ENEL 649 Project

Name: Deep Vyas

UCID: 30139014

Course: Random Variables and Stochastic Processes

```
# Project Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import scipy.stats
from scipy.integrate import quad

class color:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'
```

Common Distributions

1. Exponential Distribution

2. Normal Distribution

3. Custom Exponential Distribution

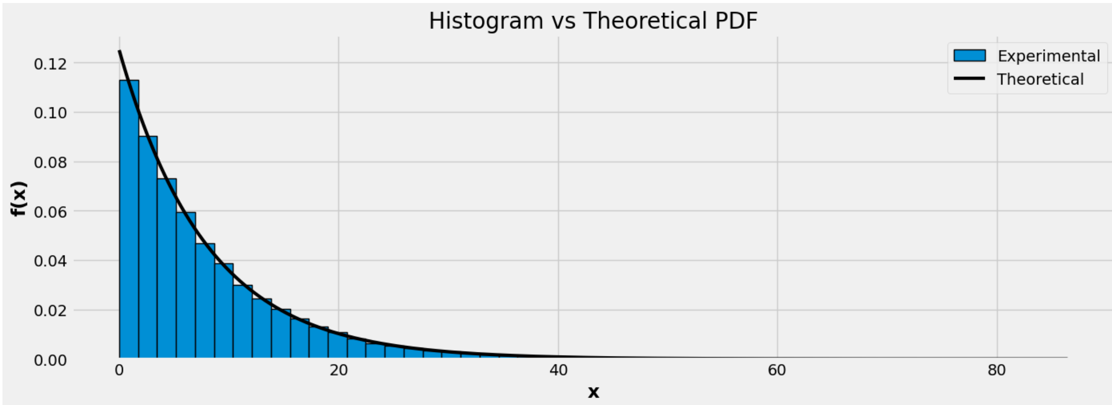
```
# 1 Exponential Distribution
def exp_f(x, lam):
    return lam*np.exp(-lam*x)

# 2 Normal Distribution
def normal_dist(x , mean , sd):
    prob_density = 1/(sd * np.sqrt(2 * np.pi)) * np.exp(-(x - mean)**2
/ (2 * sd**2))
    return prob_density

# 3 Custom Exponential Distribution
# Used in Problem 5,6,7
def exp_f1(x, a):
```

```
    return np.exp(-a*x)
```

Problem 1:

Generate 100,000 samples of an exponentially distributed random variable with a mean of 8. Generate a histogram of these samples, normalize to have the same area as a PDF. Plot your histogram and the theoretical PDF function together on the same figure. They should match.

```
# Generate 100,000 samples of an exponentially distributed random
variable with a mean of 8.
mean = 8        # Lamda
samples_count = 100000
total_bins = 50

samples_arr = np.random.exponential(scale=mean, size=samples_count)
print("Samples: ", samples_arr)

fig, axs = plt.subplots(1, 1, figsize=(15, 5))

values, bins, _ = axs.hist(x=samples_arr, bins=total_bins,
density=True, edgecolor='black', linewidth=1, label="Experimental")
# print(sum(values))
area = sum(np.diff(bins)*values)
print("Total Area:", area)

x = np.linspace(np.min(samples_arr), np.max(samples_arr),
samples_count)
axs.plot(x, exp_f(x, 1/mean), color='black', linewidth=3,
label="Theoretical")
axs.set_xlabel('x', fontweight ='bold')
axs.set_ylabel('f(x)', fontweight ='bold')
axs.legend()

plt.title("Histogram vs Theoretical PDF")
plt.style.use('fivethirtyeight')
plt.show()

Samples:  [ 0.62583467  0.06559114 36.69397598 ... 11.39855094
5.35167342
  6.38794004]
Total Area: 0.9999999999999993
```



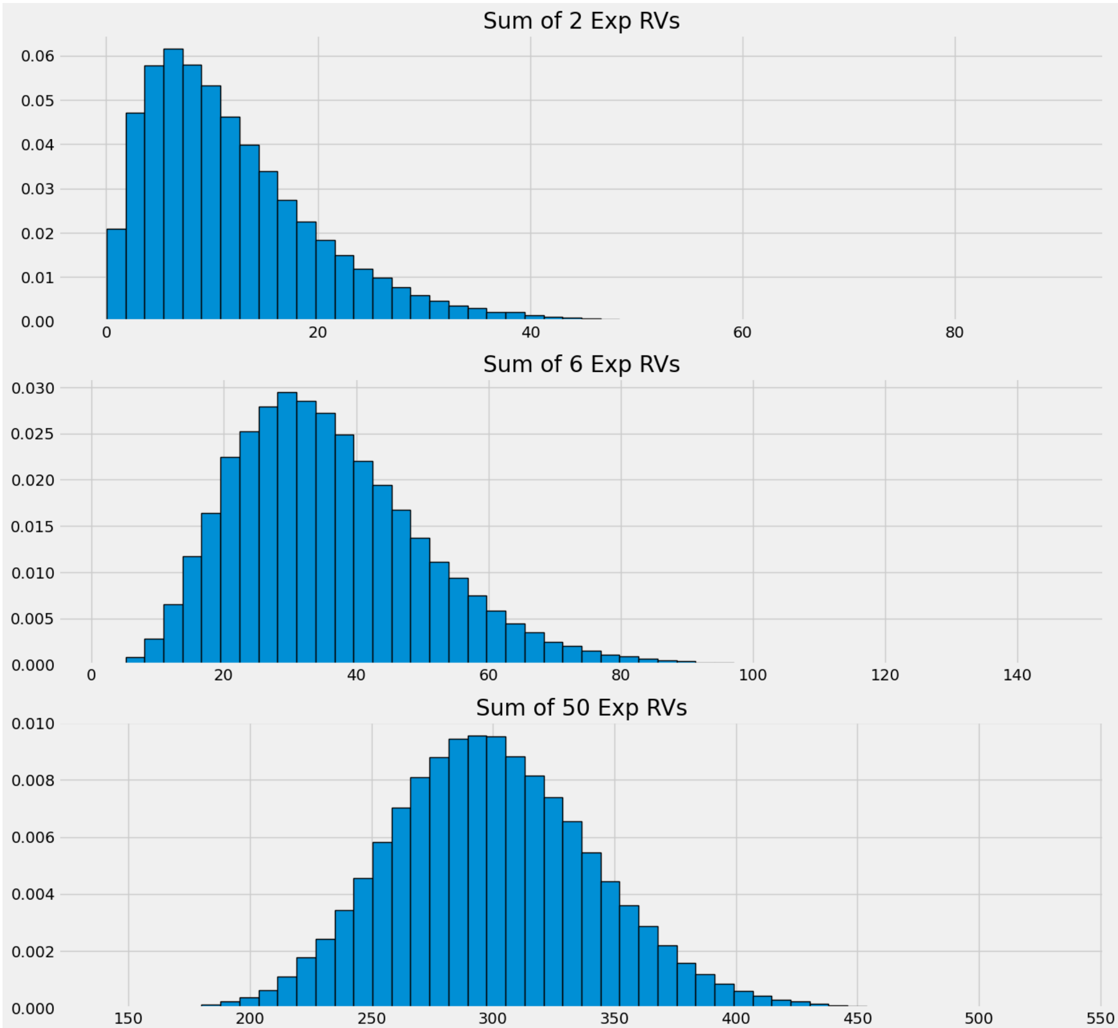
Problem 2:

Generate 100,000 samples of the sum of 2, 6 and 50 exponentially distributed random variables, each with a mean of 6. Create histograms of each sum, normalize to have the same area as a PDF and plot. For each distribution, choose the number of histogram bins that produce plots that clearly show the shape of the distribution.

```
from matplotlib.pyplot import title


mean = 6
samples_count = 100000
total_bins = 50

# Sum of 2 exponentially distributed random variables
samples_arr_2 = 0
for counter in range(2):
    samples_arr_2 = samples_arr_2 + np.random.exponential(scale=mean,
size=samples_count)

# Sum of 6 exponentially distributed random variables
samples_arr_6 = 0
for counter in range(6):
    samples_arr_6 = samples_arr_6 + np.random.exponential(scale=mean,
size=samples_count)

# Sum of 50 exponentially distributed random variables
samples_arr_50 = 0
for counter in range(50):
    samples_arr_50 = samples_arr_50 +
np.random.exponential(scale=mean, size=samples_count)

fig, axs = plt.subplots(3, 1, figsize=(15, 15))

values, bins, _ = axs[0].hist(x=samples_arr_2, bins=total_bins,
density=True, edgecolor='black', linewidth=1)
area = sum(np.diff(bins)*values)
```

```
print("Total Area:", area)
values, bins, _ = axs[1].hist(x=samples_arr_6, bins=total_bins,
density=True, edgecolor='black', linewidth=1)
area = sum(np.diff(bins)*values)
print("Total Area:", area)
values, bins, _ = axs[2].hist(x=samples_arr_50, bins=total_bins,
density=True, edgecolor='black', linewidth=1)
area = sum(np.diff(bins)*values)
print("Total Area:", area)
axs[0].set_title("Sum of 2 Exp RVs")
axs[1].set_title("Sum of 6 Exp RVs")
axs[2].set_title("Sum of 50 Exp RVs")

plt.style.use('fivethirtyeight')
plt.show()
```

```
Total Area: 0.9999999999999998
Total Area: 0.9999999999999997
Total Area: 0.9999999999999999
```



Bin Merging Script

Verified on Assignment 3 Problem 1

```
def merge_bins(bins_mid, samples, samples_count, isNormalized=True):
    #Here Samples are Normalized to samples_count
    # #We need to de normalized it before merging bins
    # For Chi square each bin must have atleast 5 samples

    merged_bins_mid = 0
    merged_samples = 0
    merged_samples_normalized = 0
    min_sample_in_bin = 5
    print("Min Sample in one bin: ", min_sample_in_bin)

    if isNormalized:
        total_samples = sum(samples)
        denormalized_samples =
np.around(samples*samples_count/total_samples)
        merged_samples = denormalized_samples.copy()
```

```
        merged_samples_normalized = samples.copy()
    else:
        merged_samples = samples.copy()
        merged_samples_normalized = samples.copy()
    merged_bins_mid = bins_mid.copy()

    #TODO Merge bins algorithm
    index = 0
    while index < len(merged_samples) - 1:
        if (merged_samples[index] < min_sample_in_bin):
            #Merge 2 cells in merged_samples array and delete one cell
            merged_samples[index+1] = merged_samples[index] +
merged_samples[index+1]
            # del merged_samples[index]
            merged_samples = np.delete(merged_samples, index)

            #Merge 2 Cells in merged_samples_normalized array and
delete one cell
            merged_samples_normalized[index+1] =
merged_samples_normalized[index] + merged_samples_normalized[index+1]
            # del merged_samples[index]
            merged_samples_normalized =
np.delete(merged_samples_normalized, index)

            #Merge 2 cells in bin array and delete one cell
            merged_bins_mid[index+1] = (merged_bins_mid[index] +
merged_bins_mid[index+1])/2
            # del merged_bins_mid[index]
            merged_bins_mid = np.delete(merged_bins_mid, index)
        else:
            index = index + 1
    #Ff last sample is less then limit then we need to merge
    if merged_samples[index] < min_sample_in_bin:
        merged_samples[index-1] = merged_samples[index-1] +
merged_samples[index]
        # del merged_samples[index]
        merged_samples = np.delete(merged_samples, index)

        merged_samples_normalized[index-1] =
merged_samples_normalized[index-1] + merged_samples_normalized[index]
        # del merged_samples[index]
        merged_samples_normalized =
np.delete(merged_samples_normalized, index)

        merged_bins_mid[index-1] = (merged_bins_mid[index-1] +
merged_bins_mid[index])/2
        # del merged_bins_mid[index]
        merged_bins_mid = np.delete(merged_bins_mid, index)
    # print(merged_samples)
```

```
    return merged_bins_mid, merged_samples_normalized


def get_bin_edges(leftmost_edge, bins_mid_merged, binwidth):
    '''
    @Parameters:leftmost_edge: Left most edge of the histogram
                binwidth: Constant Bin Width
                bin_index: Index of the Bin of which limits is to be 
returned
    @Return:    left_edges
    @Description: To get left edges of merged bins for integration 
limits
    '''
    left_edges = [None]*(len(bins_mid_merged)+1)
    left_edges[0] = leftmost_edge
    index = 0

    while index < len(bins_mid_merged):
        left_edges[index+1] = 2*bins_mid_merged[index] - 
left_edges[index]
        index = index + 1

    return left_edges
# hk1, bins_left1, _ = axs.hist(x=samples_arr_50, bins=total_bins, 
density=True, edgecolor='black', linewidth=1)
# hk2, bins_left2, _ = axs.hist(x=samples_arr_50, bins=total_bins, 
density=False, edgecolor='black', linewidth=1)
# print(hk1[0])
# print(hk2[0])
# print(hk1[0]*100000/sum(hk1))

print("Checking Logic with Assignment 3 Q1")
bins_mid = [4.09885, 4.29443, 4.49001, 4.68559, 4.88117, 5.07675, 
5.27233, 5.4679, 5.66348, 5.85905]
samples =  [4, 7, 4, 3, 7, 4, 6, 4, 5, 6]
print("bins_mid: ", bins_mid)
print("samples: ", samples)
#Samples count only needed if isNormalized = True otherwise
merged_bins, merged_samples = merge_bins(bins_mid=bins_mid, 
samples=samples, samples_count=0,
                                        isNormalized=False)
print("bins_mid_merged: ", merged_bins)
print("samples_merged: ", merged_samples)

binwidth = bins_mid[1] - bins_mid[0]
print("Binwidth: ", binwidth)
leftmost_edge = bins_mid[0] - binwidth/2
merged_bins_left = get_bin_edges(leftmost_edge, merged_bins, binwidth)
```

```
print("bins_left_merged: ", merged_bins_left)
```

```
Checking Logic with Assignment 3 Q1
bins_mid:  [4.09885, 4.29443, 4.49001, 4.68559, 4.88117, 5.07675,
5.27233, 5.4679, 5.66348, 5.85905]
samples:  [4, 7, 4, 3, 7, 4, 6, 4, 5, 6]
Min Sample in one bin:  5
bins_mid_merged:  [4.19664 4.5878  4.88117 5.17454 5.56569 5.85905]
samples_merged:  [11  7  7 10  9  6]
Binwidth:  0.19558000000000053
bins_left_merged:  [4.001059999999999, 4.392220000000002,
4.783379999999975, 4.978960000000025, 5.370119999999998,
5.761260000000002, 5.956839999999998]
```

Integration Verification

Verified on Exponential and Gaussian Distribution

```
# Integration on Exponential Distribution
mean = 8
lower_limit = 0
upper_limit = np.Infinity
f = lambda x: exp_f(x, mean)
print("Integrating Exponential Distribution with mean {mean} from \
{lower_limit} to {upper_limit} = {value}" \
.format(mean = mean, lower_limit = lower_limit, upper_limit =
upper_limit, \
value = quad(f, lower_limit, upper_limit)[0]))

# Integration on Normal Distribution
mean = 0
sd = 1
lower_limit = -np.Infinity
upper_limit = 0
f = lambda x: normal_dist(x , mean , sd)
print("Integrating Normal Distribution with mean {mean} and \
sd {sd} from {lower_limit} to {upper_limit} = {value}" \
.format(mean = mean, sd = sd, lower_limit = lower_limit, upper_limit =
upper_limit, \
value = quad(f, lower_limit, upper_limit)[0]))
```

```
Integrating Exponential Distribution with mean 8 from 0 to inf =
1.0000000000000002
Integrating Normal Distribution with mean 0 and sd 1 from -inf to 0 =
0.49999999999999983
```

Problem 3:

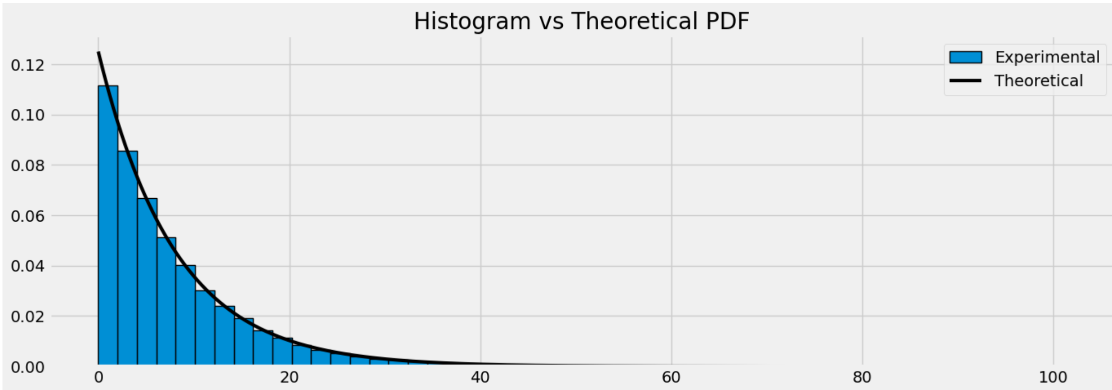
Apply a Chi-squared goodness-of-fit test to see if the random vector you generated in Problem 1 matches a theoretical exponential distribution. Your test should calculate and display a confidence value that should reveal your vector of random numbers does match an exponential distribution.

```
# Generate 100,000 samples of an exponentially distributed random
variable with a mean of 8.
mean = 8        # Lamda
samples_count = 100000
total_bins = 50

samples_arr = np.random.exponential(scale=mean, size=samples_count)
print("Samples: ", samples_arr)
print("Total Samples: ", len(samples_arr))

fig, axs = plt.subplots(1, 1, figsize=(15, 5))

# hk: Experimental Samples, ek: Expected Samples
hk, bins_left, _ = axs.hist(x=samples_arr, bins=total_bins,
density=True, edgecolor='black', linewidth=1, label="Experimental")
#Convert bins to numpy array
bins_left = np.array(bins_left)
print("Min Sample: ", np.min(samples_arr))
print("Max Sample: ", np.max(samples_arr))
binwidth = (np.diff(bins_left))[0]          #binwidth is same for all
bins
bins_mid = bins_left + binwidth/2           #Generate bin_mid array
containing midpoint of all bins
bins_mid = bins_mid[:-1]                    #Droppping Last Element of
bin_mid since bins have extra element right edge of last bin
total_bins = len(bins_mid)                  #Should be equal to
total_bins defined earlier
print("Total Bins: ", total_bins)
print("Binwidth: ", binwidth)
print("1st Bin Left Edge: ", bins_left[0])
print("1st Bin Midpoint: ", bins_mid[0])
print("1st Bin Right Edge: ", bins_left[1])
print("{total_bins}th Bin Left Edge:
{value}".format(total_bins=total_bins, value=bins_left[total_bins-1]))
print("{total_bins}th Bin Midpoint:
{value}".format(total_bins=total_bins, value=bins_mid[total_bins-1]))
print("{total_bins}th Bin Right Edge:
{value}".format(total_bins=total_bins, value=bins_left[total_bins]))

#Generate Expected samples using distribution for each bid midpoint
ek = exp_f(bins_mid, mean)
print("1st Expected Value: ", ek[0])
print("1st Experimental Value: ", hk[0])
```

```
print("{total_bins}th Expected Value:
{value}".format(total_bins=total_bins, value=ek[total_bins-1]))
print("{total_bins}th Experimental Value:
{value}".format(total_bins=total_bins, value=hk[total_bins-1]))


#Need to merge bins based on value
bins_mid_merged, hk_merged = merge_bins(bins_mid, hk, samples_count,
True)
bins_left_merged = get_bin_edges(bins_left[0], bins_mid_merged,
binwidth)
index = 0
# print("bins_mid_merged[0]", bins_mid_merged[0])
# print("bins_left_merged[0]", bins_left_merged[0])
# print("bins_left_merged[1]", bins_left_merged[1])
total_bins_merged = len(bins_mid_merged)

# Integrate from left edge to right edge of all bins
ek_merged = [None]*(total_bins_merged)
f = lambda x: exp_f(x, 1/mean)
index = 0
while index < total_bins_merged:
    hk_merged[index] = hk_merged[index]*(bins_left_merged[index+1] -
bins_left_merged[index])
    ek_merged[index] = quad(f, bins_left_merged[index],
bins_left_merged[index+1])[0]
    index = index + 1
#print("ek_merged", ek_merged)
#print("hk_merged", hk_merged)

print("---------After Bin Merge----------------------------")
print("Total Bins After Merge: ", len(bins_mid_merged))
print("1st Expected Value: ", ek_merged[0])
print("1st Experimental Value: ", hk_merged[0])
print("{total_bins}th Expected Value:
{value}".format(total_bins=total_bins_merged,
value=ek_merged[total_bins_merged-1]))
print("{total_bins}th Experimental Value:
{value}".format(total_bins=total_bins_merged,
value=hk_merged[total_bins_merged-1]))


C = np.sum(((hk_merged - ek_merged)**2)/ek_merged)
DOF = len(hk_merged) - 1
print("C: ", C)
print("DOF: ", DOF)

#Get Confidance Value
p_value = 1 - scipy.stats.chi2.cdf(C, DOF)
```

```
print("\033[92m\033[1mConfidence: {p_value} \
033[0m".format(p_value=p_value))


# area = sum(np.diff(bins_left)*hk)
# area = sum((binwidth)*hk)
# print("Total Area:", area)
plt.title("Histogram vs Theoretical PDF")

x = np.linspace(np.min(samples_arr), np.max(samples_arr),
samples_count)
axs.plot(x, exp_f(x, 1/mean), color='black', linewidth=3,
label="Theoretical")
axs.legend()

plt.style.use('fivethirtyeight')
plt.show()
```

```
Samples:  [1.16943336 7.21458462 5.26033326 ... 1.56100396 1.84361897
8.56886362]
Total Samples:  100000
Min Sample:  3.822705153580897e-06
Max Sample:  101.34145047533714
Total Bins:  50
Binwidth:  2.0268289330526397
1st Bin Left Edge:  3.822705153580897e-06
1st Bin Midpoint:  1.0134182892314734
1st Bin Right Edge:  2.0268327557577934
50th Bin Left Edge: 99.3146215422845
50th Bin Midpoint: 100.32803600881081
50th Bin Right Edge: 101.34145047533714
1st Expected Value:  0.002410539320268085
1st Experimental Value:  0.11152889931344252
50th Expected Value: 0.0
50th Experimental Value: 4.933815497166222e-06
Min Sample in one bin:  5
---------After Bin Merge----------------------------
Total Bins After Merge:  35
1st Expected Value:  0.22380653464895445
1st Experimental Value:  0.22604999999999995
35th Expected Value: 0.00014076100345604317
35th Experimental Value: 0.0013500061035156285
C:  0.010861239789922648
DOF:  34
Confidence: 1.0
```



Problem 4:

Apply a Chi-squared goodness-of-fit test to the sum of 50 exponentially distributed random vectors from Problem 2 and see if it matches a Gaussian theoretical distribution. In your PDF file, comment on what these results say about the utility of the central limit theorem in this particular case.

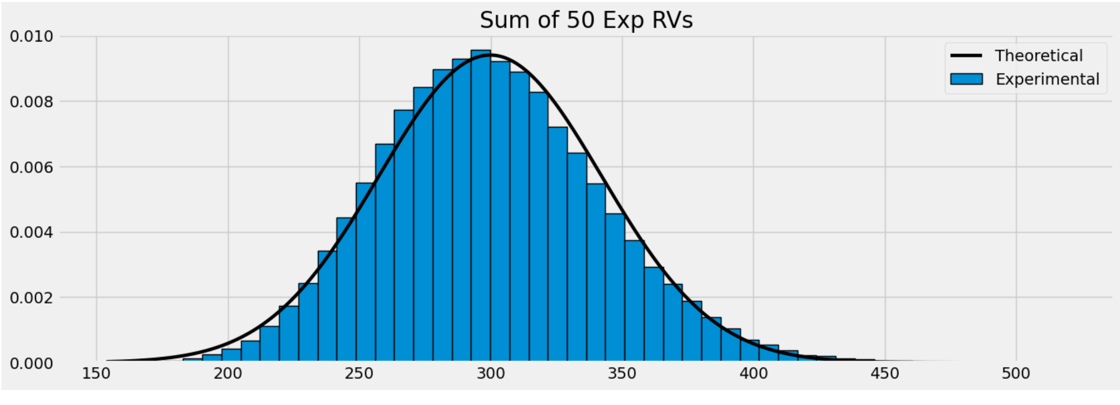
```
import math
from matplotlib.pyplot import title


mean = 6
samples_count = 100000
total_bins = 50

#For Normal Distribution
normal_mean = 0
normal_variance = 0
normal_std_dev = 0

# Sum of 50 exponentially distributed random variables
# We will also calculate mean and Std Deviation so that we can use it
for normal distribution later on
# Central Limit Theorm:
# Y_mean = X1_mean + X2_mean + ... + Xn_mean
# Y_variance = X1_variance + X2_variance + ... + Xn_variance
samples_arr_50 = 0
for counter in range(50):
    samples_arr = np.random.exponential(scale=mean,
size=samples_count)
    normal_mean = normal_mean + np.mean(samples_arr)
    normal_variance = normal_variance + np.var(samples_arr)
    samples_arr_50 = samples_arr_50 + samples_arr

normal_std_dev = math.sqrt(normal_variance)
print("Normal Mean Value: ", normal_mean)
print("Normal Variance Value: ", normal_variance)
print("Normal Standard Deviation Value: ", normal_std_dev)
```

```
fig, axs = plt.subplots(1, 1, figsize=(15, 5))

x = np.linspace(np.min(samples_arr_50), np.max(samples_arr_50),
samples_count)
axs.plot(x, normal_dist(x, normal_mean, normal_std_dev),
color='black', linewidth=3, label="Theoretical")

# hk: Experimental Samples, ek: Expected Samples
hk, bins_left, _ = axs.hist(x=samples_arr_50, bins=total_bins,
density=True, edgecolor='black', linewidth=1, label="Experimental")
#Convert bins to numpy array
bins_left = np.array(bins_left)
print("Min Sample: ", np.min(samples_arr_50))
print("Max Sample: ", np.max(samples_arr_50))
binwidth = (np.diff(bins_left))[0]          #binwidth is same for all
bins
bins_mid = bins_left + binwidth/2           #Generate bin_mid array
containing midpoint of all bins
bins_mid = bins_mid[:-1]                    #Droppping Last Element of
bin_mid since bins have extra element right edge of last bin
total_bins = len(bins_mid)                  #Should be equal to
total_bins defined earlier
print("Total Bins: ", total_bins)
print("Binwidth: ", binwidth)
print("1st Bin Left Edge: ", bins_left[0])
print("1st Bin Midpoint: ", bins_mid[0])
print("1st Bin Right Edge: ", bins_left[1])
print("{total_bins}th Bin Left Edge:
{value}".format(total_bins=total_bins, value=bins_left[total_bins-1]))
print("{total_bins}th Bin Midpoint:
{value}".format(total_bins=total_bins, value=bins_mid[total_bins-1]))
print("{total_bins}th Bin Right Edge:
{value}".format(total_bins=total_bins, value=bins_left[total_bins]))


#
#Generate Expected samples using distribution for each bid midpoint
ek = normal_dist(bins_mid, normal_mean, normal_std_dev)
print("1st Expected Value: ", ek[0])
print("1st Experimental Value: ", hk[0])
print("{total_bins}th Expected Value:
{value}".format(total_bins=total_bins, value=ek[total_bins-1]))
print("{total_bins}th Expected Value:
{value}".format(total_bins=total_bins, value=hk[total_bins-1]))


#Need to merge bins based on value
bins_mid_merged, hk_merged = merge_bins(bins_mid, hk, samples_count,
```

```
True)
bins_left_merged = get_bin_edges(bins_left[0], bins_mid_merged,
binwidth)
print("bins_mid_merged[0]", bins_mid_merged[0])
print("bins_left_merged[0]", bins_left_merged[0])
print("bins_left_merged[1]", bins_left_merged[1])
total_bins_merged = len(bins_mid_merged)

# Integrate from left edge to right edge of all bins
ek_merged = [None]*(total_bins_merged)
f = lambda x: normal_dist(x, normal_mean, normal_std_dev)
index = 0
while index < total_bins_merged:
    hk_merged[index] = hk_merged[index]*(bins_left_merged[index+1] -
bins_left_merged[index])
    ek_merged[index] = quad(f, bins_left_merged[index],
bins_left_merged[index+1])[0]
    index = index + 1
#print("Value", quad(f, bins_left_merged[0], bins_left_merged[22])[0])
# print("ek_merged", ek_merged)
# print("hk_merged", hk_merged)


print("---------After Bin Merge----------------------------")
print("Total Bins After Merge: ", len(bins_mid_merged))
print("1st Expected Value: ", ek_merged[0])
print("1st Experimental Value: ", hk_merged[0])
print("{total_bins}th Expected Value:
{value}".format(total_bins=total_bins_merged,
value=ek_merged[total_bins_merged-1]))
print("{total_bins}th Experimental Value:
{value}".format(total_bins=total_bins_merged,
value=hk_merged[total_bins_merged-1]))


C = np.sum(((hk_merged - ek_merged)**2)/ek_merged)
DOF = len(hk_merged) - 2
print("C: ", C)
print("DOF: ", DOF)

#Get Confidance Value
p_value = 1 - scipy.stats.chi2.cdf(C, DOF)
print("\033[92m\033[1mConfidence: {p_value} \
033[0m".format(p_value=p_value))




area = sum(np.diff(bins)*values)
print("Total Area:", area)
```

```
axs.set_title("Sum of 50 Exp RVs")
axs.legend()

plt.style.use('fivethirtyeight')
plt.show()
```

```
Normal Mean Value:  299.85041876157516
Normal Variance Value:  1797.7990792575183
Normal Standard Deviation Value:  42.400460837796544
Min Sample:  153.72077987653503
Max Sample:  519.0485486789069
Total Bins:  50
Binwidth:  7.3065553760474415
1st Bin Left Edge:  153.72077987653503
1st Bin Midpoint:  157.37405756455877
1st Bin Right Edge:  161.02733525258247
50th Bin Left Edge: 511.74199330285944
50th Bin Midpoint: 515.3952709908832
50th Bin Right Edge: 519.0485486789069
1st Expected Value:  3.324006443921395e-05
1st Experimental Value:  2.73726796974188e-06
50th Expected Value: 2.3010002803966926e-08
50th Expected Value: 1.3686339848709453e-06
Min Sample in one bin:  5
bins_mid_merged[0] 161.02733525258247
bins_left_merged[0] 153.72077987653503
bins_left_merged[1] 168.33389062862992
---------After Bin Merge---------------------------
Total Bins After Merge:  45
1st Expected Value:  0.0006777956283623157
1st Experimental Value:  0.00018
45th Expected Value: 8.085723530986595e-06
45th Experimental Value: 0.0004749999999999973
C:  0.041499776007957
DOF:  43
Confidence: 1.0
Total Area: 0.9999999999999999
```



Comments:

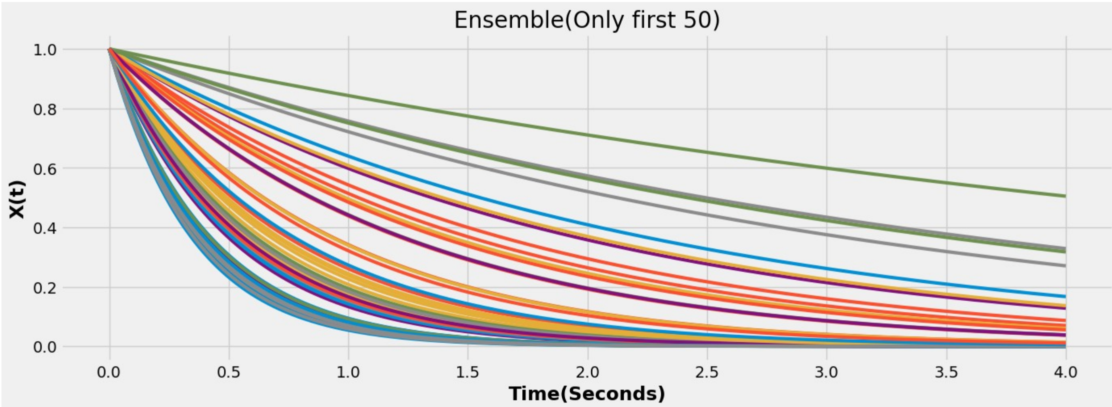
- Above histogram plot is sum of 50 exponentially distributed random variables. As we can see from the blue plot it is similar to to normal distribution.
- As per central limit thoeorm:
- Y_mean = X1_mean + X2_mean + … + Xn_mean = (around 8.33)
- Y_variance = X1_variance + X2_variance + … + Xn_variance = (around 1.38)
- Now if we plot theoretical normal distribution with above Y_mean and Y_variance it should match according to central limit theorm.
- As we can see from the black plot it perfectly matching with out experimental blue plot and we also verfied goodness of fit using Chi square method. COnfidence results from the test is comming out 100% in this case.

Problem 5:

Create a 10,000 waveform ensemble of a stochastic process where the waveform is X(t) = exp(−Y t), where Y is uniformly distributed between 0 and 3. Your time vector should go from 0 to 4 seconds with a sampling interval of 1 ms.

```
#Generate 10,000 samples of Y Uniformaly distributed between 0 and 3
total_wavefroms = 10000
min_time = 0            # 0 Second
max_time = 4            # 4 Second
sampling_interval = 0.001   # 1 Milisecond
# Y is Uniformly distributed between 0 to 3: Y ~ U(0,3)
b = 3
total_time_samples = int((max_time-min_time)/sampling_interval)
Y = np.random.uniform(low=0.0, high=b, size=total_wavefroms)
fig, axs = plt.subplots(1, 1, figsize=(15, 5))

# ensemble 2D array contains all samples for all generated waveforms
ensemble = np.zeros((total_wavefroms, total_time_samples),
dtype=float)
x = np.linspace(min_time, max_time, total_time_samples)
for counter in range(total_wavefroms):
    ensemble[counter] = exp_f1(x, Y[counter])

print("Ensemble Shape: ", ensemble.shape)
print("Plotting first 50 wavefroms from Ensemble...")
for counter in range(50):
    axs.plot(x, ensemble[counter], linewidth=3) #color='black'

axs.set_xlabel('Time(Seconds)', fontweight ='bold')
axs.set_ylabel('X(t)', fontweight ='bold')
plt.style.use('fivethirtyeight')
plt.title("Ensemble(Only first 50)")
plt.show()
```

```
Ensemble Shape:  (10000, 4000)
Plotting first 50 wavefroms from Ensemble...
```



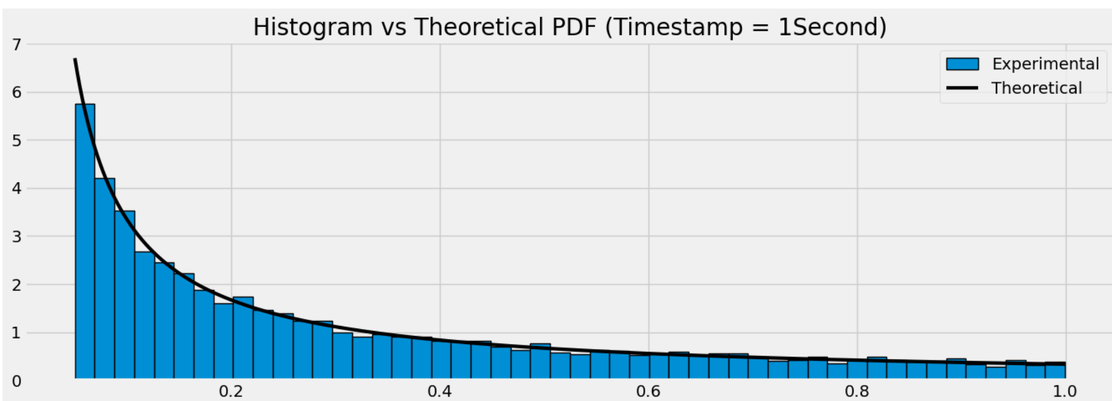
Problem 6:

Use your 10,000 waveform ensemble from Problem 5 to numerically calculate a histogram that represents the first order PDF of this stochastic process. Normalize your histogram to have the same area as a PDF and plot your histogram on the same figure as the theoretical expression for the first order PDF for this stochastic process. They should match. You can generate your plot for a single time sample that does a good job of illustrating the zero and non-zero regions of the PDF.

```
# Theoretical first order pdf of Exponential process
# X(t) = exp(-Yt)
# b: Y ~ U(0,b)
# t: Timestamp
def first_order_pdf(x, b, t):
    return 1/(b*abs(t*x))

time_stamp = 1          #To get samples from ensembles at
time_stamp=1Second
time_stamp_index = int(time_stamp/sampling_interval)
#print(time_stamp_index)

#Get 1000th column of ensemble
#samples_1000: samples at 1000ms timestamp
samples_1000 = ensemble[:, time_stamp_index]
samples_count = len(samples_1000)
print(np.mean(samples_1000))
print((1-np.exp(-3))/3)
print("Samples Count: ", samples_count)


total_bins = 50
fig, axs = plt.subplots(1, 1, figsize=(15, 5))
values, bins, _ = axs.hist(x=samples_1000, bins=total_bins,
density=True, edgecolor='black', linewidth=1, label="Experimental")
# print(sum(values))
area = sum(np.diff(bins)*values)
print("Total Area:", area)
```

```python
#Plotting theoretical first order pdf at time stamp 1second
x = np.linspace(np.min(samples_1000), np.max(samples_1000),
samples_count)
axs.plot(x, first_order_pdf(x, b, time_stamp), color='black',
linewidth=3, label="Theoretical")
axs.legend()

plt.style.use('fivethirtyeight')
plt.title("Histogram vs Theoretical PDF (Timestamp = 1Second)")
```

```
0.31859872940660794
0.3167376438773787
Samples Count:  10000
Total Area: 0.9999999999999999

Text(0.5, 1.0, 'Histogram vs Theoretical PDF (Timestamp = 1Second)')
```



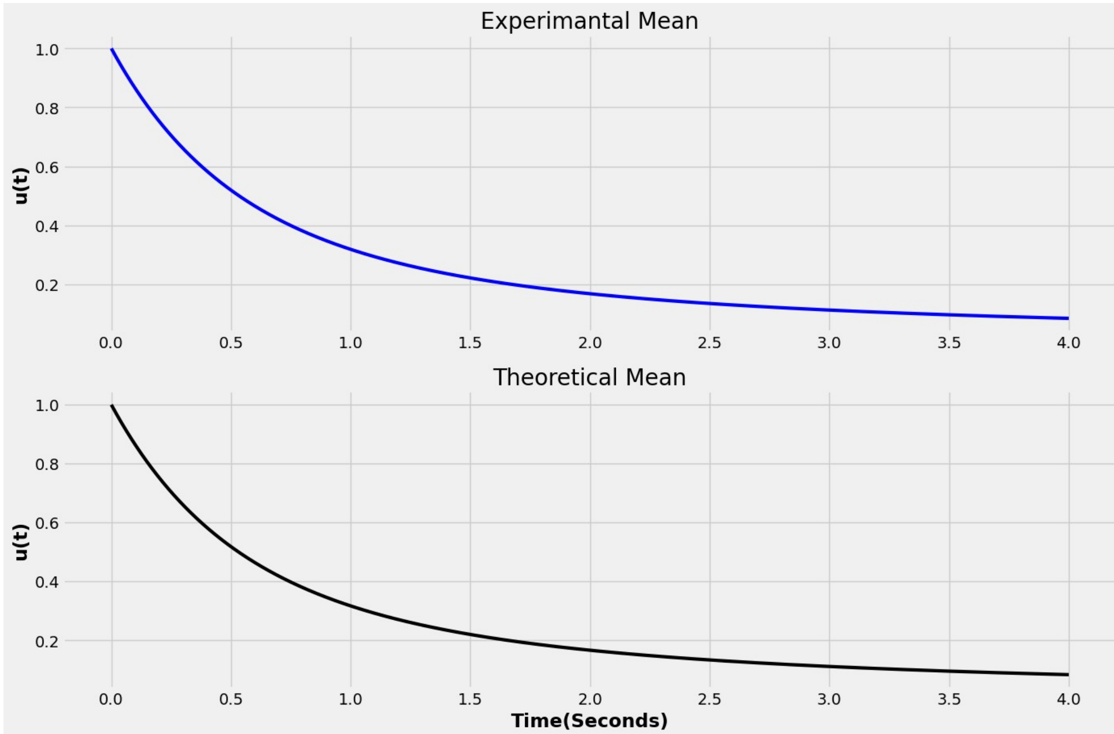
Problem 7:

Use your 10,000 waveform ensemble from Problem 5 to numerically calculate the mean of the stochastic process. Plot the numerical mean along with the theoretical mean expression on the same figure. They should match.

```python
# Theoretical mean of exponential process
# X(t) = exp(-Yt)
# b: Y ~ U(0,b)
# t: Timestamp
# u(t) = 1/bt(1-exp(-bt))
def theoretical_mean(t, b):
    if(t == 0):         #to avoid divide by zero error
        return 1
    return (1-np.exp(-b*t))/(b*t)


# Calculate mean at each timestamp from ensembles
mean_experimental = np.zeros(total_time_samples, dtype=float)
mean_theoretical = np.zeros(total_time_samples, dtype=float)
for time_stamp in range(total_time_samples):
```

```
    mean_experimental[time_stamp] = np.mean(ensemble[:, time_stamp])
    mean_theoretical[time_stamp] = 
theoretical_mean(time_stamp*sampling_interval, b)
print("Means shape: ", mean_experimental.shape)
print("Experimental Means: ", mean_experimental)
print("Theoretical Means: ", mean_theoretical)
mse = (np.square(mean_theoretical - mean_experimental)).mean()
print("MSE Theoretical vs Experimental Mean: ", mse)

#Plotting theoretical mean vs experimental mean for each time stamp
fig, axs = plt.subplots(2, 1, figsize=(15, 10))
x = np.linspace(min_time, max_time, total_time_samples)
axs[0].plot(x, mean_experimental, color='blue', linewidth=3)
axs[0].set_title("Experimantal Mean")

axs[1].plot(x, mean_theoretical, color='black', linewidth=3)
axs[1].set_title("Theoretical Mean")

axs[1].set_xlabel('Time(Seconds)', fontweight ='bold')
axs[0].set_ylabel('u(t)', fontweight ='bold')
axs[1].set_ylabel('u(t)', fontweight ='bold')

plt.style.use('fivethirtyeight')
```

```
Means shape:  (4000,)
Experimental Means:  [1.         0.99850583 0.99701464 ... 0.08481224
0.08479113 0.08477004]
Theoretical Means:  [1.         0.9985015  0.99700599 ... 0.08339536
0.08337451 0.08335366]
MSE Theoretical vs Experimental Mean:  2.807011094227653e-06
```


Experimantal Mean
u(t)
1.0
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Theoretical Mean
u(t)
1.0
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Time(Seconds)