CS237 Lab 3

Vidya Akavoor

Collaborators: Lauren DiSalvo, Sreeja Keesara, Kathryn Quirk, Nathan Weinberg

Late Days Used: 0

Late Days Left: 3

Sources:

http://stackoverflow.com/questions/4941753/is-there-a-math-ncr-function-in-python

https://onlinecourses.science.psu.edu/stat414/node/76

https://www.wolframalpha.com

http://math.stackexchange.com/questions/206050/how-do-i-tell-if-this-function-is-a-probability-density-function

https://docs.python.org/2/library/binascii.html

http://stackoverflow.com/questions/13428318/reading-rows-from-a-csv-file-in-python

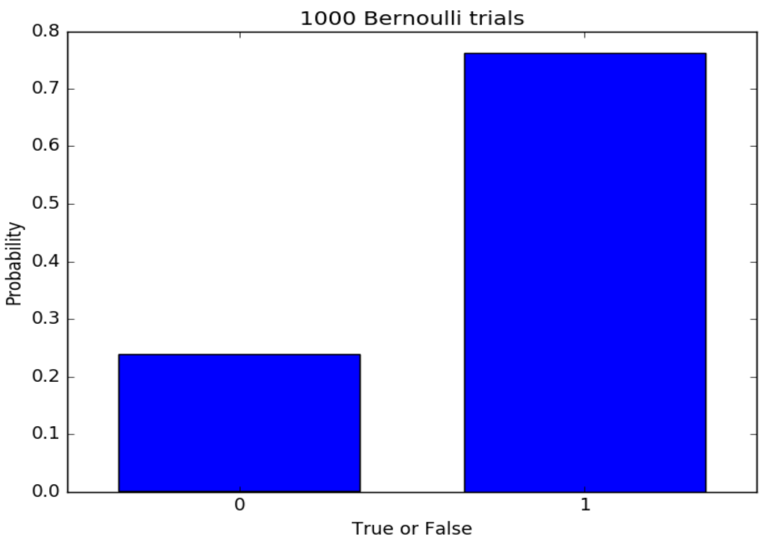
https://en.wikipedia.org/wiki/Poisson_distribution

Part 2 – Bernoulli Distribution

1)
```
def Bernoulli_trial(p):
    choice = random.random()
    if choice <= p:
        return 1
    else:
        return 0
```

2)
```
def Bernoulli_hist(p, m):
    result_list = [0 for x in range(m)]
    for i in range(1000):
        result_list[i] = Bernoulli_trial(p)
    plt.figure(1)
    plt.hist(result_list, bins = 2, rwidth = .7, align = 'mid',\
    weights = np.zeros_like(result_list) + 1. / len(result_list))
    plt.xlim(0, 1)
```

```
ax1 = plt.axes()
ax1.set_xticks([.25, .75])
ax1.set_xticklabels([0, 1])
plt.title("1000 Bernoulli trials")
plt.xlabel("True or False")
plt.ylabel("Probability")
plt.show()
```



<u>Part 3 – Geometric</u>

3) $X \sim G(p)$ is a probability distribution

PROOF: $X \sim G(p)$ means $Pr[X = i]$ with param $p \to p(1-p)^i$

for $X \sim G(p)$ to be prob. distribution, $\sum_{i=1}^{k} p(1-p)^{i+1} = 1$

$\lim_{i\to\infty} \sum_{i=0}^{k-1} p(1-p)^i = \lim_{i\to\infty} p \sum_{i=0}^{k-1} (1-p)^i = \lim_{i\to\infty} p\left(\frac{1-(1-p)^i}{1-(1-p)}\right)$

as $i \to \infty$, $(1-p)^i \to 0$ so $= \lim_{i\to\infty} \frac{p}{p} = \boxed{1}$

4) $\lim_{i\to\infty} \sum_{i=0}^{k} (1-p)^i = \lim_{i\to\infty} \frac{1-(1-p)^i}{1-(1-p)}$

as $i \to \infty$, $(1-p)^i \to 0$ so $= \boxed{\frac{1}{p}}$

(4 follows same logic as 3 without the p in front)

Part 4 – Binomial

5)
```
def binomial_draw(n,p):
    count = 0
    for x in range(n):
        if Bernoulli_trial(p):
            count += 1
    return count
```

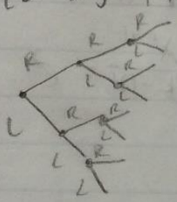
6)
```
def binom_trials(n, p, numExpts):
    trial_list = [0 for x in range(numExpts)]
        for i in range(len(trial_list)):
            trial_list[i] = binomial_draw(n,p)
    return trial_list
```

7)
```
def binom_hist(n, p, numExpts):
    y_coords = binom_trials(n, p, numExpts)
    plt.figure(1)
    plt.hist(y_coords, bins = n, rwidth = .7, align = 'mid',\
    weights = np.zeros_like(y_coords) + 1. / len(y_coords))
    plt.xlim(0, max(y_coords))
    plt.title("1000 Binomial trials")
    plt.xlabel("Number of Successes")
    plt.ylabel("Probability")
    plt.show()
```

8)

8) A. L has range {0, 1, 2, 3} because Scarface takes 3 total steps, each of which can be a left step

B. Pr[L = l] for all l in range(L) → $\{\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}\}$



RRR, RRL, RLR, RLL, LRR, LRL, LLR, LLL each have $\frac{1}{8}$ probability, then just add the cases where L = 0, then L = 1, L = 2, and L = 3 (separately).

Ans: $Pr[L = 0] = \frac{1}{8}$, $Pr[L = 1] = \frac{3}{8}$, $Pr[L = 2] = \frac{3}{8}$, $Pr[L = 3] = \frac{1}{8}$

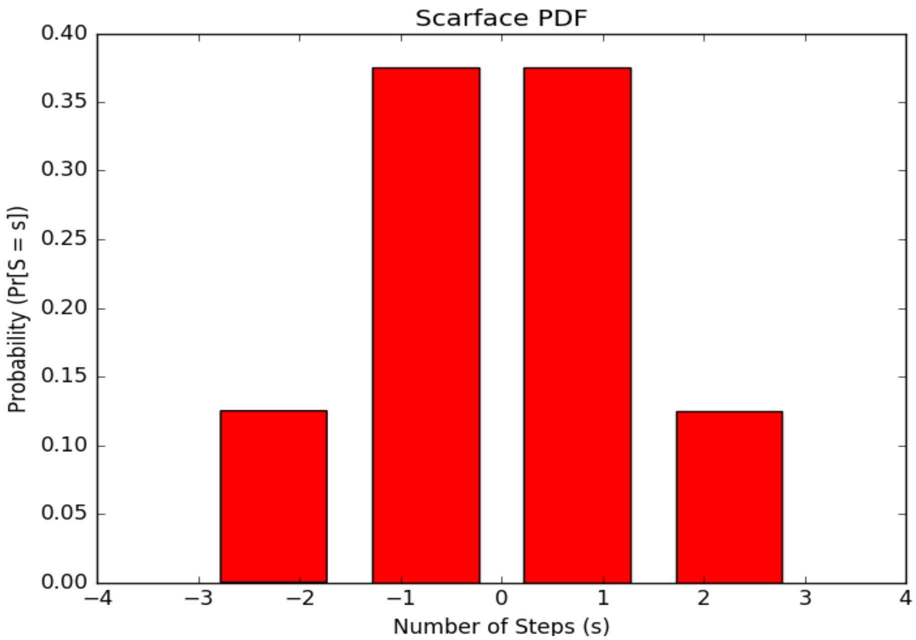
C. If L = 2, S = -1 because 2 left steps (-2) plus 1 right step (+1) is -1.

D. $S = -2L + 3$

E. range(S) = {-3, -1, 1, 3}

F. Pr[S = s] ∀ s in range(S) → $Pr[S = -3] = \frac{1}{8}$, $Pr[S = -1] = \frac{3}{8}$, $Pr[S = 1] = \frac{3}{8}$, $Pr[S = 3] = \frac{1}{8}$

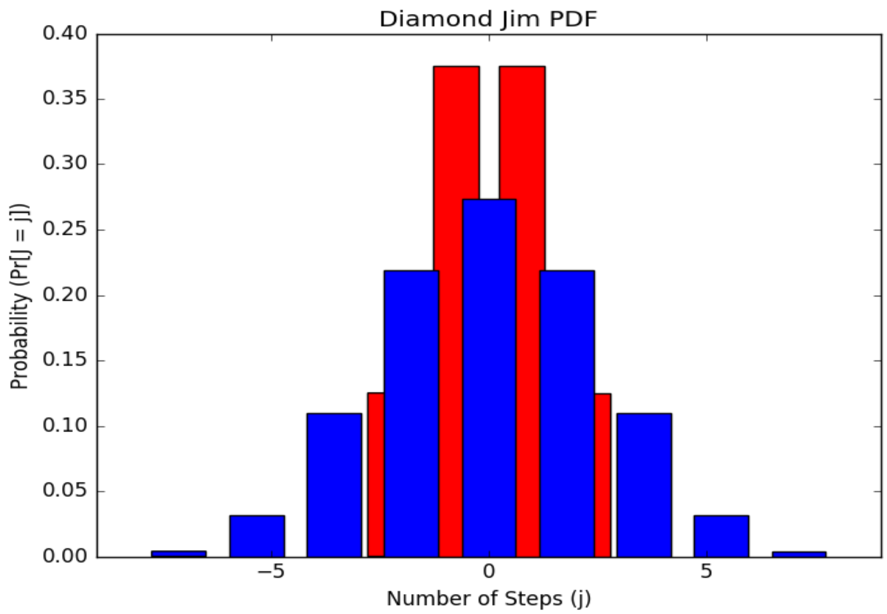
g.
```
def S_step_prob(steps):
      answers = []
      for L in range(steps+1):
            S = -2*L + steps
            choose = nCr(steps, L)
            for x in range(choose):
                  answers += [S]
      plt.figure(1)
      plt.hist(answers, bins = steps+1, rwidth = .7, align = 'mid',\
      weights = np.zeros_like(answers) + 1. / len(answers), color = 
'r')
      plt.xlim(-(steps+1), (steps+1))
      plt.title("Scarface PDF")
      plt.xlabel("Number of Steps (s)")
      plt.ylabel("Probability (Pr[S = s])")
     plt.show()
```



9)

9) A. $Pr[J=-8] = \frac{1}{256}$, $Pr[J=-6] = \frac{8}{256}$, $Pr[J=-4] = \frac{28}{256}$, $Pr[J=-2] = \frac{56}{256}$,
$Pr[J=0] = \frac{70}{256}$, $Pr[J=2] = \frac{56}{256}$, $Pr[J=4] = \frac{28}{256}$, $Pr[J=6] = \frac{8}{256}$
$Pr[J=8] = \frac{1}{256}$

b.
```
def J_step_prob(steps):
    answers = []
    for L in range(steps+1):
        J = -2*L + steps
        choose = nCr(steps, L)
        answers += [J]*choose
    plt.figure(1)
    plt.hist(answers, bins = steps+1, rwidth = .7, align = 'mid',\
    weights = np.zeros_like(answers) + 1. / len(answers), color = 
'b')
    plt.xlim(-(steps+1), (steps+1))
    plt.title("Diamond Jim PDF")
    plt.xlabel("Number of Steps (j)")
    plt.ylabel("Probability (Pr[J = j])")
    plt.show()
```



10) – 13)

10) $Pr[D \mid S=1] = (Pr[S=1] \cdot Pr[J=2]) + (Pr[S=1] \cdot Pr[J=0]) = \boxed{\frac{189}{1024}}$

11) $Pr[D \mid S=-1] = (Pr[S=-1])(Pr[J=0] + Pr[J=-2]) = \boxed{\frac{189}{1024}}$

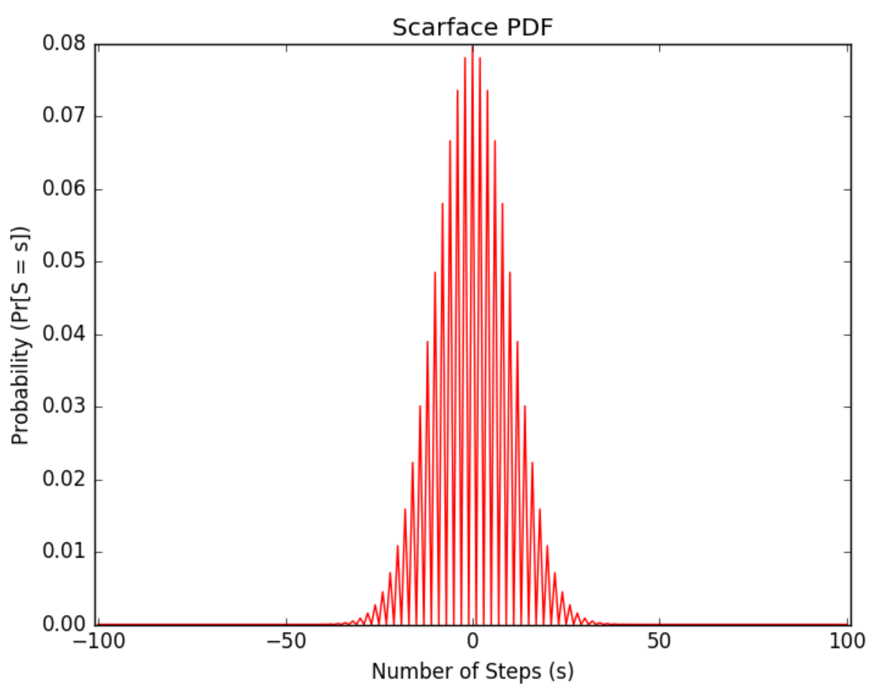
12) $Pr[D \mid S=3] = (Pr[S=3])(Pr[J=4] + Pr[J=2]) = \boxed{\frac{21}{512}}$

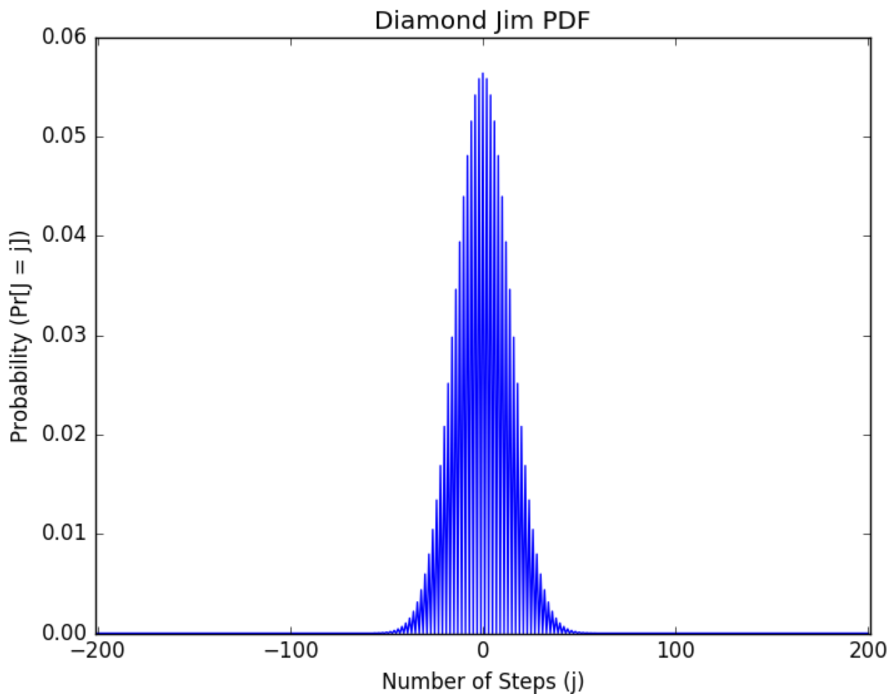
13) $Pr[D] = Pr[D \mid S=-1]\,Pr[S=-1] + Pr[D \mid S=1]\,Pr[S=1] + Pr[D \mid S=3]\,Pr[S=3] + Pr[D \mid S=-3]\,Pr[S=-3] = \boxed{\frac{609}{4096}}$

14)
```
def S_step_pdf(steps):
    plt.figure(6)
    xcoord = range(-steps, (steps+1))
    ycoord = [0] * ((2*steps) + 1)
    for i in range(steps):
        S = (-2*i) + steps
        prob = d((nCr(steps, i))) / d(2**steps)
        ycoord[steps+ S] = prob
    plt.plot(xcoord, ycoord, linestyle = '-', color = 'r')
    plt.xlim(-(steps+1), (steps+1))
    plt.title("Scarface PDF")
    plt.xlabel("Number of Steps (s)")
    plt.ylabel("Probability (Pr[S = s])")
    plt.show()
```

```
def J_step_pdf(steps):
    plt.figure(2)
    xcoord = range(-steps, (steps+1))
    ycoord = [0] * ((2*steps) + 1)
    for i in range(steps):
        J = (-2*i) + steps
        prob = d((nCr(steps, i))) / d(2**steps)
        ycoord[steps+ J] = prob
    plt.plot(xcoord, ycoord, linestyle = '-', color = 'b')
    plt.xlim(-(steps+1), (steps+1))
    plt.title("Diamond Jim PDF")
    plt.xlabel("Number of Steps (j)")
    plt.ylabel("Probability (Pr[J = j])")
    plt.show()
```

(I only made separate functions to keep the labels different)






```
def prob_D(stepsS, stepsJ):
    distJ = J_step_pdf(stepsJ)
    d_prob = 0
    for i in range(stepsS+1):
        total_steps = -i + (stepsS-i)
        temp = d(distJ[stepsJ + total_steps])
        temp += d(distJ[stepsJ + total_steps+1])
        temp += d(distJ[stepsJ + total_steps+2])
        temp += d(distJ[stepsJ + total_steps-1])
        temp += d(distJ[stepsJ + total_steps-2])
        temp += d(distJ[stepsJ + total_steps+1])
        temp *= (d(nCr(stepsS, i))/d(2**stepsS))**2
        d_prob += temp
    print d_prob
```

>>Pr[D] = 0.008473905769112531648588157686 (or approximately .8%)

15)

15) for $X \sim B(n,p)$, $E[X] = np$

PROOF: if $X \sim B(n,p)$ then X is a sum of Bernoullis, X_i's with param p.

$E[X_i] = 0(1-p) + 1p = p$

if $X = \sum_{i=1}^{n} X_i$ then $E[X] = E[\sum_{i=1}^{n} X_i]$

because of linearity of expectation $\rightarrow E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$

$\sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} p = np$

therefore $\rightarrow \boxed{E[X] = np}$

<u>Part 5 – Bitcoin</u>

16) The probability of getting 71 leading zeros is (½)^(71) because it is the same as the probability of flipping a coin 71 times and getting all tails.

17) The number of blocks confirmed per hour is the number of hashes per hour (3600 * (2^18)) multiplied by the probability of getting the right nonce ((½)^(71)) which equals approximately 3.0493 blocks. The number of blocks confirmed per day is the number of hashes per day (24 * 3600 * (2^18)) multiplied by the probability of getting the right nonce ((½)^(71)) which equals approximately 7.3184 blocks.

18)

18) If $Pr[Y=k] = \frac{e^{-\lambda}\lambda^k}{k!}$, then $E[Y] = \lambda$

PROOF:

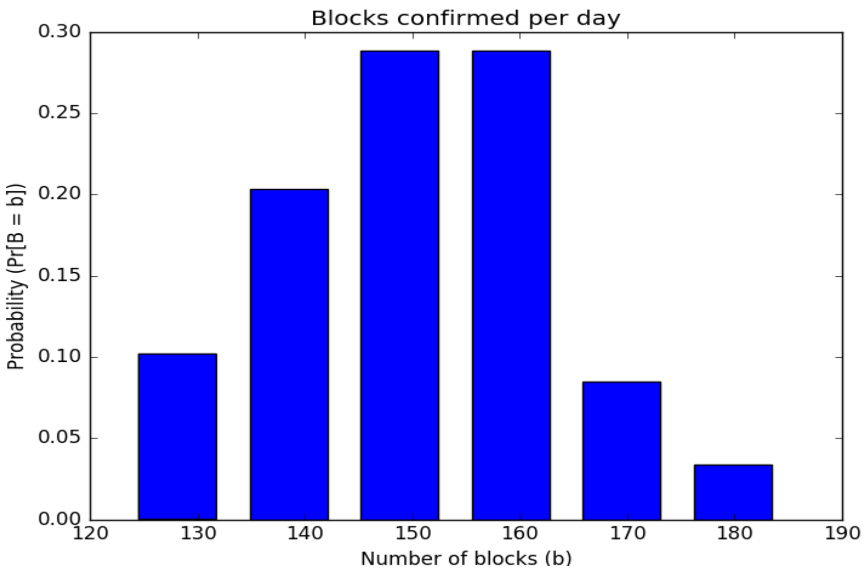
$$E[Y] = \sum_{k=0}^{\infty} k \frac{e^{-\lambda}\lambda^k}{k!} = e^{-\lambda}\lambda \sum_{k=0}^{\infty} \frac{\lambda^{k-1}}{(k-1)!}$$

Since $e^x = \sum_{r=1}^{\infty} \frac{x^r}{r!} \rightarrow \sum_{k=1}^{\infty} \frac{\lambda^{k-1}}{(k-1)!} = e^{\lambda}$

$= e^{-\lambda}\lambda e^{\lambda}$

$= \boxed{\lambda}$

19)



20)

```
def hash_exp(num_zeros):
    while True:
        s = random.randint(0, 99999999999)
        hash_val = hashlib.sha256(b'wubba lubba dub\
            dub'+str(s)).hexdigest()
        hash_val2 = int(hash_val, 16)
        if hash_val2 < (2**(256-num_zeros)):
            return s, hash_val
    return s
```

(Answer for 15 leading zeros)

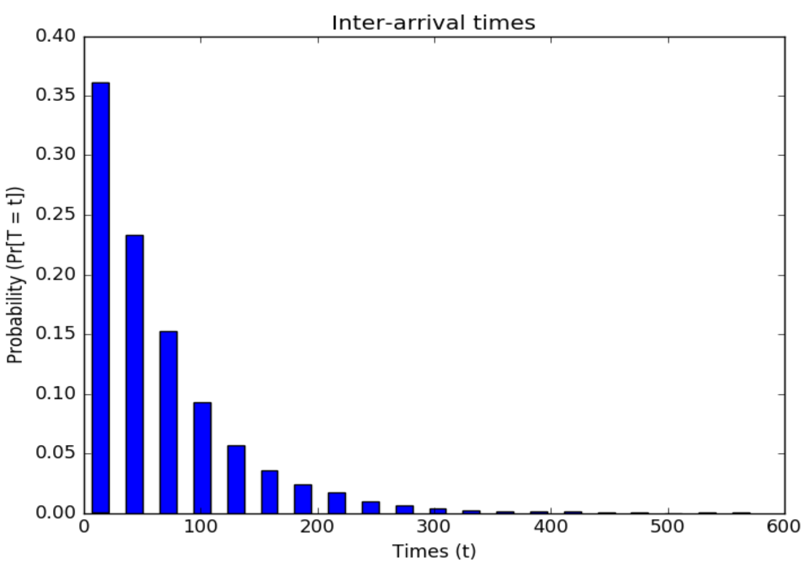
```
>>> hash_exp(15)
(42586678054L, '00018bfccc0d5da6d0fba6550cfbc09d05182ff98897899d04aa9f19b451121a')
```

21)

```
def fake_hash_exp(num_zeros, trials):
    v1 = [0]*trials
    for i in range(trials):
        hash_val = random.randint(0, (2**(256))-1)
        if hash_val < (2**(256-num_zeros)):
            v1[i] = 1
    return v1
```

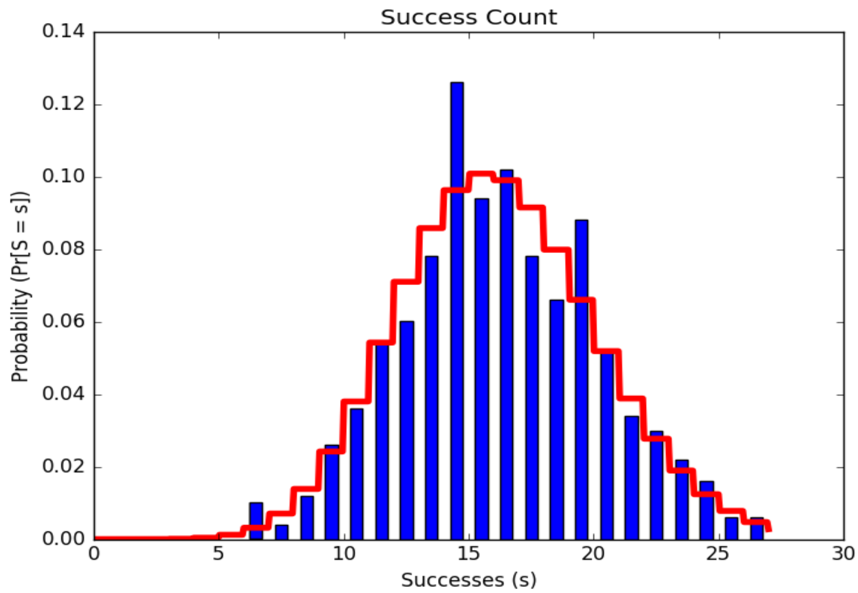
22)
```
def inter_arrival():
    v1 = fake_hash_exp(6, 500000)
    count = 0
    v2 = []
    for i in range(len(v1)):
        if v1[i] == 0:
            count += 1
        else:
            v2 += [count]
            count = 0
    return v2
```

23)
```
def inter_arrival_hist():
    v2 = inter_arrival()
    plt.figure(4)
    plt.hist(v2, bins = 20, rwidth = .5, align = 'mid',\
    weights = np.zeros_like(v2) + 1. / len(v2), color = 'b')
    plt.title("Inter-arrival times")
    plt.xlabel("Times (t)")
    plt.ylabel("Probability (Pr[T = t])")
    plt.show()
```



24) (and 25)

```
def success_count_hist():
    v1 = fake_hash_exp(6,500000)
    count = v1[0]
    v3= []
    i = 1
    while i<len(v1):
        if i%1000 == 0 and i !=0:
            count += v1[i]
            v3 += [count]
            count = 0
        else:
            count += v1[i]
        i += 1
    v3+=[count]
    plt.figure(5)
    plt.hist(v3, bins = max(v3)-min(v3), rwidth = .5, align = 'mid',\
    weights = np.zeros_like(v3) + 1. / len(v3), color = 'b')
    lamda = np.mean(v3)
    plt.title("Success Count")
    plt.xlabel("Successes (s)")
    plt.ylabel("Probability (Pr[S = s])")
    x = np.linspace(0, max(v3), len(v3))
    length = len(x)
    y = []
    for i in range(length):
        y += [((math.e**-\
           lamda))*(lamda**int(x[i])))/math.factorial(int(x[i]))]
    plt.plot(x, y, linestyle = '-', color = 'r', linewidth = 4)
    plt.draw()
    plt.show()
```



Success Count
Probability (Pr[S = s])
Successes (s)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
5
10
15
20
25
30