CS589 ASSIGNMENT 1

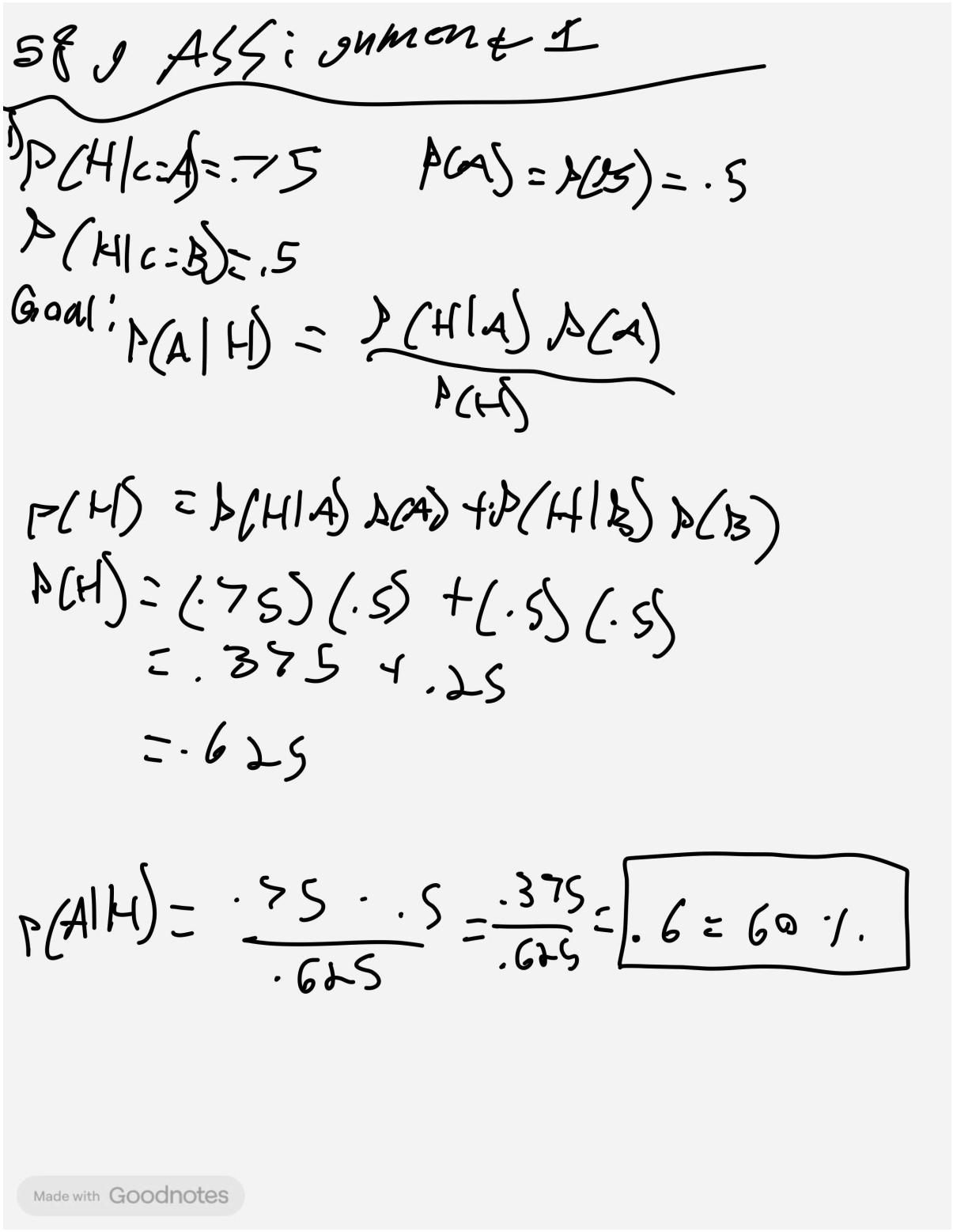
Name: Dorian Benhamou Goldfajn
Email: dbenhamougol@umass.edu
Discussed With: Aryan Nair

In [58]:
```
import pangolin as pg
from matplotlib import pyplot
from IPython.display import Image
```

Question 1

In [59]:
```
Image(filename='IMG_0041.jpg', width=800)
```

Out[59]:



Question 2

In [43]:

```
#A = 1, B = 0
coin = pg.categorical([0.5, 0.5])
# can also do coin = pg.bernoulli(0.5)

#P(H | A) = 0.75, P(H | B) = 0.5
P_H_A = pg.makerv(0.75)
P_H_B = pg.makerv(0.5)
```

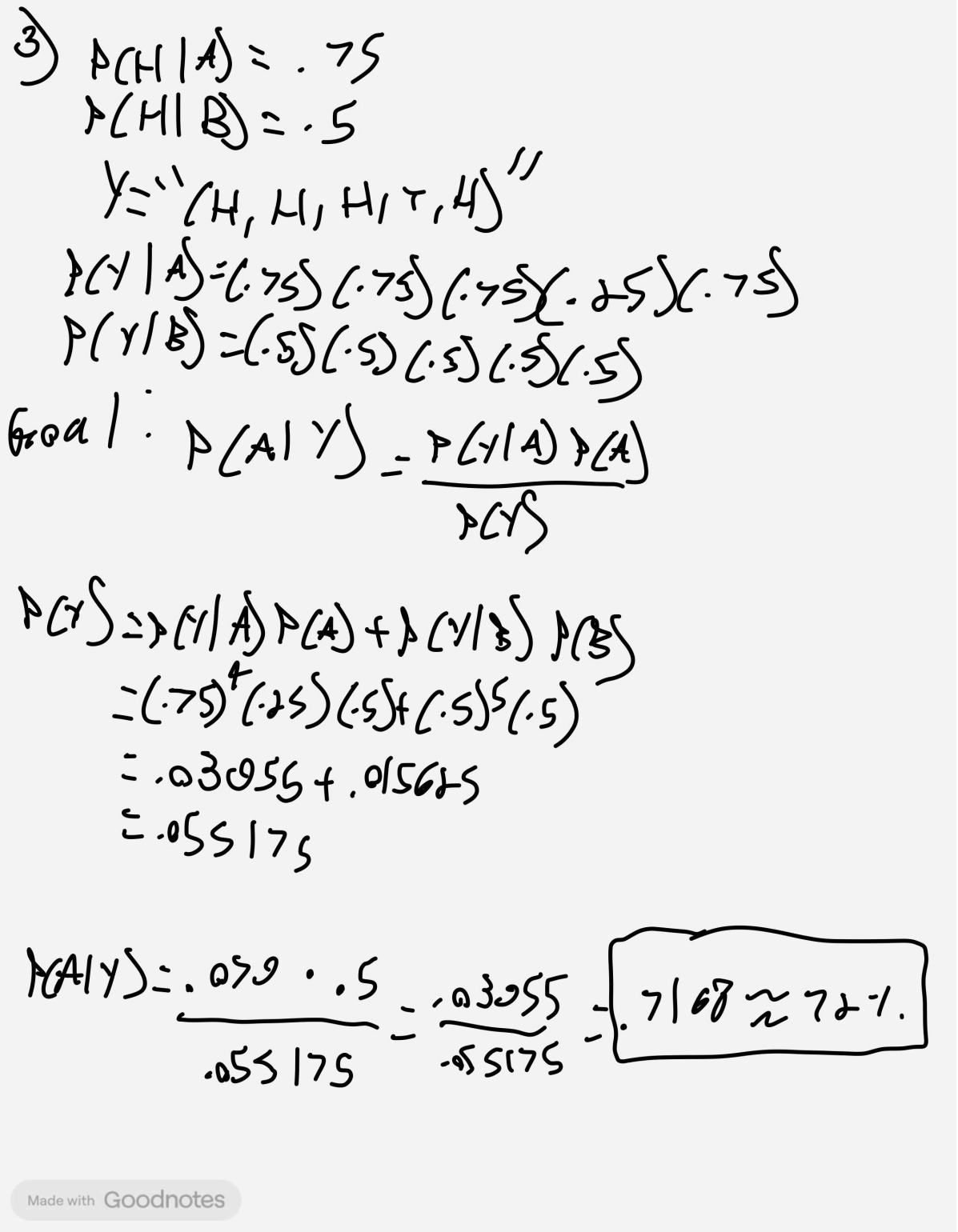
```
prob_heads = (coin * P_H_A + (1 - coin) * P_H_B)
expected_prob_heads = pg.E(prob_heads, niter=1000000)
prob_a = (coin * P_H_A) / expected_prob_heads
pg.E(prob_a)
```

Out[43]: Array(0.60438824, dtype=float32)

Question 3

In [60]:
```
Image(filename='IMG_0042.jpg', width=800)
```

Out[60]:



Question 4

In [42]:

```
coin = pg.categorical([0.5, 0.5])

# Y = H, H, H, T, H
# P(H | A) = 0.75, P(H | B) = 0.5

P_H_A = pg.makerv(0.75)
P_H_B = pg.makerv(0.5)
```

```
prob_Y_a = (P_H_A)**3 * (1 - P_H_A) * (P_H_A)
prob_Y_b = (P_H_B)**3 * (1 - P_H_B) * (P_H_B)

prob_Y = coin * prob_Y_a + (1 - coin) * prob_Y_b
expected_prob_Y = pg.E(prob_Y, niter=1000000)

prob_a = (coin * prob_Y_a) / expected_prob_Y
pg.E(prob_a)
```

Out[42]:
```
Array(0.7185237, dtype=float32)
```

Question 5

In [16]:
```
from scipy.stats import norm
import math
```

In [41]:
```
weights = [i for i in range(1, 6)]
prior_prob = .2

#PDF
probs = [norm.pdf(2.9, loc=i, scale=1) for i in weights]

#Posterior
posteriors = [prior_prob * p for p in probs]

#Normalize
s = sum(posteriors)
normal_posteriors = [p / s for p in posteriors]

normal_posteriors
```

Out[41]:
```
[0.06624585357252528,
 0.2686401832109018,
 0.40076406009289894,
 0.2199439795072686,
 0.04440592361640537]
```

Question 6

In [61]:
```
weights = pg.categorical([.2, .2, .2, .2, .2]) + 1
model_weight = weights
measurement = 2.9

models = [pg.normal(i, 1) for i in range(1, 6)]
samples = [pg.sample(model) for model in models]
count_by_sample = [0 for _ in range(len(samples))]

for i, sample in enumerate(samples):
    for s in sample:
        if s <= 2.95 and s >= 2.85:
            count_by_sample[i] += 1


probs = [count / 10000 for count in count_by_sample]
```

```
posteriors = [.2 * p for p in probs]

s = sum(posteriors)
normal_posteriors = [p / s for p in posteriors]

normal_posteriors
```

Out[61]:
```
[0.07312252964426877,
 0.2707509881422925,
 0.391304347826087,
 0.22826086956521735,
 0.036561264822134384]
```

Question 7

In [48]:
```
# Each weight has diff prob for each measurement

# get the weights and measurements
weights = [i for i in range(1, 6)]
prior_prob = .2
measurements = [2.9, 4.2, 3.5, 2.5]

# get each measurement probability for each weight from PDFs
probs = [[norm.pdf(m, loc=w, scale=1) for m in measurements] for w in weights]

# get each weight probability from measurement probabilities and prior to get poste
posteriors = [math.prod([m for m in w]) * prior_prob for w in probs]

#Normalize
s = sum(posteriors)
normal_posteriors = [p / s for p in posteriors]

normal_posteriors
```

Out[48]:
```
[2.555216750311888e-05,
 0.030968385695699294,
 0.687434716938027,
 0.2794900989593687,
 0.002081246239401805]
```

Question 8

In [50]:
```
# prior
prior_prob = .2

# measurements
measurements = [2.9, 4.2, 3.5, 2.5]

# model for each weight 1 - 5
models = [pg.normal(w, 1) for w in range(1, 6)]

counts = [[0 for i in range(len(measurements))] for j in range(len(models))]
probs = []

# calculate each measurment probability for each weight
```

```
for i, model in enumerate(models):
    sample = pg.sample(model)
    for j, measurement in enumerate(measurements):
        for s in sample:
            if s <= measurement + .05 and s >= measurement - .05:
                    counts[i][j] += 1
    probs.append([count / 10000 for count in counts[i]])

# calculate each weight probability from measurement probabilities and prior to get
posteriors = [math.prod(p) * prior_prob for p in probs]

# normalize
s = sum(posteriors)
normal_posteriors = [p / s for p in posteriors]

normal_posteriors
```

Out[50]:
```
[3.363772211453114e-05,
 0.03001396499487237,
 0.6706392259831316,
 0.2978299815487276,
 0.0014831897511538574]
```

Question 9

In [38]:
```
weight = pg.uniform(1,5)
measurements = [2.9, 4.2, 3.5, 2.5]

# Get sample weights, this could be done by either directly sampling from the unifo

# sample_weights = [w for w in pg.sample(weight, niter = 100)]

i = 1
sample_weights = [i + .04 * j for j in range(0, 100)]

# model for each weight taking noise into consideration
models = [pg.normal(w, 1) for w in sample_weights]

counts = [[0 for i in range(len(measurements))] for j in range(len(models))]
probs = []

#calculate each measurement probability for each weight
for i, model in enumerate(models):
    sample = pg.sample(model)
    for j, measurement in enumerate(measurements):
        for s in sample:
            if s <= measurement + .05 and s >= measurement - .05:
                    counts[i][j] += 1
    probs.append([count / 10000 for count in counts[i]])

#calculate each weight's probability by multiplying the probabilities of each measu
posteriors = [math.prod(p) * 1/len(sample_weights) for p in probs]

# normalize
s = sum(posteriors)
normal_posteriors = [p / s for p in posteriors]
```

```
# sample from the posterior
normal_posteriors_rv = pg.categorical(normal_posteriors)
posteriors_sample = pg.sample(normal_posteriors_rv)

#map indices to weights
mapped_sample = [sample_weights[s] for s in posteriors_sample]

#plot the histogram
pyplot.hist(mapped_sample, bins=100, density=True)
pyplot.show()
```