**EE 511**

**PROJECT # 2**

BY

PRANAV GUNDEWAR

USC ID: 4463612994

EMAIL: [gundewar@usc.edu](mailto:gundewar@usc.edu)

**Problem 1 Random Number Generation - Networking**

**Summary-**

* This is extension of Networking problem from last assignment. For given network sampled and probability, we have to plot network graph and are supposed to observe the structure of obtained graph.
* I have used “NetworkX” library on Python to plot graph. NetworkX includes many graph generator functions and facilities to read and write graphs in many formats.
* Degree of vertex is number of connections each node or vertex has on sample graphs. Histogram of degree of vertex supports binomial distribution as it has sum of IID samples.
* This exercise allows to play with network graph and its distribution and how network flow works in systems.

**Result and Analysis-**

* Histogram for all the given probabilities has been plotted and observed. Network graph follows CLT theorem and hence has binomial distribution.
* For every network graph, as probability increases, number of edges increases and it appears densely crowded

**Code-**

# -\*- coding: utf-8 -\*-

"""

@author: Pranav Gundewar

Project 2

Q1- Networking Part 2

Routine to plot network graph and histogram of degree vertex

"""

# Importing Libraries

import numpy as np

import random as rand

import matplotlib.pyplot as plt

import networkx as nx

# Initializing Variables

n = 50 # Nodes

N = (int) (n\*(n-1)/2)

G1 = nx.Graph() # initialize the graphs

G2 = nx.Graph()

G3 = nx.Graph()

d = np.zeros([n]) # Array for

#H = nx.path\_graph(n)

#G1.add\_nodes\_from(H)

for i in range(n-1):

for j in range(i+1,n):

a = rand.uniform(0,1) ;

if (a < 0.02):

G1.add\_edge(i,j) # Add edge if it exits

else :

G1.add\_node(i)

G1.add\_node(j)

if (a < 0.09):

G2.add\_edge(i,j)

else :

G2.add\_node(i)

G2.add\_node(j)

if (a < 0.12):

G3.add\_edge(i,j)

else :

G3.add\_node(i)

G3.add\_node(j)

options = { 'node\_color': 'red','node\_size': 100,'width': 3,}

nx.draw\_circular(G1, with\_labels=True, \*\*options)

plt.title('Network Graph for n = 50, p = 0.02')

plt.show()

nx.draw\_circular(G2, with\_labels=True, \*\*options)

plt.title('Network Graph for n = 50, p = 0.09')

plt.show()

nx.draw\_circular(G3, with\_labels=True, \*\*options)

plt.title('Network Graph for n = 50, p = 0.12')

plt.show()

for i in range(n):

d[i] = G1.degree[i] # Calculate Degree of Vertex

plt.hist(d, bins = 'auto', facecolor='green') # Plotting histogram

plt.xlabel('Vertex Degree')

plt.ylabel('Number of Connections')

plt.title('n = 50, p = 0.12')

plt.grid(True)

plt.legend(n = 50, p = 0.02)

#plt.savefig('Q3.jpeg')

plt.show()

n = 100

N = (int) (n\*(n-1)/2)

G4 = nx.Graph()

d = np.zeros([n])

#H = nx.path\_graph(n)

#G1.add\_nodes\_from(H)

for i in range(n-1):

for j in range(i+1,n):

a = rand.uniform(0,1) ;

if (a < 0.06):

G4.add\_edge(i,j)

else :

G4.add\_node(i)

G4.add\_node(j)

options = { 'node\_color': 'red','node\_size': 100,'width': 3,}

nx.draw\_circular(G4, with\_labels=True, \*\*options)

plt.title('Network Graph for n = 100, p = 0.06')

plt.show()

for i in range(n):

d[i] = G4.degree[i]

plt.hist(d, bins = 'auto', facecolor='green')

plt.xlabel('Vertex Degree')

plt.ylabel('Number of Connections')

plt.title('n = 100, p = 0.06')

plt.grid(True)

plt.legend(n = 100, p = 0.06)

#plt.savefig('Q3.jpeg')

plt.show()

**Output:**

Network Graphs:

|  |  |
| --- | --- |
| C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\q1a0.02.png | C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\q1a0.09.png |
| C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\q1a0.12.png | C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\q1a0.06.png |

**Histogram:**

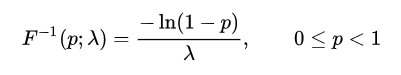
|  |  |
| --- | --- |
| C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\hist0.02.png | C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\hist0.09.png |
|  |  |
| C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\hist0.12.png | C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\n100p0.06.png |

* Hence histogram assumes binomial distribution as it follows Central Limit Theorem (CLT) hence it is collection of random IID samples which follows normal distribution.
* Histogram displays out of available nodes; how many nodes has given degree of vertex and is plotted against number of nodes. It follows binomial distribution

**Problem 2 Waiting Time - Goodness of Fit**

**Summary-**

* Exponential function of waiting time of 0.2 units is given. We are supposed to generate exponential PDF using inverse CDF and compare it with computer generated RNGs to quantify performance of our generator with various tests.
* Inverse CDF is given by



* The quartiles are given by - first quartile: ln(4/3)/λ

[median](https://en.wikipedia.org/wiki/Median): ln(2)/λ

third quartile: ln(4)/λ

* Chi-Square Goodness of fit: The test is applied when you have one [categorical variable](http://stattrek.com/Help/Glossary.aspx?Target=Categorical%20variable) from a single population. It is used to determine whether sample data are consistent with a hypothesized distribution.
* Every hypothesis test requires the analyst to state a [null hypothesis](http://stattrek.com/Help/Glossary.aspx?Target=Null%20hypothesis) (H0) and an [alternative hypothesis](http://stattrek.com/Help/Glossary.aspx?Target=Alternative%20hypothesis) (Ha). The hypotheses are stated in such a way that they are mutually exclusive. That is, if one is true, the other must be false; and vice versa.
* For a chi-square goodness of fit test, the hypotheses take the following form.

H0: The data are consistent with a specified distribution.

Ha: The data are *not* consistent with a specified distribution

* In second part, we have to code a routine to count number of exponential distributed intervals in each unit time. In simpler words, we have to check arrival time for each unit samples. Hence distribution will be Poisson as it follows arrival time.

**Result and Analysis-**

* The RNGs generated by inverse CDF method is compared with exponential CDF defined in scipy library using Chi-Square goodness of fit.
* While plotting histogram, plot histogram function in mathplotlib has several settings to decide number of bins in histogram.

**‘auto’ -** Maximum of the ‘sturges’ and ‘fd’ estimators. Provides good all-around performance.

**‘fd’ (Freedman Diaconis Estimator) -** Robust (resilient to outliers) estimator that considers data variability and data size.

**‘doane’ -** An improved version of Sturges’ estimator that works better with non-normal datasets.

**‘scott’-** Less robust estimator that that takes into account data variability and data size.

**‘rice’-** Estimator does not take variability into account, only data size. Commonly overestimates number of bins required.

**‘sturges’-** R’s default method, only accounts for data size. Only optimal for gaussian data and underestimates number of bins for large non-gaussian datasets.

* On trial and error basis, I found out sturges option performs better. It gives optimal result for normal distribution helps in deciding bin sizes.

Power\_divergenceResult = (statistic = 10.290621774943048,

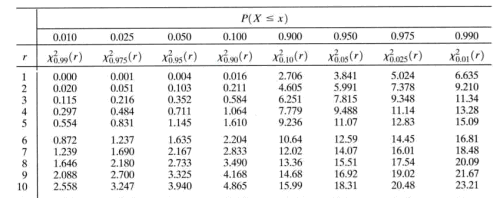
pvalue = 0.41537674668092484)

Bins for Inverse CDF: [ 427. 224. 152. 81. 44. 34. 19. 4. 7. 6. 2.]

Bins for Random Exponential: [ 416. 249. 141. 82. 45. 25. 21. 8. 6. 6. 1.]

I get best output for 11 bins and chi-square value is found out to be 10.29.

From Chi-Square table, I found out the value for 10 degrees of freedom for 95% confidence which is 18.35.



Chi-Square Table

Obtained value 1.14 < Table Value 18.31. Hence RNG performs exceptionally well for given confidence interval. (Generally, 95% confidence interval is assumed.)

* Chi-Square goodness of fit formula: Χ2 = Σ [ (Oi - Ei)2 / Ei ]

Oi is the observed frequency count for the *i*th level of the categorical variable, and Ei is the expected frequency count for the *i*th level of the categorical variable

* For second part, we have to count waiting time until event occurs. Ultimately, we are counting arrival time which follows Poisson arrival. The count for each unit interval is distributed in curve represented by Poisson curve. Here, counter is used to count until addition of PDF values is lesser than equal to one.
* As number of intervals increases, binomial converges Poisson. One main reason being it follows Central Limit Theorem hence it has Gaussian(Poisson) curve. It is collection of random IID samples.

**Code:**

# -\*- coding: utf-8 -\*-

"""

@author: Pranav Gundewar

Project 2

Q2- Waiting time exponential distribution

Routine to Chi-Square GOF and distribution of count

"""

# Importing Libraries

import numpy as np

import random as rand

import matplotlib.pyplot as plt

from scipy.stats import chisquare

from scipy.stats import expon

# Initializing Variables and definitions

def expon\_pdf(x, lmabd=5):

"""PDF of exponential distribution."""

return lmabd\*np.exp(-lmabd\*x)

def expon\_cdf(x, lambd=5):

"""CDF of exponetial distribution."""

return 1 - np.exp(-lambd\*x)

def expon\_icdf(u, lambd=5):

"""Inverse CDF of exponential distribution - i.e. quantile function."""

return -np.log(1-u)/lambd

# Finding Inverse CDF and random generated PDF

u = np.random.random(1000)

v = np.zeros(1000)

c = np.zeros(1000)

for i in range (1000):

u = rand.uniform(0,1)

v[i] = expon\_icdf(u)

r = expon.rvs(size=1000,scale = 0.2)

# Finding Count

for i in range (1000):

b = 0

cntr = 0

while (b <= 1):

u = rand.uniform(0,1)

a = expon\_icdf(u)

b = b + a

cntr = cntr + 1

c[i] = cntr

plt.figure(figsize = (12,4))

plt.hist(c, bins = 'auto', linewidth=2, facecolor='green')

plt.xlabel('Arrival Time Count')

plt.ylabel('Occurance')

plt.title('Count of Exponentially distributed time interval of 1 unit time');

# Plotting Histogram

plt.figure(figsize = (12,4))

plt.subplot(121)

plt.title('Histogram of exponential PDF using inverse method');

(x1, y1, z1) = plt.hist(v, bins = 'sturges', linewidth=2, facecolor='green')

plt.subplot(122)

plt.title('Histogram of exponential PRNGs');

(x2, y2, z2) = plt.hist(r, bins = 'sturges', linewidth=2, facecolor='blue')

l = chisquare(x1,x2)

print (l)

print(x1)

print(x2)

#Power\_divergenceResult(statistic=10.290621774943048, pvalue=0.41537674668092484)

#[ 427. 224. 152. 81. 44. 34. 19. 4. 7. 6. 2.]

#[ 416. 249. 141. 82. 45. 25. 21. 8. 6. 6. 1.]

# Plotting Histogram

plt.figure(figsize = (12,4))

plt.subplot(121)

plt.title('Histogram of exponential PDF using inverse method');

binsn = [0.04, 0.1, 0.18, 0.32]

#(x1, y1, z1) = plt.hist(v, bins = 'sturges', linewidth=2, facecolor='green')

(x1, y1, z1) = plt.hist(v, binsn, linewidth=2, facecolor='green')

plt.subplot(122)

plt.title('Histogram of exponential PRNGs');

#(x2, y2, z2) = plt.hist(r, bins = 'sturges', linewidth=2, facecolor='blue')

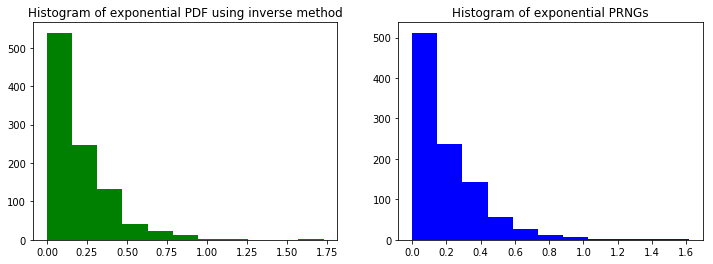
(x2, y2, z2) = plt.hist(r, binsn, linewidth=2, facecolor='blue')

l = chisquare(x1,x2)

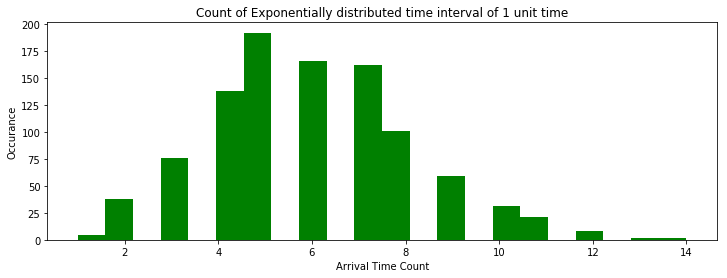
print (l)

Power\_divergenceResult(statistic=1.1149688558297048, pvalue=0.5726477902759346)

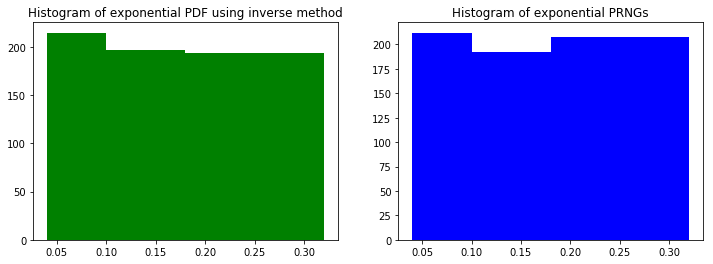
**Output:**



Histogram using both methods to generate Exponential PDF



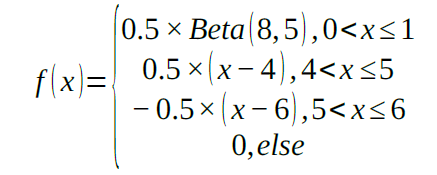
Gaussian distribution for counts



**Problem 3 Double Rejection - Goodness of Fit**

**Summary –**

* Random variable has bimodal distribution given by



Which is equally weighted, convex summation of beta and triangle pdf.

* For 1 envelop, we are supposed to code a routine for 1000 accepted samples. Accept-reject method is used here. It follows given algorithm:
* The requirement in rejection sampling is that there exists unnormalized density g and a constant M such that for all states

x ∈ Ω, h(x) ≤ Mg(x).

We call g the (unnormalized) instrumental density. The idea is that g is chosen such that it is easy to sample from g and h(x)/(Mg(x)) can be considered as a probability with which we accept a draw from g:

Accept/rejection algorithm:

1. Generate X ∼ g and U ∼ uniform(0,1).

2. Accept Y = X if U ≤ h(X)/(Mg(X)), else repeat 1.–2

* The rejection rate is the average number of rejected candidates per sample. This is a measure of the efficiency of your RNG. Here, we should find out how many samples were rejected along the process to obtain 1000 accepted samples.
* For given PDF, maximum value is found out to be 1.46 at x = 0.66 using beta PDF. Hence one random generator would be from 0 to 6 and another one would be from 0 to 0.5.

**Results and Conclusion:**

* We have find the routine to count the accepted samples until it reaches 1000. After it reaches the required count, we have to find rejection ratio which is ratio of rejected samples to the all the samples under consideration.
* The measure of efficiency should be as high as possible which means rejection ratio should be as low as possible.
* Rejection Rate: 88.30272546496666
* The rejection rate can be improves using better algorithm than accept reject or by using different PDF.
* The motive behind this problem is to get used to methods to quantify your results and improve the algorithm to obtain desired efficiency.
* Rejection rate can be improved using different value of c i.e. changing the distribution most likely.

**Code:**

# -\*- coding: utf-8 -\*-

"""

@author: Pranav Gundewar

Project 2

Q3- Double rejection

Convex summation of beta and triangle distribution

"""

# Importing Libraries

import numpy as np

import random

import math

import matplotlib.pyplot as plt

ar = np.zeros(1000)

at = 0

rt = 0

def beta\_pdf(x, a, b):

"""PDF of beta distribution."""

return math.gamma(a+b)/(math.gamma(a)\*math.gamma(b))\*x\*\*(a-1)\*(1-x)\*\*(b-1)

for i in range(100000):

while (at < 1000):

a = 6\*random.uniform(0,1)

if (a > 0 and a <= 1):

ar[i] = 0.5\*beta\_pdf(a,8,5)

elif (a > 4 and a <= 5):

ar[i] = 0.5\*(a - 4)

elif (a > 5 and a <= 6):

ar[i] = -0.5\*(a - 6)

else:

ar[i] = 0

if (ar[i] >= 1.46\*random.uniform(0,1)):

at = at + 1

else:

rt = rt + 1

plt.hist(ar, bins = 'auto')

k = rt / (rt + at) \* 100

print('Rejection Rate: \n')

print(k)

**Output:**

Rejection Rate: 88.30272546496666

Rejection rate is very high. RNGs is working not so efficiently. We need to change c value.