STA6106- Homework 2 -By Anmol Sureshkumar Panchal UID:4446829

Problem 1:

```
##Import these packages:

import math

import pandas

import numpy as np

import scipy.stats

from scipy.stats import uniform

from scipy.stats import binom

import math

from scipy.stats import norm

import matplotlib

import matplotlib.pyplot as plt
```

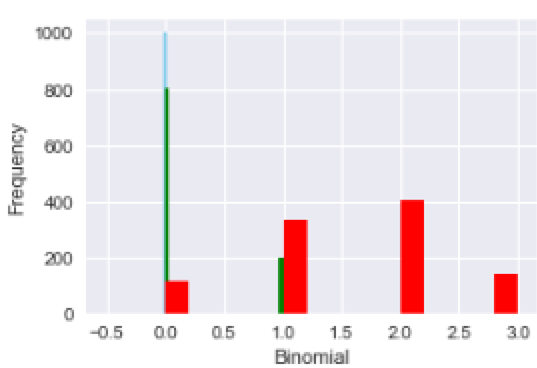
Python Code:

```
##########################################Problem 1###########################################

x = binom.rvs(0, 0.3, size=1000)  // generates random variates from a binomial distribution with X=0, pmf =  0.3

y = binom.rvs(1, 0.2, size=1000)  //// generates random variates from a binomial distribution with X=1, pmf =  0.2

z = binom.rvs(3, 0.5, size=1000)  //// generates random variates from a binomial distribution with X=3, pmf =  0.5

print(x)

print(y)

print(z)

##########################################Problem 1###########################################

# for inline plots in jupyter

%matplotlib inline

# import matplotlib

import matplotlib.pyplot as plt

# import seaborn

import seaborn as sns

# settings for seaborn plotting style

sns.set(color_codes=True)

# settings for seaborn plot sizes

sns.set(rc={'figure.figsize':(4.5,3)})

data_binom_0 = binom.rvs(n=0,p=0.3, size=1000)

print(data_binom_0)
```

```
ax = sns.distplot(data_binom_0,
                  kde=False,
                  color='skyblue',
                  hist_kws={"linewidth": 15,'alpha':1})
ax.set(xlabel='Binomial', ylabel='Frequency')
data_binom_1 = binom.rvs(n=1,p=0.2, size=1000)
print(data_binom_1)
ax = sns.distplot(data_binom_1,
                  kde=False,
                  color='green',
                  hist_kws={"linewidth": 15,'alpha':1})
ax.set(xlabel='Binomial', ylabel='Frequency')
data_binom_3 = binom.rvs(n=3,p=0.5, size=1000)
print(data_binom_3)
ax = sns.distplot(data_binom_3,
                  kde=False,
                  color='red',
                  hist_kws={"linewidth": 15,'alpha':1})
ax.set(xlabel='Binomial', ylabel='Frequency')
############################################Problem 1############################################
```

Output:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 ………………. . 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 ………… . . 1 0 0 0 0 0 0
0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0]

[2 0 0 1 3 1 2 1 3 2 1 1 2 3 2 2 2 0 0 3 2 2 ………… . . 1 0 2 2 2 3 1 2 1 2 0 0 2
3 3 3 2 2 2 3 1 2 2 1 2 1 1 1 1 1 1 3 2 2 1 2 2 2 3 2 1 2 2 2 1 2 0 2 1 2 1 1
1 2 1 1 2 1 1 1 2 1 2 3 2 1 2 1 1 1 1 1 2 0 2 0 1 3 3 1 3 2 1 2 0 2 1 1 2 2 2
1 1 2 1 1 3 3 0 1 1 2 1 2 1 2 1 0 1 2 0 3 2 3 1 2 1 2 2 1 1 3 1 2 3 2 1 1 2
3]
```



Problem 2:

The Box-Muller transform is a method for generating normally distributed random numbers from uniformly distributed random numbers. The Box-Muller transformation can be summarized as follows, suppose u1 and u2 are independent random variables that are uniformly distributed between 0 and 1 and let then z1 and z2 are independent random variables with a standard normal distribution. Intuitively, the transformation maps each circle of points around the origin to another circle of points around the origin where larger outer circles are mapped to closely-spaced inner circles and inner circles to outer circles.

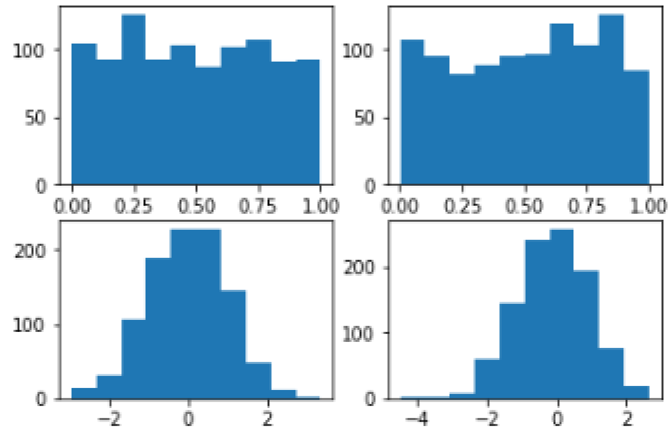
Python Code:

```
##########################################Problem 2##########################################

def box_muller():

    u1 = random.random()

    u2 = random.random()

    t = math.sqrt((-2) * math.log(u1))

    v = 2 * math.pi * u2

    return t * math.cos(v), t * math.sin(v)

from numpy import random, sqrt, log, sin, cos, pi

from pylab import show,hist,subplot,figure


# transformation function

def gaussian(u1,u2):

  z1 = sqrt(-2*log(u1))*cos(2*pi*u2)

  z2 = sqrt(-2*log(u1))*sin(2*pi*u2)

  return z1,z2


# uniformly distributed values between 0 and 1

u1 = random.rand(1000)

u2 = random.rand(1000)


# run the transformation

z1,z2 = gaussian(u1,u2)


# plotting the values before and after the transformation

figure()

subplot(221) # the first row of graphs

hist(u1)     # contains the histograms of u1 and u2

subplot(222)

hist(u2)
```

```
subplot(223) # the second contains
hist(z1)     # the histograms of z1 and z2
subplot(224)
hist(z2)
show()
########################################Problem 2########################################
##Random number generation with Box-Muller algorithm
import math
import random
import sys
import traceback
class RndnumBoxMuller:
    M     = 10        # Average
    S     = 2.5       # Standard deviation
    N     = 10000     # Number to generate
    SCALE = N // 100  # Scale for histogram

    def __init__(self):
        self.hist = [0 for _ in range(self.M * 5)]

    def generate_rndnum(self):
        ##Generation of random nos.
        try:
            for _ in range(self.N):
                res = self.__rnd()
                self.hist[res[0]] += 1
                self.hist[res[1]] += 1
        except Exception as e:
            raise
    def display(self):
       ##showing
        try:
            for i in range(0, self.M * 2 + 1):
                print("{:>3}:{:>4} | ".format(i, self.hist[i]), end="")
                for j in range(1, self.hist[i] // self.SCALE + 1):
                    print("*", end="")
```

```
                print()
        except Exception as e:
            raise
    def __rnd(self):
   ##random integers generation.
        try:
            r_1 = random.random()
            r_2 = random.random()
            x = self.S \
              * math.sqrt(-2 * math.log(r_1)) \
              * math.cos(2 * math.pi * r_2) \
              + self.M
            y = self.S \
              * math.sqrt(-2 * math.log(r_1)) \
              * math.sin(2 * math.pi * r_2) \
              + self.M
            return [math.floor(x), math.floor(y)]
        except Exception as e:
            raise
if __name__ == '__main__':
    try:
        obj = RndnumBoxMuller()
        obj.generate_rndnum()
        obj.display()
    except Exception as e:
        traceback.print_exc()
        sys.exit(1)
##########################################Problem 2###########################################
```

Output:




```
0: 1 | 1: 6 | 2: 35 | 3: 124 | * 4: 304 | *** 5: 675 | ****** 6:1235 |
************ 7:1860 | ****************** 8:2657 |
************************** 9:3162 | *******************************
10:3122 | ******************************* 11:2646 |
************************** 12:1914 | ******************* 13:1169 |
*********** 14: 642 | ****** 15: 288 | ** 16: 107 | * 17: 39 | 18: 10 |
19: 3 | 20: 1 |
```

In the first row of the graph we can see, respectively, the histograms of u1 and u2 before the transformation and in the second row we can see the values after the transformation, respectively z1 and z2. We can observe that the values before the transformation are distributed uniformly while the histograms of the values after the transformation have the typical Gaussian shape.

Problem 3:

To find an explicit formula for $F^{-1}(r)$ for the c.d.f. of a r.v or to generate $F(x) = P(X \leq (x))$ is not always possible. Even if we can, that may not be the most efficient method for generating a r.v. distributed according to F. Let us assume the continuous case and that X has c.d.f. F and p.d.f. f. I'll give the discrete case later, which is very similar. The basic idea is to find an alternative probability distribution G, with density g(x), from which we can easily simulate (e.g., inverse-transform etc.). However, we'll also want g(x) 'close' to f(x). • So we assume that f(x)/g(x) is bounded by a constant c > 0. Hence $\sup_x\{f(x)/g(x)\} \leq c$. In practice we want c close to 1 as possible. So Accept-Reject Algorithm can be shown as below:

1. Generate a r.v. Y according to G (remember we assumed this was easy).
2. Generate $R \sim U(0, 1)$ independent of Y .
3. If $U \leq f(Y)/cg(Y)$, then set X = Y , i.e., 'accept'; otherwise start again i.e., 'reject'.

The Laplace distribution is similar to the Gaussian/normal distribution, but is sharper at the peak and has fatter tails. It represents the difference between two independent, identically distributed exponential random variables. So we use numpy.random.laplace to draw out the random variables.

Python Code:

```
###########################################Problem 3###########################################

    import numpy as np
    from scipy.stats import binom
    import seaborn as sns
    def rlaplace(n, mu, sigma):
       U = np.random.uniform(0,1,n)
```

```
sign = binom.rvs(1, 0.5, size = n)
sign[sign>0.5] = 1
sign[sign<0.5] = -1
y = mu + sign*sigma/np.sqrt(2)*np.log(1-U)
print(y)
sns.distplot(y)
```

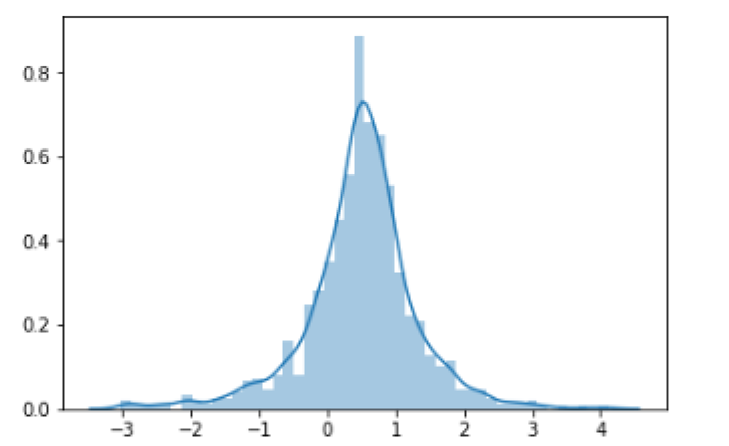
##Problem 3##

```
rlaplace(1000,0.5,0.8)
```

##Problem 3##

Output:

```
[ 1.97957488e+00 9.11064488e-01 -7.47118832e-02 -1.11035006e-01
1.75052280e-01 1.28033891e+00 3.54853738e-01 1.25000945e+00 7.33231156e-01
7.30976973e-01 9.66269390e-02 1.11508146e+00 1.62765440e-01 3.78388291e-01
4.65458059e-02 -8.69660965e-01 4.31252719e-01 1.21465695e+00 4.81954685e-
01 8.04330333e-01 1.84063739e+00 9.09176418e-01 -8.48805240e-02
6.81097944e-02 4.97880519e-01 6.07943130e-01 9.75030687e-01 7.98664702e-02
2.11572906e+00 -3.02104963e-01 5.68779354e-01 6.61962489e-01 8.58086541e-
01 -2.37536623e-02 3.98677561e-01 4.08136371e-01 -2.01465910e-01
3.20109031e-01 -1.27358877e+00 4.86182050e-01 6.79726582e-01 -
2.04511567e+00 -2.59261110e-01 9.84961303e-01 4.62006691e-01 3.45308444e-
01 7.86371303e-01 9.02604451e-01 1.19131000e-01 6.07052726e-01
4.47364199e-01 1.31275986e+00 5.03099869e-01 -2.05769767e-01
1.00345963e+00 -7.50119451e-01 9.90887682e-01 -2.10246610e+00 6.96781387e-
02 4.74202582e-01 2.95653769e-01 4.75301314e-01 9.93965187e-02 -
1.24026995e+00……...]
```

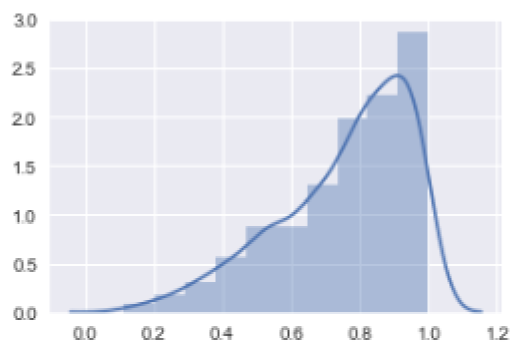


Problem 4:

Python Code:

```
###########################################Problem 4############################################
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
n = 1000
u = np.random.uniform(low = 0.0, high = 1.0, size = n)
x = u**(1/3)
sns.distplot(x, hist = True, kde=True, bins = 10)
plt.show()
###########################################Problem 4############################################
```

Output:



Problem 5:

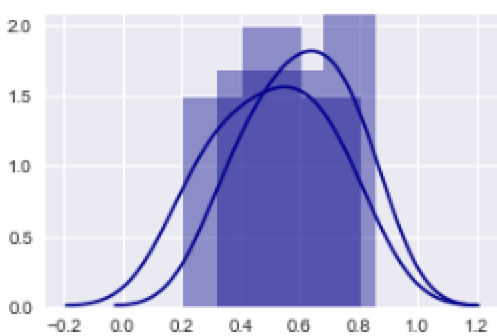
Python Code:

```
###########################################Problem 5############################################
import pandas as pd
from scipy.stats import beta
from scipy.stats.mstats import mquantiles
n = 1000
k = 0
j = 0
y = np.zeros(n)
while k<n-1:
    u = np.random.uniform(size = 1)
```

```
    j = j+1
    x = np.random.uniform(size = 1)
    if x*(1-x) > u:
        k = k+1
        y[k] = x
p = np.linspace(start = 0.1, stop = 0.9, num=10)
q_hat =  scipy.stats.mstats.mquantiles(y, p)
##q_hat = np.quantile(y,p)
r = beta.ppf(p, 3, 2)
z = np.sqrt(p*(1-p)/(n*beta.pdf(r, 3, 2)**2))
print(q_hat)
print(r)
print(z)
temp = np.array([q_hat, r])
for i in temp:
    sns.distplot(i, hist = True, kde = True, color = 'darkblue')


plt.show()
##########################################Problem 5##########################################
```

Output:

```
[0.20360727 0.28521864 0.34565156 0.41911701 0.4965357  0.54697297
 0.60732269 0.66017803 0.72476103 0.80970146]
[0.32046058 0.40829619 0.47627164 0.53494954 0.58857456 0.63953933
 0.68961098 0.7405407  0.7947675  0.85744068]
[0.00113286 0.0010457  0.00099354 0.00095421 0.00092081 0.00089017
 0.00086031 0.00082954 0.00079568 0.00075429]
```



Problem 6:

Python Code:

```
##########################################Problem 6###########################################
import numpy as np
import math
import scipy.stats
import seaborn as sns
import collections as col

n = 1000
theta = 0.5
u = np.random.uniform(size = n)
v = np.random.uniform(size = n)
x = (1+np.log(v)/np.log(1-(1-theta)**u))
x = np.floor(x)

k = []
for i in range(1, int(max(x))+1):
    k.append(i)

p = []
for j in range(0, len(k)):
    temp = -1/np.log(1-theta)*(theta*k[j])/k[j]
    p.append(temp)

se = []
for i in range(0, len(p)):
    temp = np.sqrt(p[i]*(1-p[i])/n)
    se.append(temp)

c = col.Counter(x).values()
c = list(c)
p_hat = []
for i in range(0, len(c)):
    temp = c[i]/n
    p_hat.append(temp)
```

print("P_hat: ", p_hat)

print("\nP: ", p)

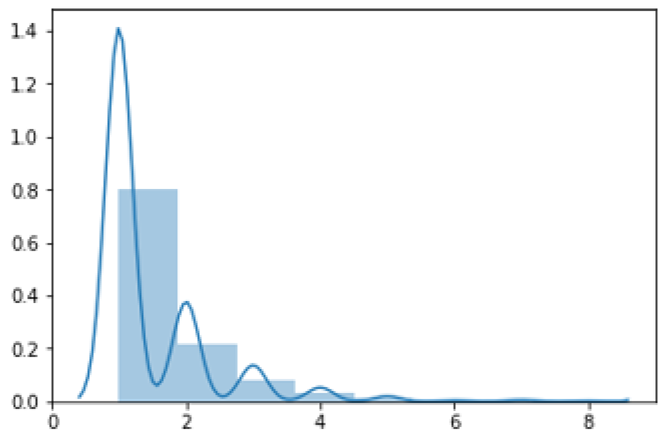
print("\nse: ", se)

sns.distplot(x, hist = True, kde = True, bins = 8)

##Problem 6##

Output:

```
P_hat:  [0.702, 0.188, 0.01, 0.068, 0.026, 0.003, 0.002, 0.001]

P:  [0.7213475204444817, 0.7213475204444817, 0.7213475204444816,
7213475204444817, 0.7213475204444817]

se:  [0.014177632919252768, 0.014177632919252768, 0.014177632919
1925277, 0.014177632919252768, 0.014177632919252768]

C:\Users\anmol\Anaconda3\lib\site-packages\matplotlib\axes\_axes
s been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been

.]: <matplotlib.axes._subplots.AxesSubplot at 0x24d0372f6d8>
```



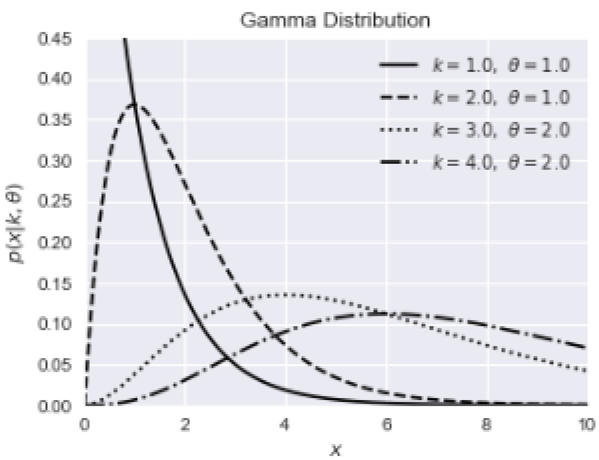
Problem 7:

Python Code:

```
########################################Problem 7########################################
import numpy as np
from scipy.stats import gamma
from matplotlib import pyplot as plt
#------------------------------------------------------------
# plot the distributions
k_values = [1, 2, 3, 4]
theta_values = [1, 1, 2, 2]
linestyles = ['-', '--', ':', '-.']
x = np.linspace(1E-6, 10, 1000)

#------------------------------------------------------------
# plot the distributions
fig, ax = plt.subplots(figsize=(5, 3.75))

for k, t, ls in zip(k_values, theta_values, linestyles):
    dist = gamma(k, 0, t)
    plt.plot(x, dist.pdf(x), ls=ls, c='black',
             label=r'$k=%.1f,\ \theta=%.1f$' % (k, t))

plt.xlim(0, 10)
plt.ylim(0, 0.45)

plt.xlabel('$x$')
plt.ylabel(r'$p(x|k,\theta)$')
plt.title('Gamma Distribution')

plt.legend(loc=0)
plt.show()
###from scipy.stats import gamma
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0,10,.1)
```

```
y1 = np.random.gamma(shape=4, scale=3, size=1000) + 2  # sets loc = 2

y2 = np.hstack((y1, 10*np.random.rand(100)))  # add noise from 0 to 10


# fit the distributions, get the PDF distribution using the parameters

shape1, loc1, scale1 = gamma.fit(y1)

g1 = gamma.pdf(x=x, a=shape1, loc=loc1, scale=scale1)

g1

hist(g1)

import seaborn as sns

sns.distplot(g1)

###########################################Problem 7###########################################
```

Output:




```
<matplotlib.axes._subplots.AxesSubplot at 0x24d03815438>
```