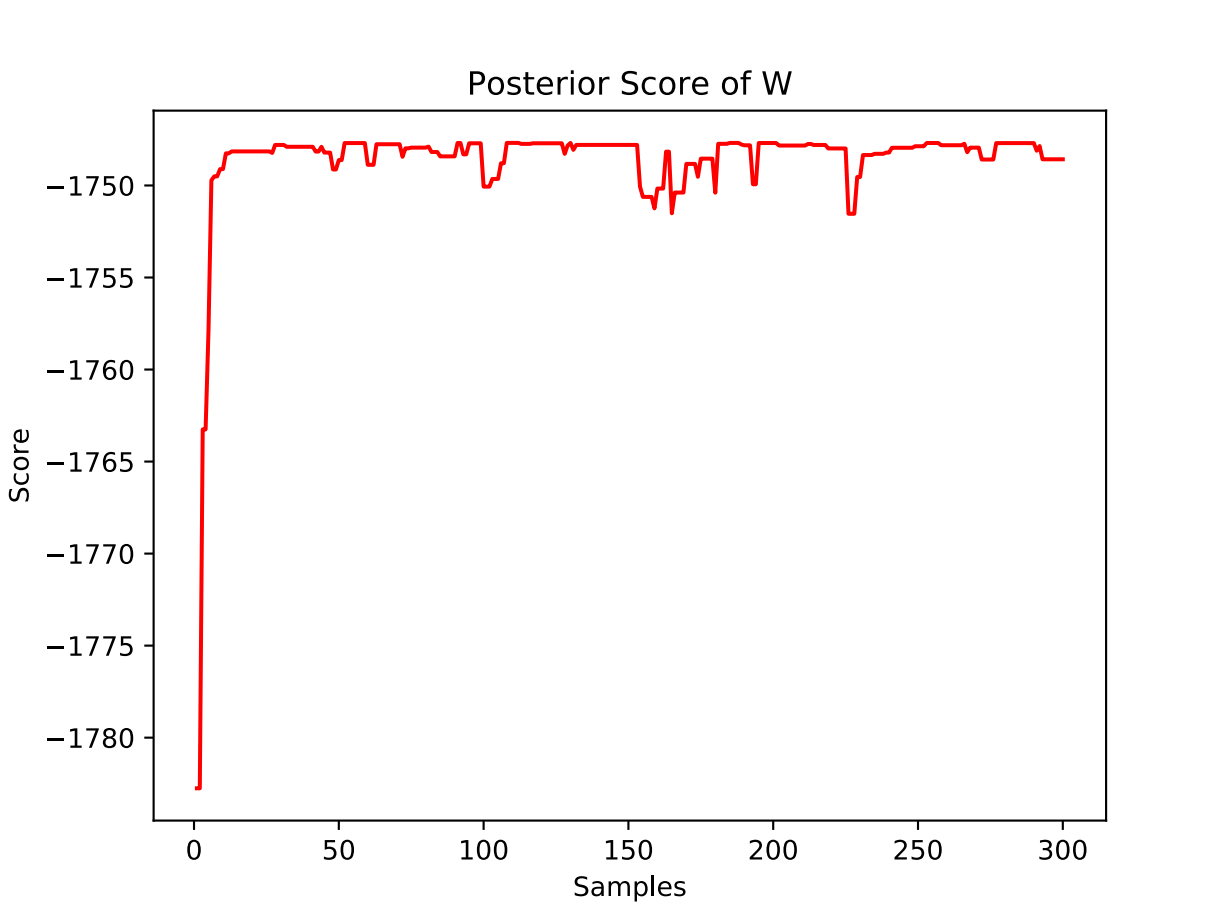
```
"""Problem 3:
    Write functions to compute something proportional to the log prior and log posterior in
    this model. You will prevent later frustration by ensuring that your prior correctly
handles
    cases when W<0 (what should it return?).

    ANSWER:
        I'm not entirely sure what the thought process is behind what should be returned
        when W < 0. I used 1 in order to make the prior have no effect on the Posterior
        This doesn't seem to be an issue though as np.exp(-W) will ever be negative
causing
        an error in np.log.
    """

def logPrior(W):
    if W < 0:
        return 1

    return np.log(np.exp(-W))

def logPosterior(n1, n2, a, W):
    """Returns P(W|D)"""

    return logPrior(W) + log_likelihood(n1, n2, a, W)

"""Problem 4:
    Implement the Metropolis algorithm, starting from a random W, and plot:

    (a) the posterior score of W over the first 300 samples;
    (b) the value of W over the first 300 samples, and
    (c) a histogram of the samples of W over the first 10,000 samples after 1000 samples
of “burn in.”"""

def metropolis(n, probFunc, burnIn=False): #needs testing

    W = random.uniform(0, 1)
    primeW, next, ratio = 0, 0, 0
    abAxis = list(range(1, n + 1))
    wVals = [] #start with W?
    posVals = []
    priorVals = []
    intervalProb = 0
    modifier = 0

    if burnIn:
        abAxis = list(range(1, n - 1000 + 1))
```

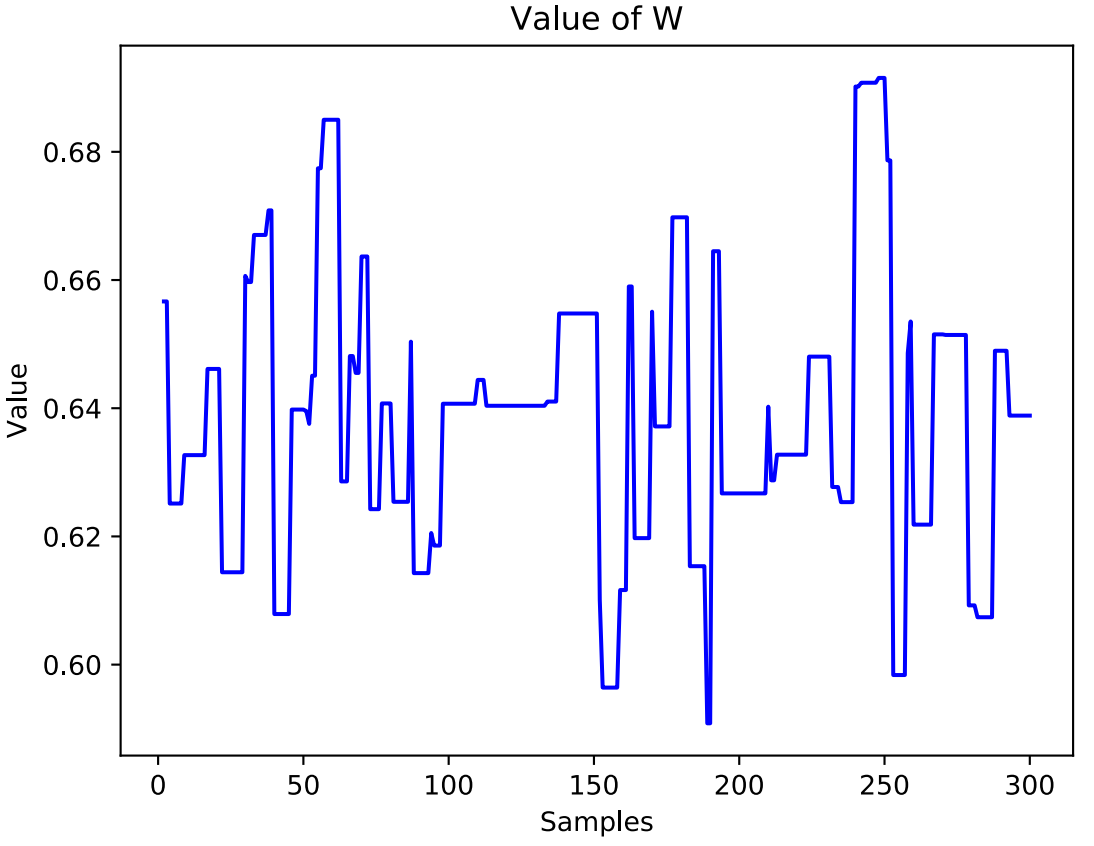
```
    modifier = 1000

    for i in range(1000):
        primeW = W + random.gauss(0, 0.1)
        pPrime = 0

        if probFunc == logPosterior:
            pW = probFunc(n1, n2, a, W)
            pPrime = probFunc(n1, n2, a, primeW)
        if probFunc == logPrior:
            pW = probFunc(W)
            pPrime = probFunc(pPrime)

        ratio = np.exp(pPrime - pW)

        if pW < pPrime:
            W = primeW
        else:
            if random.random() <= ratio:
                W = primeW

for i in range(n - modifier):
    primeW = W + random.gauss(0, 0.1)
    if probFunc == logPosterior:
        pW = probFunc(n1, n2, a, W)
        pPrime = probFunc(n1, n2, a, primeW)

    if probFunc == logPrior:
        pW = probFunc(W)
        pPrime = probFunc(pPrime)
    ratio = np.exp(pPrime - pW)
    priorVals += [logPrior(W)]

    if pW < pPrime:
        W = primeW
        posVals += [pPrime]
    else:
        if random.random() <= ratio:
            W = primeW
            posVals += [pPrime]
        else:
            posVals += [pW]

    wVals += [W]

    if W >= .60 and W <= .65:
        intervalProb += 1
```

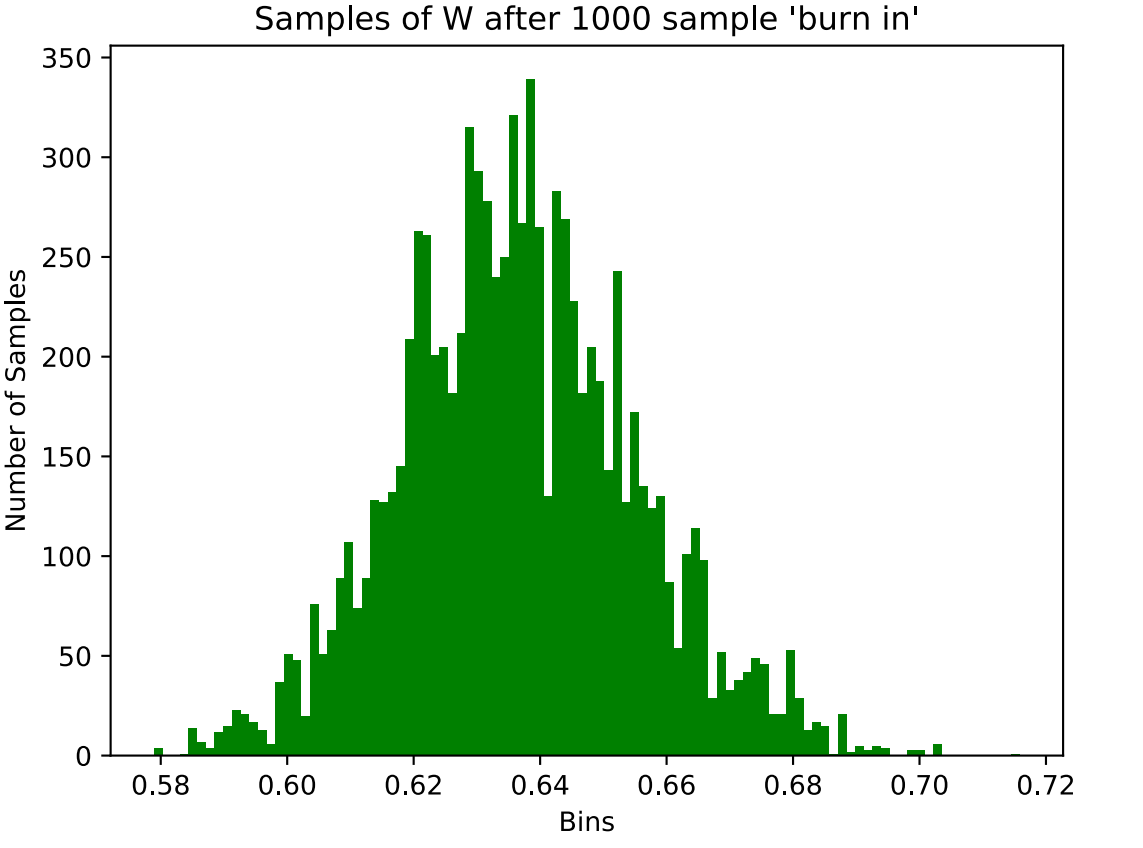
```
    return abAxis, wVals, posVals, priorVals

# xAxis, wData, posteriorData, priorData = metropolis(10000, logPosterior, burnIn
=True)
# xAxis, wData2, posteriorData2, priorData2 = metropolis(10000, logPrior, burnIn =True)
#for Q6
# # data = [wData, wData]
# # bins = [-.70, -.60, .60, .70]
# plt.hist(wData2, 100, color='blue', label='P(W)')
# # xAxis, wData, posteriorData = metropolis(300)
# # colors = ['blue', 'green']
# names = ['P(WID)', 'P(W)']
#
# plt.hist(wData, 100, color='green', label='P(WID)')
# plt.legend()
# plt.title('Samples of P(WID) = P(W)*P(DIW) and P(W) after 1000 sample burn in')
# # plt.title(" Samples of W after 1000 sample 'burn in' ")
# plt.xlabel('Bins')
# plt.ylabel('Number of Samples')
# plt.savefig('a9_p6.pdf')
#
# # plt.plot(xAxis, wData)
# # plt.plot(xAxis[1:], wData[1:], color='blue')
# # plt.title('Value of W')
# # plt.ylabel('Value')
# # plt.xlabel('Samples')
# # plt.savefig('a9_p4b.pdf')
# # plt.show()
# #for histogram
# # alldata = [priordata, posteriordata]
# # plt.hist(alldata, number of bins)
# plt.show()


"""Problem 5:
    Use your sampler to determine the probability that W is in the interval [0.60, 0.65].
    What is that probability?

    ANSWER:
        P(W in [0.60, 0.65]) = ~0.66 """


"""Problem 6:
    Run your sampler on the prior (assuming no data) and plot histograms of the prior
samples
    and the posterior samples (from Q4c) in the same graph. What relationship between
```

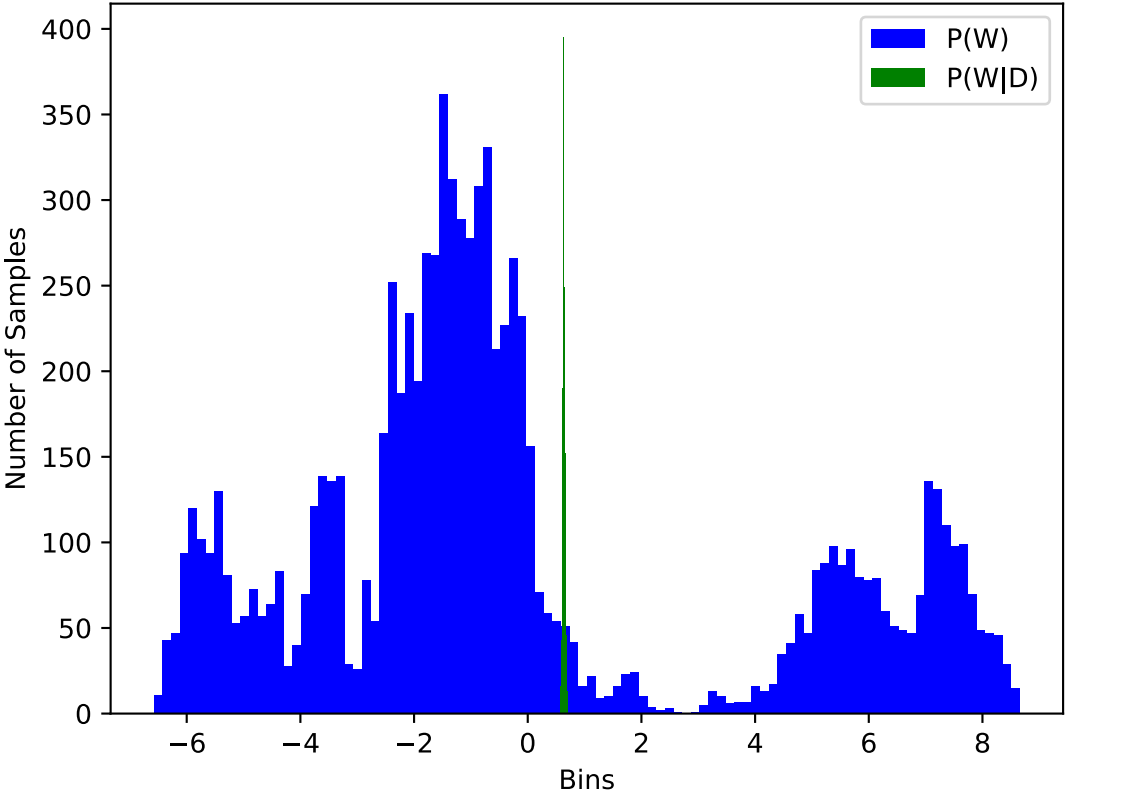
these
distributions does your plot show, and what does it mean?
ANSWER:
Using just P(W) for P(WID) with no data results in a random W.
Using data, W is bound between ~[.60, .70] showing there is a pattern
in the data and that the Metropolis algorigthm finds this pattern. """

"""Problem 7 (Extra Credit):
Do you get “better” estimates with this algorithm if you run one chain for 1000 steps or
100 chains for 10 steps each and just concatenate their samples? To answer this, you should
come up with both:
(a) a simulation that tries both of these options (you'll want to run more than one time!)
and
(b) a measure of how “good” a run was (i.e. how accurate its samples were).
Write a sentence or two justifying your measure of goodness, create a visualization of the
two options, and explain in a sentence how the visualization answers the question."""
#end


Posterior Score of W
−1750
−1755
−1760
−1765
−1770
−1775
−1780
Score
0
50
100
150
200
250
300
Samples


Value of W
Value
Samples
0.68
0.66
0.64
0.62
0.60
0
50
100
150
200
250
300


Samples of W after 1000 sample 'burn in'
Number of Samples
Bins
0
50
100
150
200
250
300
350
0.58
0.60
0.62
0.64
0.66
0.68
0.70
0.72


Samples of P(W|D) = P(W)*P(D|W) and P(W) after 1000 sample burn in
P(W)
P(W|D)
Number of Samples
Bins
0
50
100
150
200
250
300
350
400
−6
−4
−2
0
2
4
6
8