Ph 21 Homework 4

Emily Springer

February 22, 2020

1 Question 1.

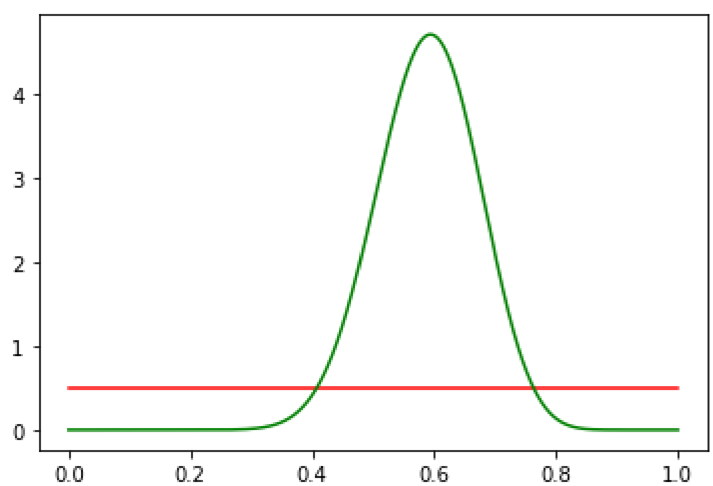
To simulate tossing the coin, I wrote the code

```
nHeads = 0;
for i in range(n):
    rand = random.randint(0, 1)
    if (rand == 1):
        nHeads +=
probHeads = nHeads / n;
```

probHeads represents the probability of getting heads in this simulation. To calculate the posterior, I started with a flat prior of 0.5. I calculated the posterior with the following code.

```
likelihood = math.factorial(n) / (math.factorial(nHeads) * \
        math.factorial(n - nHeads)) * H ** nHeads * (1 - H) ** (n - nHeads)
plt.plot(H, prior, 'r')
for j in range (nPoints):
    normalization += likelihood[j] * prior[j] * 0.001
prob = likelihood * prior / normalization
plt.plot(H, prob, 'g')
plt.show()
```

With n = 32, I got the following graph



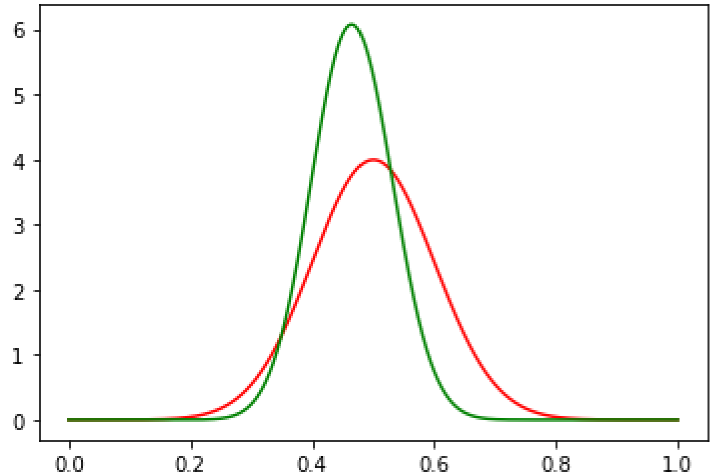
where red is the prior and green is the posterior.

For the Gaussian prior, I used

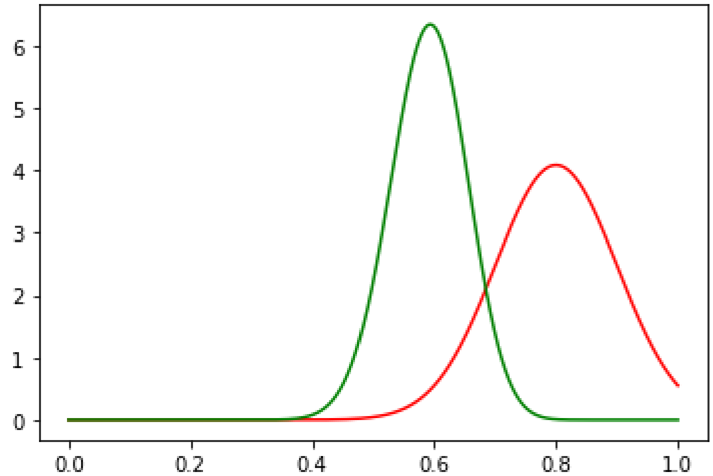
```
sigma = 0.1
```

```
mu = 0.5
norm = 0
prior = 1/(sigma * np.sqrt(2 * np.pi)) * np.exp( - (H - mu)**2 / (2 * sigma**2))
for j in range (nPoints):
        norm += prior[j] * 0.001
prior /= norm
```

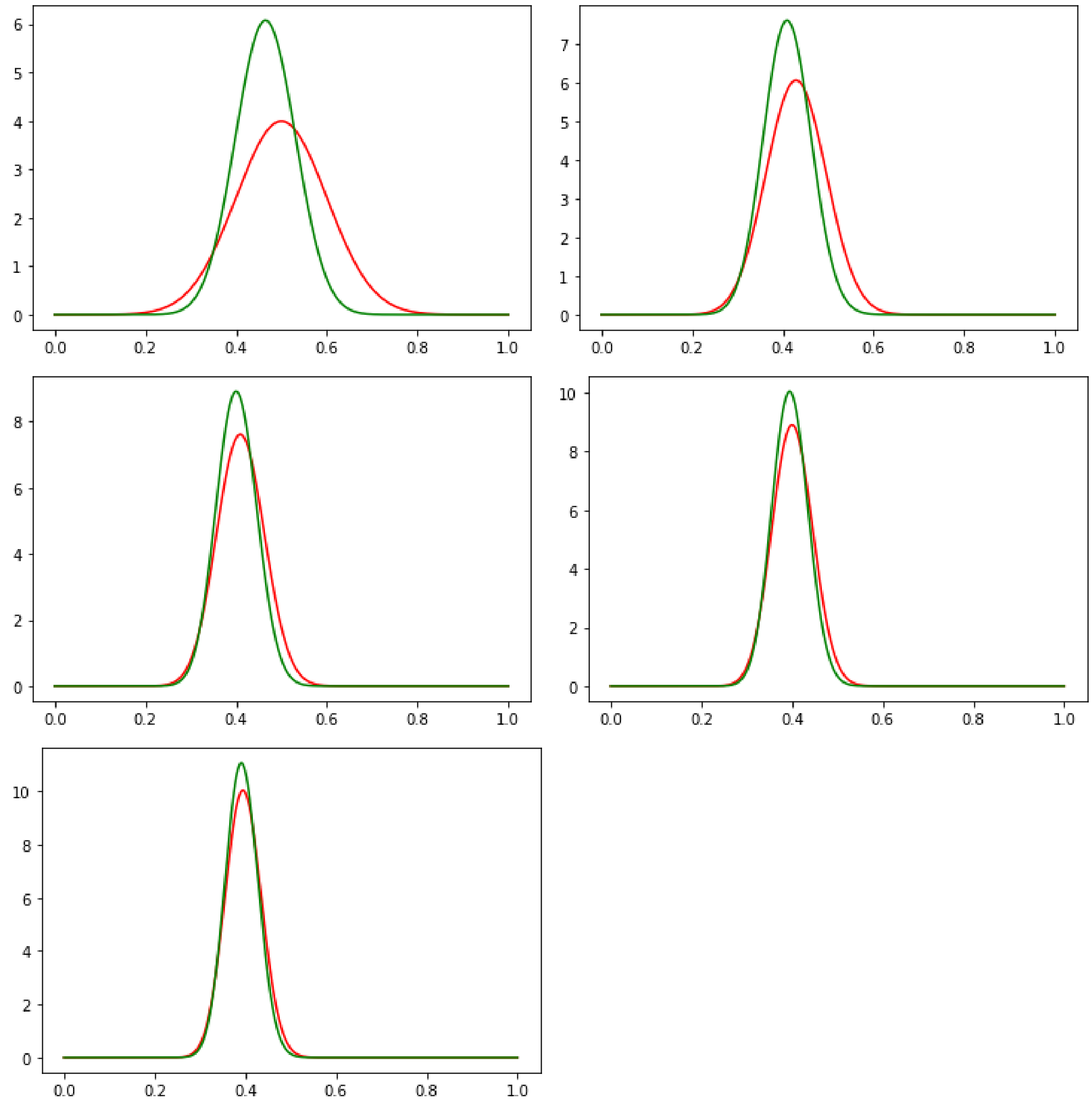
With n = 32, I got the following graph



where red is the prior and green is the posterior. This Gaussian prior was right at the "true" value for H. When the mean is moved to 3 sigma away for the Gaussian prior, I get the following graph.



TO evolve with time, I looped my posterior calculation, resetting the prior to the previous posterior each time. For n = 32 with a Gaussian prior centered at 0.5, I get the following progression.



The posterior converges to the prior.

2 Question 2.

Assuming B's and A's true values to be 1km, I wrote a program to simulate the probability the probability of getting various x_k from experiment. The code was

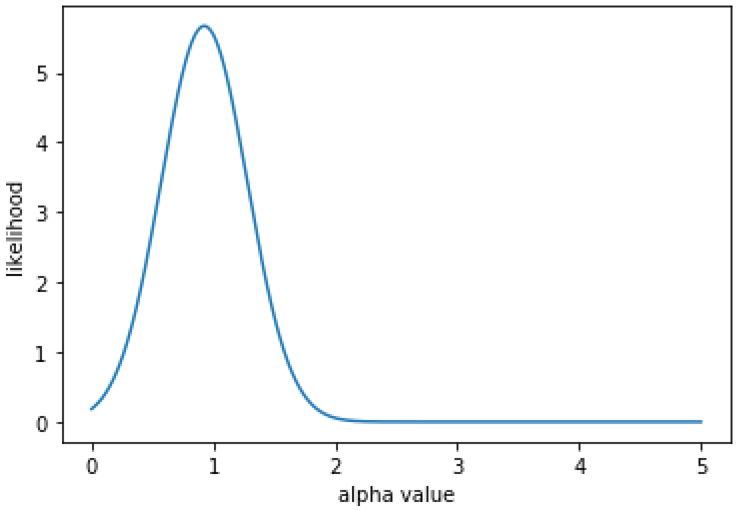
```
def getData(nPoints):
    allPoints = []
    angles = np.linspace(0, math.pi, 10000)
    B = 1
    A = 1
    for i in range(nPoints):
        randIdx = random.randint(0, nPoints - 1)
        x = math.tan(angles[randIdx]) * B + A
```

```
        allPoints.append(x)
    return allPoints
```

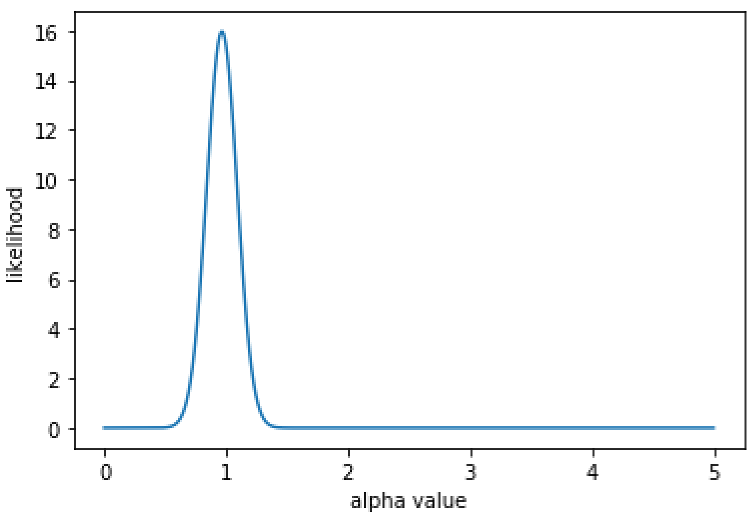
I used this to randomly sample from it to simulate taking real data. To calculate the likelihood, I used the property of

$$\ln(likelihood) = \ln(\beta) - \ln(\pi) - \sum_{k=1}^{N}(\beta^2 + (x_k - \alpha)^2)$$

Before taking the exponential of the likelihood, I needed to normalize it otherwise python would just zero everything because the values are too small. I did this by calculating the maximum exponent and adding it to everything, which is the equivalent of multiplying the function by e^{max}. This makes sure no values are too low. Then, I could take the exponential of the pre-normalized likelihood before properly normalizing it. With a flat prior of 1, the posterior curve for n = 4 is



The average value for x_k was 0.9253694822156299. The posterior curve for n = 32 is



The average value for x_k was 0.9660020428043852.

More samples focused the curve on the expected value for α, decreasing the value of sigma.

The mean of the data isn't the best estimator for the most probable value of α because it doesn't take into account that not every value is equally likely due to the angle being the random distribution and not x_k.

For two unknown values: α and β, I created a two dimensional array where A and B changed in the equation for likelihood. I made a separate function for likelihood which was

```
def getLikelihood(a, b, data):
    logB = 0
    if (b > 0):
        logB = math.log(b)
    likelihood = logB - math.log(math.pi) - sum(b**2 + (data - a) ** 2)
    return likelihood
```

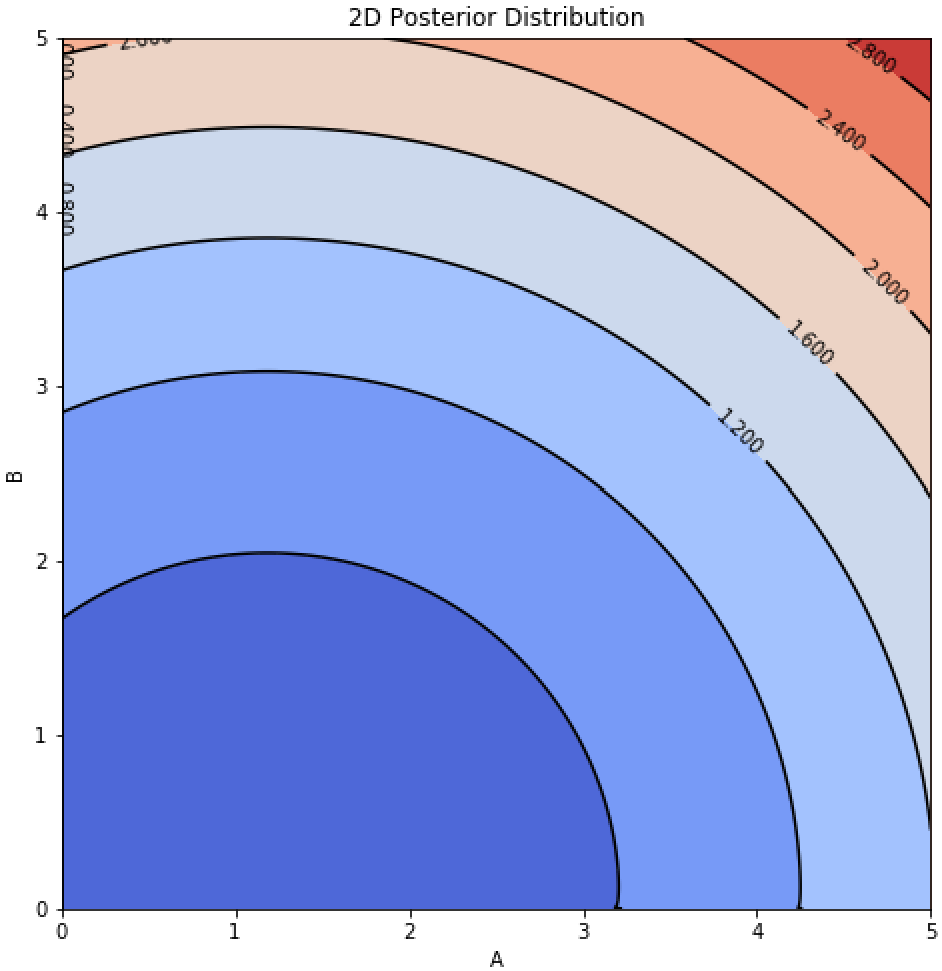
To create the 2 dimensional array, I wrote

```
likelihood = np.array([[getLikelihood(a, b, data)for a in A ]for b in B]
```

I normalized using a similar method as previously. To create the contour plot, I wrote

```
fig = plt.figure(figsize=(8,8))
ax = fig.gca()
xmin, ymin = 0, 0
xmax, ymax = 5, 5
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)
cfset = ax.contourf(A, B, posterior, cmap='coolwarm')
ax.imshow(np.rot90(posterior), cmap='coolwarm', extent=[xmin, xmax, ymin, ymax])
cset = ax.contour(A, B, posterior, colors='k')
ax.clabel(cset, inline=1, fontsize=10)
ax.set_xlabel('A')
ax.set_ylabel('B')
plt.title('2D Posterior Distribution')
```

For n=32, I got the following graph.



The average value for x_k is 1.1572337261307748.