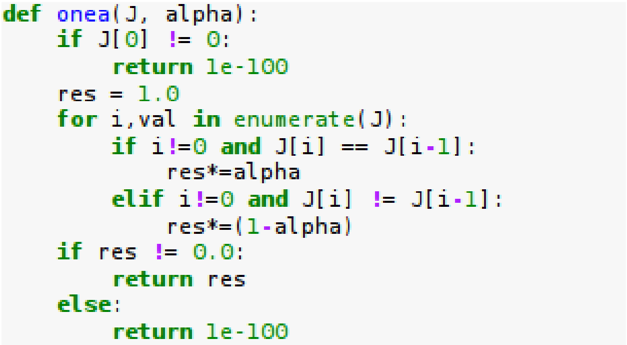
Homework - IV

Siddharth Chandrasekaran

November 21, 2017

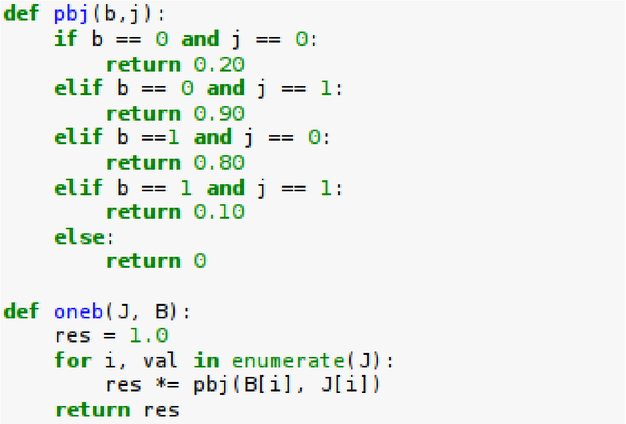
Answer 1.a. 1. Code Snippet for P(J/ α) :



```
def onea(J, alpha):
    if J[0] != 0:
        return 1e-100
    res = 1.0
    for i,val in enumerate(J):
        if i!=0 and J[i] == J[i-1]:
            res*=alpha
        elif i!=0 and J[i] != J[i-1]:
            res*=(1-alpha)
    if res != 0.0:
        return res
    else:
        return 1e-100
```

2. Sample test cases:
Result for 1a for input1 [0 1 1 0 1] 0.75: 0.01171875
Result for 1a for input2 [0 0 1 0 1] 0.2: 0.10240000000000003
3. Code output for:
Result for 1a for input3 [1 1 0 1 0 1] 0.2: 1e-100
Result for 1a for input4 [0 1 0 1 0 0] 0.2: 0.08192000000000003

Answer 1.b. 1. Code Snippet P(B/J) :



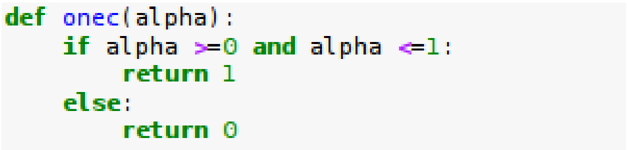
```
def pbj(b,j):
    if b == 0 and j == 0:
        return 0.20
    elif b == 0 and j == 1:
        return 0.90
    elif b ==1 and j == 0:
        return 0.80
    elif b == 1 and j == 1:
        return 0.10
    else:
        return 0

def oneb(J, B):
    res = 1.0
    for i, val in enumerate(J):
        res *= pbj(B[i], J[i])
    return res
```

2. Sample test cases:
Result for 1b for input1 [0 1 1 0 1] [1 0 0 1 1]: 0.05184000000000001
Result for 1b for input2 [0 1 0 0 1] [0 0 1 0 1]: 0.0028800000000000006

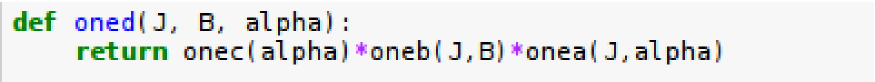
3. Code output for:
Result for 1b for input3 [0 1 1 0 0 1] [1 0 1 1 1 0]: 0.041472000000000001
Result for 1b for input4 [1 1 0 0 1 1] [0 1 1 0 1 1]: 0.00014400000000000003

Answer 1.c. 1. Code Snippet P(α) :



```
def onec(alpha):
    if alpha >=0 and alpha <=1:
        return 1
    else:
        return 0
```

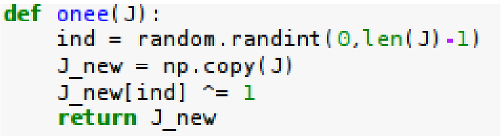
Answer 1.d. 1. Code Snippet P(α,J,B):



```
def oned(J, B, alpha):
    return onec(alpha)*oneb(J,B)*onea(J,alpha)
```

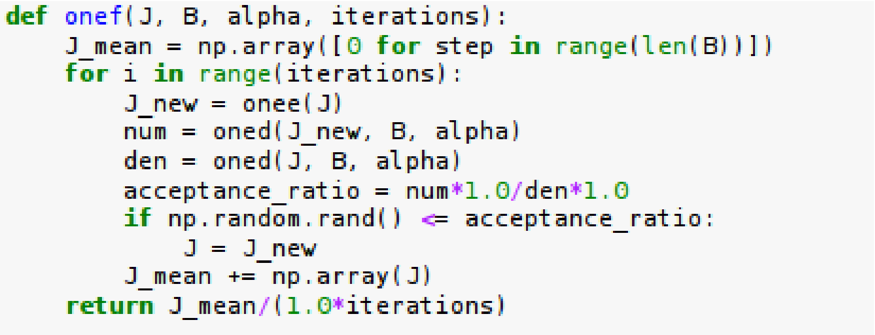
2. Sample test cases:
Result for 1d for input1 [0 1 1 0 1] [1 0 0 1 1] 0.75: 0.0006075000000000002
Result for 1d for input2 [0 1 0 0 1] [0 0 1 0 1] 0.3: 0.00029635199999999994
3. Code output for:
Result for 1d for input3 [0 0 0 0 0 1] [0 1 1 1 0 1] 0.63: 0.00011936963727360005
Result for 1d for input4 [0 0 1 0 0 1 1] [1 1 0 0 1 1 1] 0.23: 5.1191534693376025e-06

Answer 1.e. 1. Code Snippet for proposal:



```
def onee(J):
    ind = random.randint(0,len(J)-1)
    J_new = np.copy(J)
    J_new[ind] ^= 1
    return J_new
```

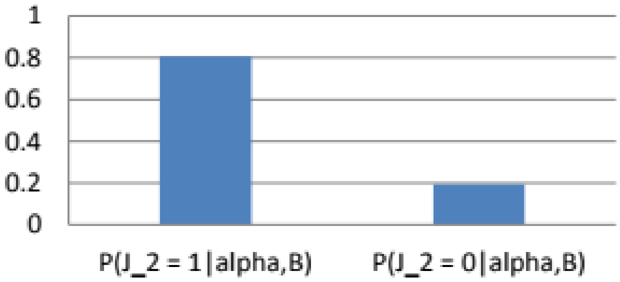
Answer 1.f. 1. Code Snippet for Sampling P(J/α,B):



```
def onef(J, B, alpha, iterations):
    J_mean = np.array([0 for step in range(len(B))])
    for i in range(iterations):
        J_new = onee(J)
        num = oned(J_new, B, alpha)
        den = oned(J, B, alpha)
        acceptance_ratio = num*1.0/den*1.0
        if np.random.rand() <= acceptance_ratio:
            J = J_new
        J_mean += np.array(J)
    return J_mean/(1.0*iterations)
```

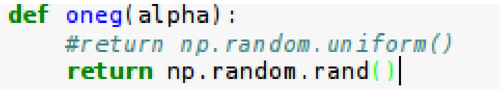
2. Sample test cases:
Result for 1f for input1 [0 0 0 0 0] [1 0 0 1 1] 0.5 10000: [0. 0.8165 0.8062 0.0953 0.1073]
3. Bar chart for P(J_2/α,B) :

Bar chart for $J_2 = 1$ and $J_2 = 0$ given alpha and B for the given case



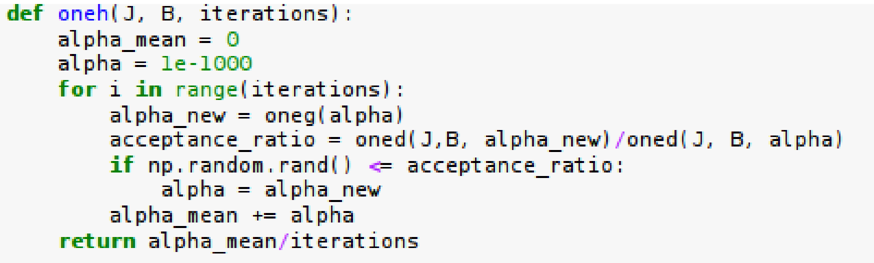
4. Code output for:
Result for 1f for input2 [0 0 0 0 0 0 0 0] [1 0 0 0 1 0 1 1] 0.11 10000: [0. 0.9705 0.1009 0.9748 0.0131 0.9733 0.0848 0.4548]
Result for 1f for input3 [0 0 0 0 0 0 0] [1 0 0 1 1 0 0] 0.75 10000: [0. 0.6719 0.6903 0.1231 0.1223 0.7621 0.8433]

Answer 1.g. Code to propose new α :



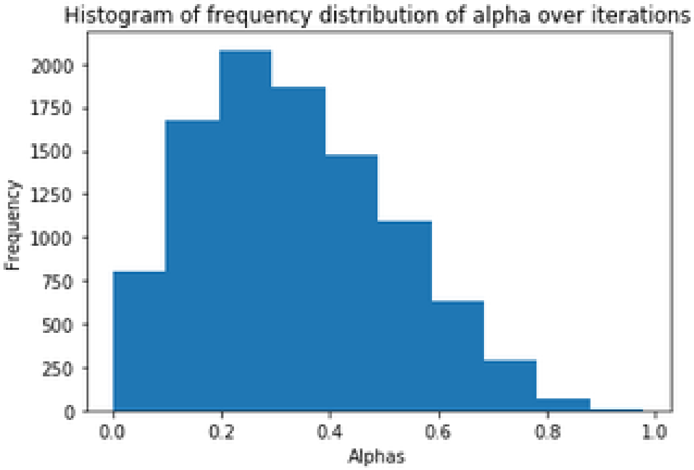
```
def oneg(alpha):
    #return np.random.uniform()
    return np.random.rand()
```

Answer 1.h. 1. Code Snippet for Sampling P(α/J,B):



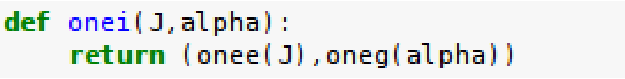
```
def oneh(J, B, iterations):
    alpha_mean = 0
    alpha = 1e-1000
    for i in range(iterations):
        alpha_new = oneg(alpha)
        acceptance_ratio = oned(J,B, alpha_new)/oned(J, B, alpha)
        if np.random.rand() <= acceptance_ratio:
            alpha = alpha_new
        alpha_mean += alpha
    return alpha_mean/iterations
```

2. Sample test cases:
Result for 1h for input1 [0 1 0 1 0] [1 0 1 0 1] 10000: 0.16653513027034067
Result for 1h for input2 [0 0 0 0 0] [1 1 1 1 1] 10000: 0.8370726987475701
3. Histogram for P(α/J,B) for input3 [0 1 1 0 1] [1 0 0 1 1] 10000



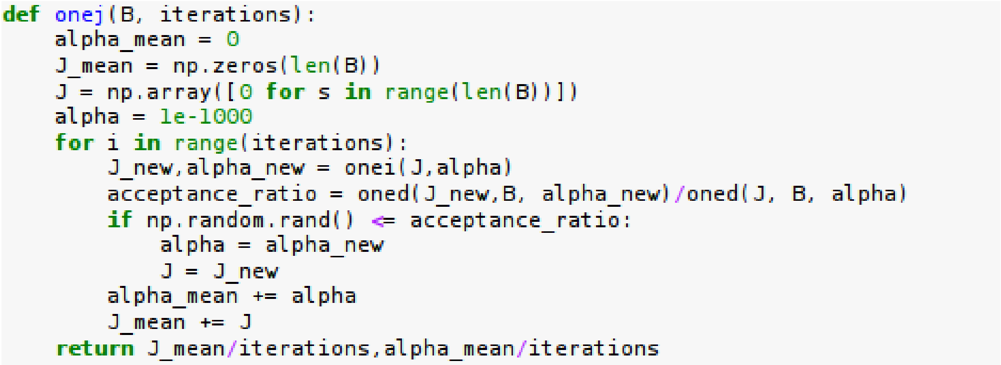
4. Code output for:
Result for 1h for input4 [0 1 1 1 1 1 1 0] [1 0 0 1 1 0 0 1] 10000: 0.6686126831412196
Result for 1h for input5 [0 1 1 0 1 0] [1 0 0 1 1 1] 10000: 0.2831392835665674

Answer 1.i. Code to propose new α and J:



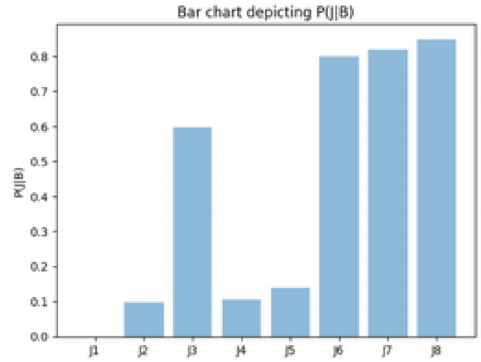
```
def onei(J,alpha):
    return (onee(J),oneg(alpha))
```

Answer 1.j. 1. Code Snippet for Sampling P(α/J,B):

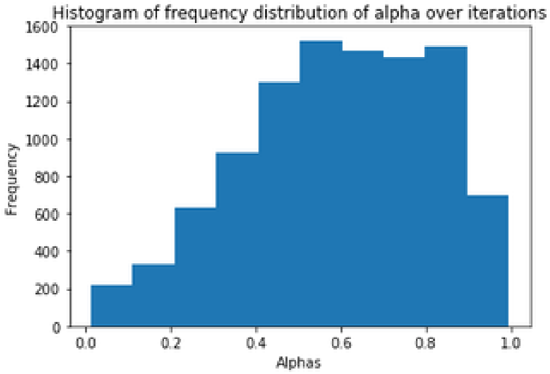


```python
def onej(B, iterations):
    alpha_mean = 0
    J_mean = np.zeros(len(B))
    J = np.array([0 for s in range(len(B))])
    alpha = 1e-1000
    for i in range(iterations):
        J_new,alpha_new = onei(J,alpha)
        acceptance_ratio = oned(J_new,B, alpha_new)/oned(J, B, alpha)
        if np.random.rand() <= acceptance_ratio:
            alpha = alpha_new
            J = J_new
        alpha_mean += alpha
        J_mean += J
    return J_mean/iterations,alpha_mean/iterations
```

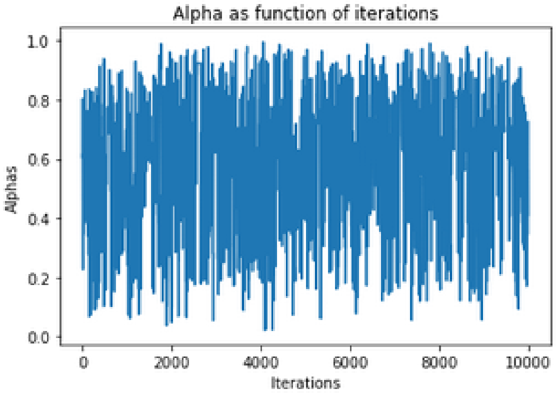
2. For the input B = [1,1,0,1,1,0,0,0]
Barcharts for P(J/B) corresponding to the mean values:



Histogram for P(α/B):

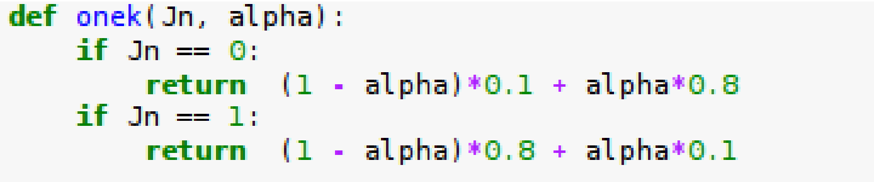


Plot α as a function of iterations:



3. The first graph shows us the probability of J0 being 1 is 0 since we start with the white ball. The histogram looks like it follows normal distribution as expected. The graph for alpha as function of iterations looks random and all over the place (since alpha is selected using a random function).

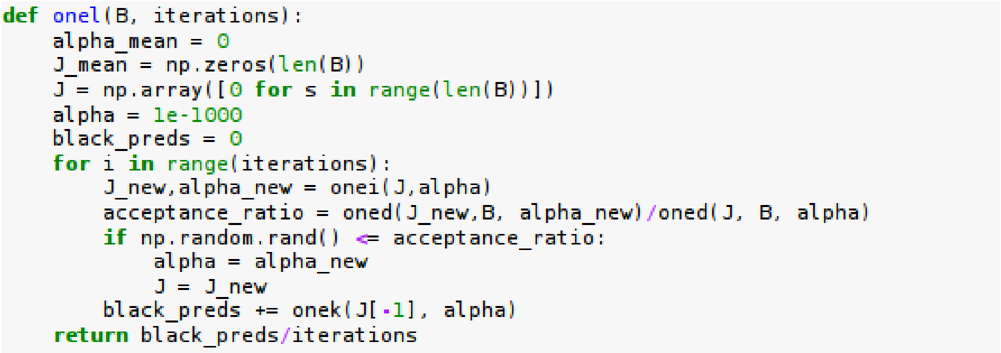
Answer 1.k. 1. Code Snippet finding the probability of black ball given α and J_n:



```python
def onek(Jn, alpha):
    if Jn == 0:
        return  (1 - alpha)*0.1 + alpha*0.8
    if Jn == 1:
        return  (1 - alpha)*0.8 + alpha*0.1
```

2. Sample test cases:
Result for 1k for input1 1 0.6:0.38000000000000006
Result for 1k for input2 0 0.99:0.793
3. Code output for:
Result for 1k for input3 0 0.33456:0.33419200000000004
Result for 1k for input4 1 0.5019:0.44867

Answer 1.l. 1. Code Snippet to compute the probability of N + 1 ball is black:



```python
def onel(B, iterations):
    alpha_mean = 0
    J_mean = np.zeros(len(B))
    J = np.array([0 for s in range(len(B))])
    alpha = 1e-1000
    black_preds = 0
    for i in range(iterations):
        J_new,alpha_new = onei(J,alpha)
        acceptance_ratio = oned(J_new,B, alpha_new)/oned(J, B, alpha)
        if np.random.rand() <= acceptance_ratio:
            alpha = alpha_new
            J = J_new
        black_preds += onek(J[-1], alpha)
    return black_preds/iterations
```

2. Sample test cases:
Result for 1l for input1 [0, 0, 1] 10000:0.3338230117685613
Result for 1l for input2 [0, 1, 0, 1, 0, 1] 10000:0.3998763453618859
3. Code output for:
Result for 1l for input3 [0, 1, 0, 0, 0, 0, 0] 10000:0.34355064023587445
Result for 1l for input4 [1, 1, 1, 1, 1] 10000:0.6612825823993896

Answer 1.m. Kaggle Username : Siddharth Chandrasekaran
1. Tried 10000 iterations
2. Tried 100000 iterations
3. Tried 1000000 iterations
The approach that worked best was with 1000000 iterations. The kaggle accuracy score was 0.00045.