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])

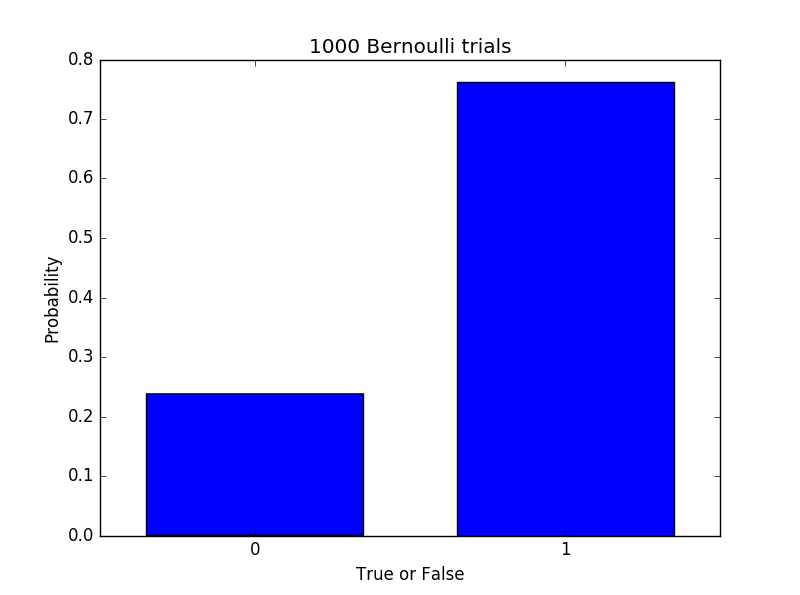
ax1.set\_xticklabels([0, 1])

plt.title("1000 Bernoulli trials")

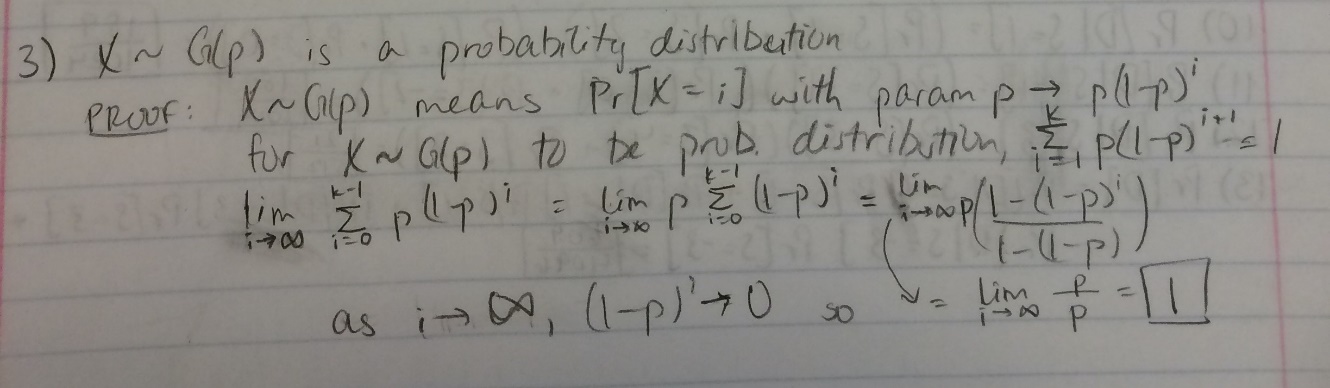
plt.xlabel("True or False")

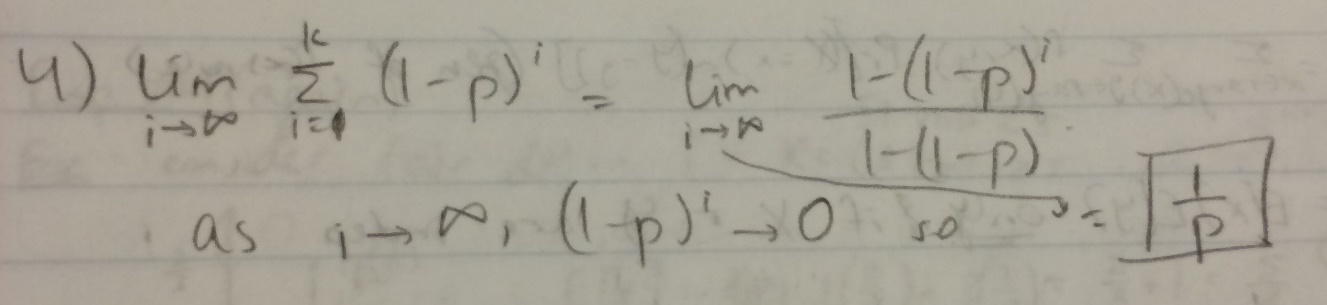
plt.ylabel("Probability")

plt.show()



Part 3 – Geometric





(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))

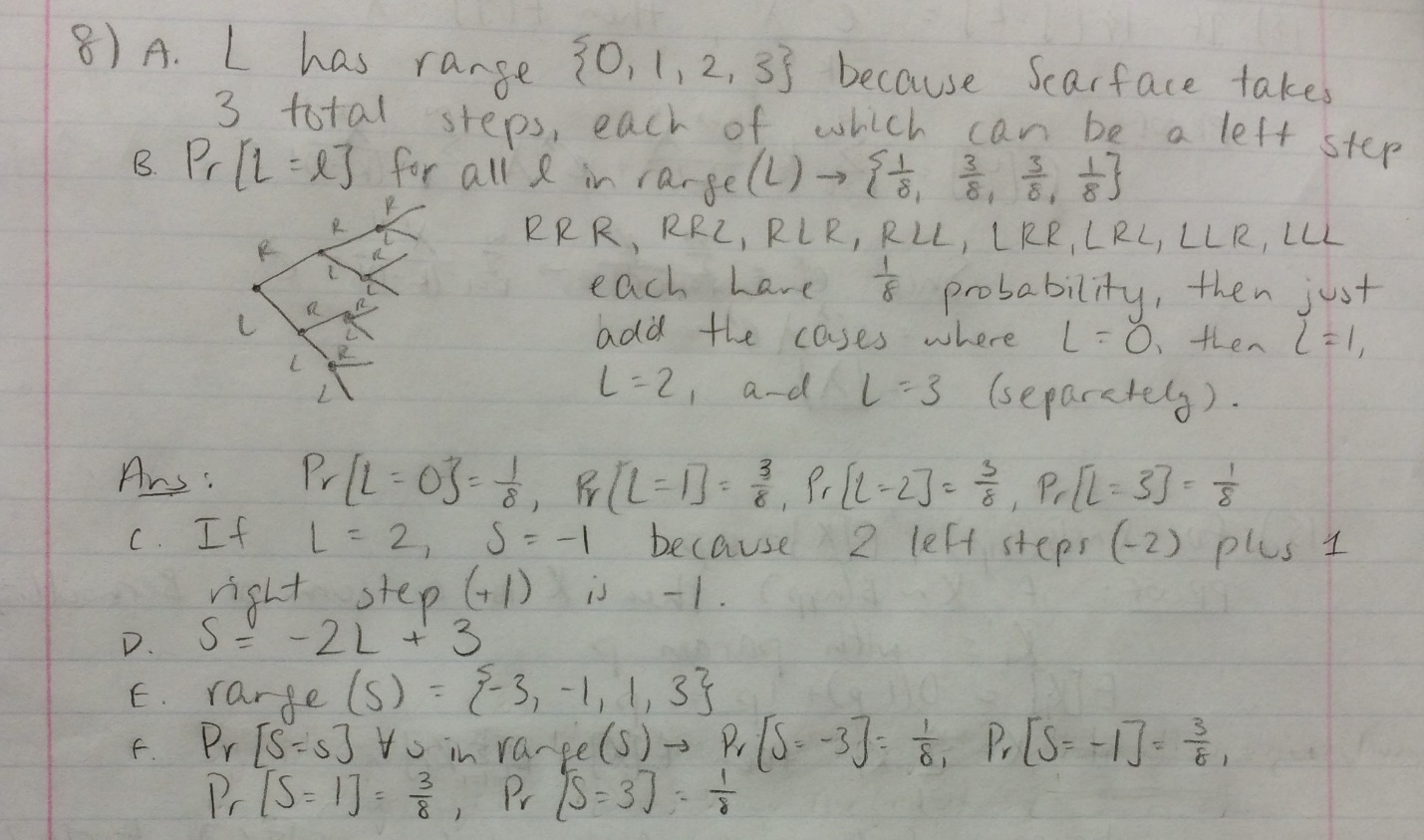
plt.xlim(0, max(y\_coords))

plt.title("1000 Binomial trials")

plt.xlabel("Number of Successes")

plt.ylabel("Probability")

plt.show()

8) 

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')

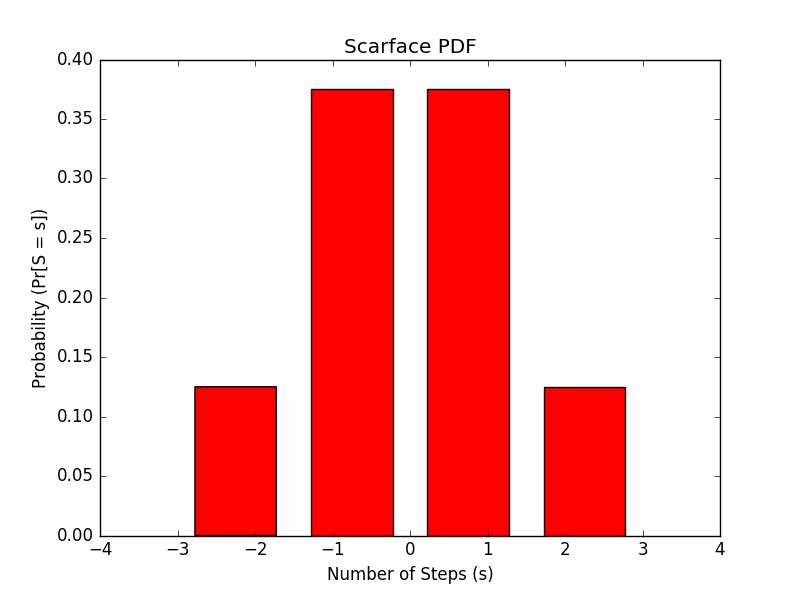
plt.xlim(-(steps+1), (steps+1))

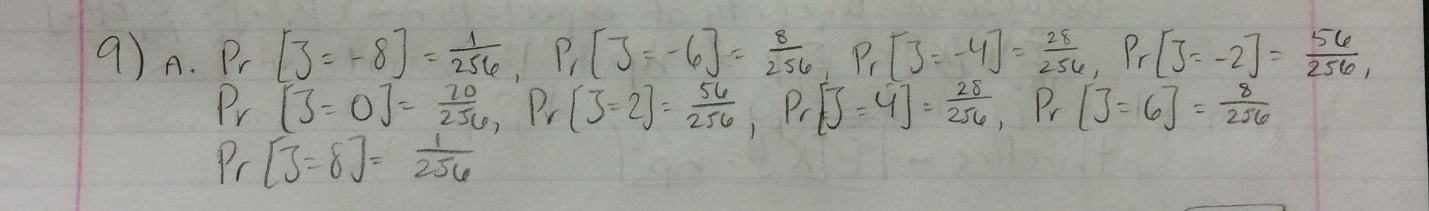
plt.title("Scarface PDF")

plt.xlabel("Number of Steps (s)")

plt.ylabel("Probability (Pr[S = s])")

plt.show()



9) 

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')

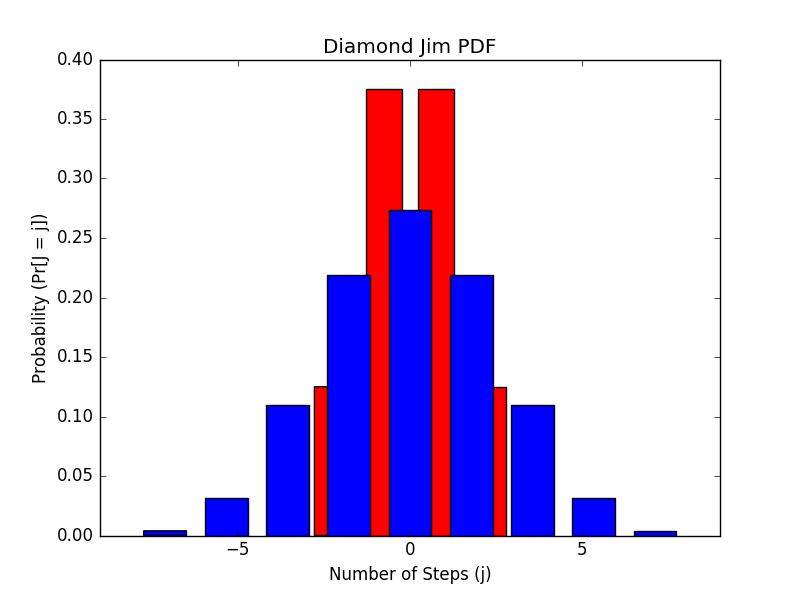
plt.xlim(-(steps+1), (steps+1))

plt.title("Diamond Jim PDF")

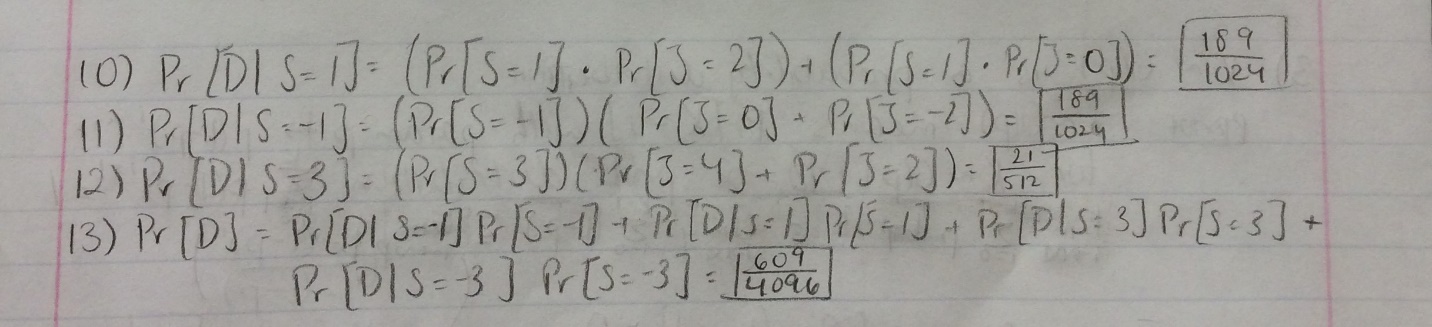
plt.xlabel("Number of Steps (j)")

plt.ylabel("Probability (Pr[J = j])")

plt.show()



10) – 13)



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))

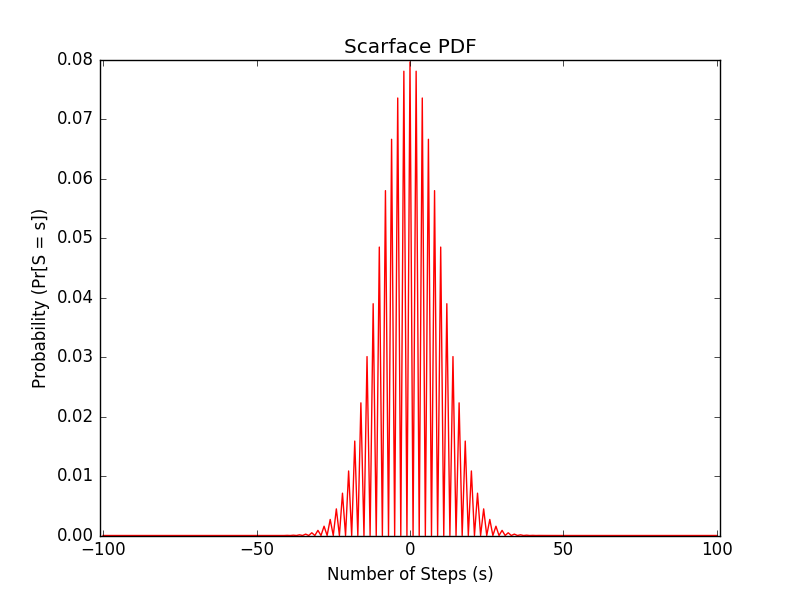
plt.title("Diamond Jim PDF")

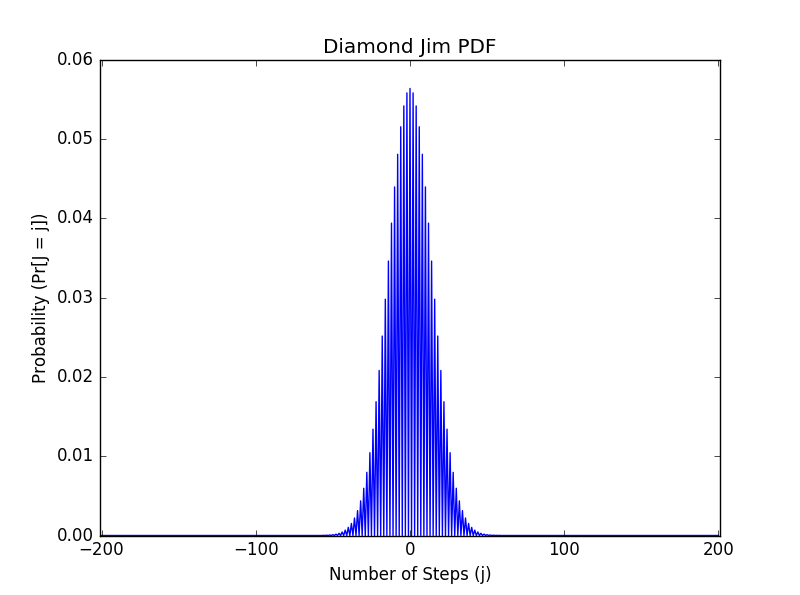
plt.xlabel("Number of Steps (j)")

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])

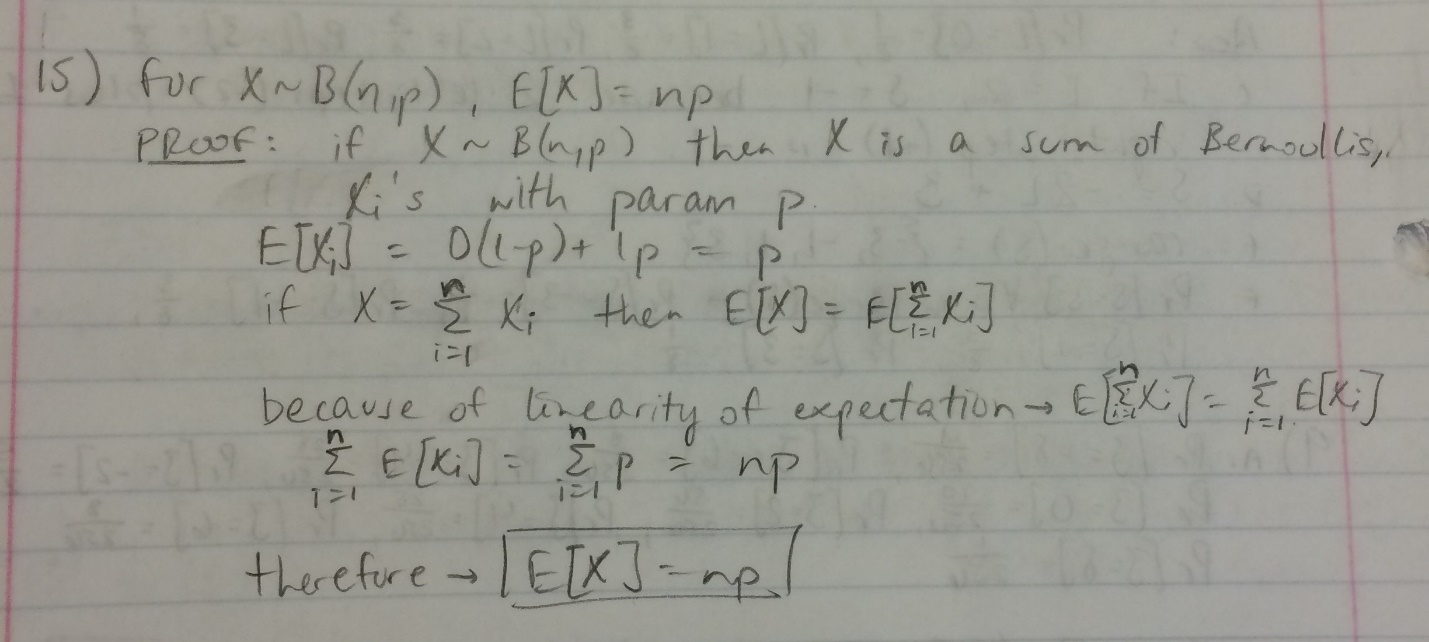
temp += d(distJ[stepsJ + total\_steps+1])

temp \*= (d(nCr(stepsS, i))/d(2\*\*stepsS))\*\*2

d\_prob += temp

print d\_prob

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

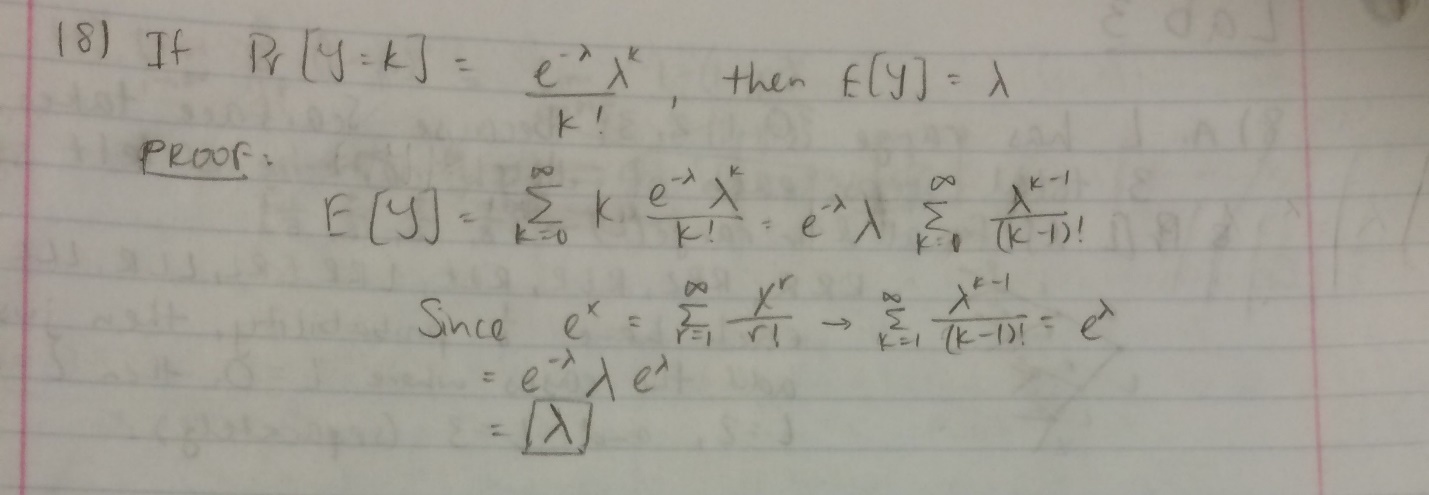
15) 

Part 5 – Bitcoin

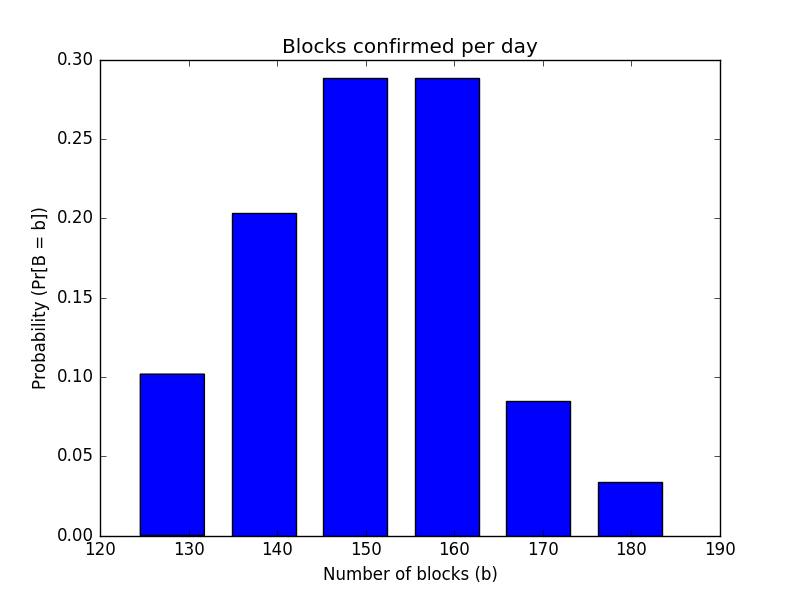
16) The probability of getting 71 leading zeros is (½)^(71) because it is the same as the probability of flipping a coin 71 times and getting all tails.

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)



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',\

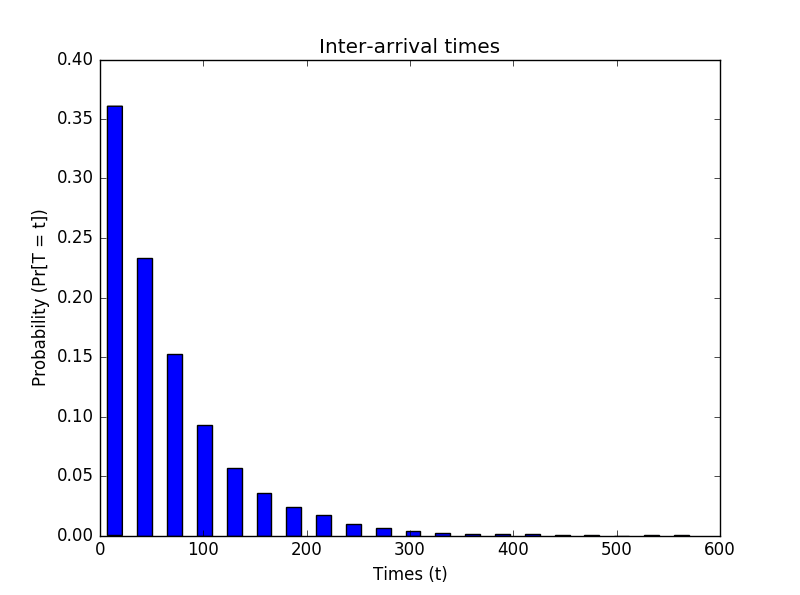
weights = np.zeros\_like(v2) + 1. / len(v2), color = 'b')

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()

