Project 4 - Central Limit Theorem

CECS 381 - Sec 06

Dustin Martin - 015180085

Problem 1:

- **Introduction**
  Problem 1 introduces the idea of simulating Uniform, Exponential, and Normal random variable distributions using Python code as well as graphing the theoretical curve over it, allowing us to compare our simulations with a perfect representation using the formula for Uniform, Exponential, and Normal distributions.

- **Methodology**
  - **1.1 Uniform**
    Simulating the Uniform distribution is quite simple, thanks to a handy numpy module `random.uniform`, which we used to generate 10,000 numbers between `a` and `b` (2.0 and 5.0 respectively).These are then plotted on a bar graph, giving us a uniform distribution between 2 and 5. We can then recalculate the mean and STD using our simulated values to compare with our theoretical ones.
    To draw the line graph of a theoretical, perfect uniform distribution, we used the formula:

$$f(x) = \begin{cases} \frac{1}{(b-a)}, & a \leq x \leq b \\ 0, & \text{otherwise} \end{cases}$$

    Which gives us the flat line used to represent the probability density of the uniform distribution between 2 and 5.

- **1.2 Exponential**

  To simulate the Exponential distribution, we use the numpy module `random.exponential`, which generates numbers using our selected β (0.33) as the mean and STD. These are then plotted on a bar graph, giving us an exponential distribution centered around β. Using the generated array, we can calculate our own mean and std and compare to our theoretical value.

  To draw the line graph of a theoretical, perfect exponential distribution, we used the formula:

  $$f_T(t\ ;\ \beta) = \begin{cases} \frac{1}{\beta}\, exp(-\frac{1}{\beta}t), & t \geq 0 \\ 0, & t < 0 \end{cases}$$

  Which gives us the exponential line used to represent the probability density of the exponential distribution.

- **1.3 Normal**

  To simulate the Normal distribution, we use the numpy module `random.normal`, which generates numbers using our selected mean and STD, (2.5 and 0.75 respectively). These are then plotted on a bar graph, giving us a normal distribution centered around our mean and STD. Using the generated array, we can calculate our own mean and std and compare to our theoretical value.

  To draw the line graph of a theoretical, perfect normal distribution, we used the formula:

  $$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\{-\frac{(x-\mu)^2}{2\sigma^2}\}$$

  Which gives us the normal line used to represent the probability density of the exponential distribution.

- **Results**
  - **1.1 Uniform**

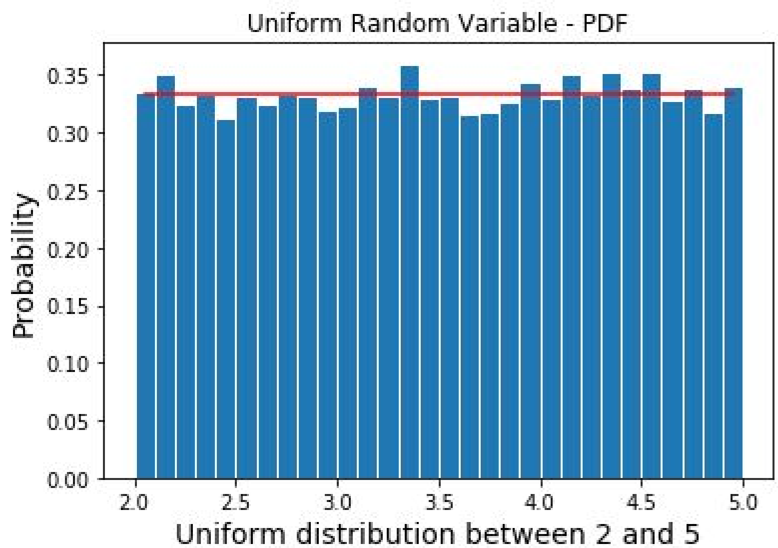


Table 1: Statistics for a Uniform Distribution

Expectation		**Standard Deviation**	
Theoretical Calculation	Experimental Measurement	Theoretical Calculation	Experimental Measurement
3.5	~3.5005	0.75	~0.8671

As seen in the above chart, most of the readings are on point apart from the experimental measurement for the standard deviation, which, ironically, has slightly deviated.

The bar-graph shows the experimental distribution while the red line is the theoretical probability density using the aforementioned formula.

- **1.2 Exponential**

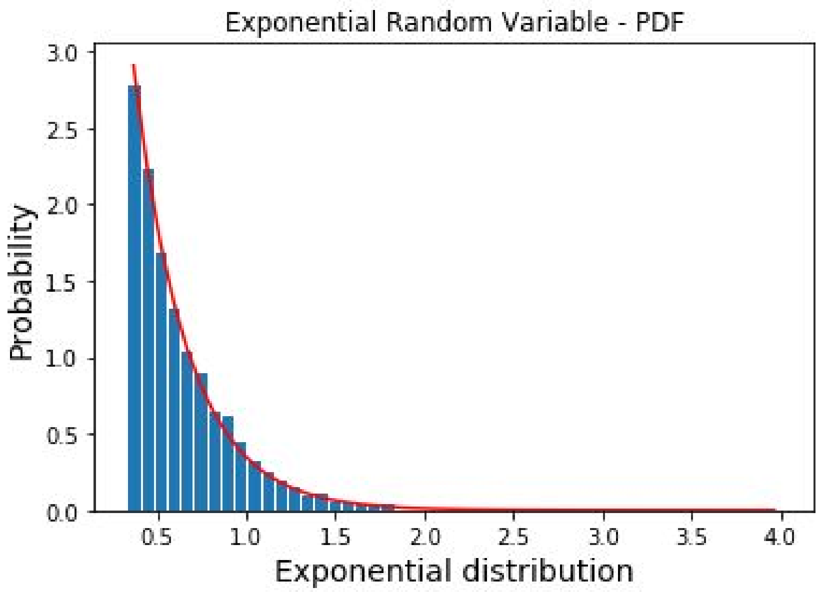


Table 2: Statistics for an Exponential Distribution			
Expectation		Standard Deviation	
Theoretical Calculation	Experimental Measurement	Theoretical Calculation	Experimental Measurement
0.33	~0.3342	0.33	~0.3334

- The blue bar-graph shows the simulated distribution of the exponential RV. The red line is the calculated, theoretical distribution.
  As seen in both the visual representation and recorded readings in the chart, the numbers are on-point.

- **1.3 Normal**

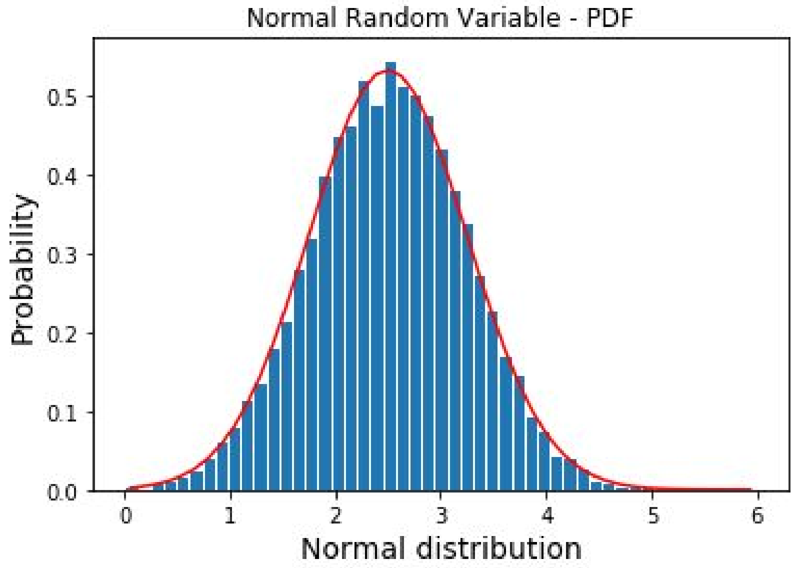


Table 3: Statistics for a Normal Distribution

Expectation		**Standard Deviation**	
Theoretical Calculation	Experimental Measurement	Theoretical Calculation	Experimental Measurement
2.5	~2.4998	0.75	~0.75002

As with the previous graphs, the bar graph shows the simulated distribution across the Normal RV and the red line is the calculated theoretical values.

- **Appendix/Source Code**
  - **1.1**

```
import numpy as np
import matplotlib.pyplot as plt

def UnifPDF(a,b,x):
    f=(1/abs(b-a))*np.ones(np.size(x))
    return f

a=2.0
b=5.0
n=10000

x= np.random.uniform(a,b,n)

#Create bins and histogram
nbins=30 # Number of bins
edgecolor='w'  # Color separating bars in the bargraph
bins=[float(x) for x in np.linspace(a, b,nbins+1)]
h1, bin_edges = np.histogram(x,bins,density=True) # Define points
on the horizontal axis
be1=bin_edges[0:np.size(bin_edges)-1]
be2=bin_edges[1:np.size(bin_edges)]
b1=(be1+be2)/2
barwidth=b1[1]-b1[0] # Width of bars in the bargraph
plt.close('all')
# PLOT THE BAR GRAPH
plt.title('Uniform Random Variable - PDF')
plt.xlabel('Uniform distribution between 2 and 5',fontsize=14)
plt.ylabel('Probability',fontsize=14,)
fig1=plt.figure(1)
plt.bar(b1,h1, width=barwidth, edgecolor=edgecolor)
#PLOT THE UNIFORM PDF
f=UnifPDF(a,b,b1)
plt.plot(b1,f,'r')

#CALCULATE THE MEAN AND STANDARD DEVIATION
mu_x=np.mean(x)
sig_x=np.std(x)

print("Mu_x: = ", mu_x)

print("Sig_x: = ", sig_x)
print("Mu_x: = ", (a+b)/2)
print("Sig_x: = ", ((b-a)**2)/12)
```

- **1.2**

```
import numpy as np
import matplotlib.pyplot as plt
n=10000
beta=0.33
x=np.random.exponential(beta, n)

def UnifPDF(beta, x):
    f=np.ones(np.size(x))
    for num in range(len(x)):
        f[num]=((1/beta)**(-(x[num]/beta)))*10
    return f

#Create bins and histogram
nbins=3 # Number of bins
edgecolor='w'  # Color separating bars in the bargraph
bins=[float(x) for x in np.linspace(beta,nbins+1)]
h1, bin_edges = np.histogram(x,bins,density=True) # Define points
on the horizontal axis
be1=bin_edges[0:np.size(bin_edges)-1]
be2=bin_edges[1:np.size(bin_edges)]
b1=(be1+be2)/2
barwidth=b1[1]-b1[0] # Width of bars in the bargraph
plt.close('all')

# PLOT THE BAR GRAPH
plt.title('Exponential Random Variable - PDF')
plt.xlabel('Exponential distribution',fontsize=14)
plt.ylabel('Probability',fontsize=14,)
fig1=plt.figure(1)
plt.bar(b1,h1, width=barwidth, edgecolor=edgecolor)

#PLOT THE UNIFORM PDF
f=UnifPDF(beta,b1)
plt.plot(b1,f,'r')

#CALCULATE THE MEAN AND STANDARD DEVIATION
mu_x=np.mean(x)
sig_x=np.std(x)

print("Mu_x: = ", mu_x)

print("Sig_x: = ", sig_x)
```

- ### 1.3

```
import math
import numpy as np
import matplotlib.pyplot as plt
mu=2.5
sigma= 0.75
n=10000

def UnifPDF(mu, sigma, x):
    f=np.ones(len(x))
    for num in range(len(x)):
        f[num]=(1/(math.sqrt(2*math.pi*sigma**2)))   * math.e  **
(-((x[num]-mu)**2)/(2*sigma**2))
    return f

x=np.random.normal(mu,sigma,n)

#Create bins and histogram
nbins=5# Number of bins
edgecolor='w'  # Color separating bars in the bargraph
bins=[float(x) for x in np.linspace(0, nbins+1)]
h1, bin_edges = np.histogram(x,bins,density=True) # Define points
on the horizontal axis
be1=bin_edges[0:np.size(bin_edges)-1]
be2=bin_edges[1:np.size(bin_edges)]
b1=(be1+be2)/2
barwidth=b1[1]-b1[0] # Width of bars in the bargraph
plt.close('all')

# PLOT THE BAR GRAPH
plt.title('Normal Random Variable - PDF')
plt.xlabel('Normal distribution',fontsize=14)
plt.ylabel('Probability',fontsize=14,)
fig1=plt.figure(1)
plt.bar(b1,h1, width=barwidth, edgecolor=edgecolor)

#PLOT THE UNIFORM PDF
f=UnifPDF(mu,sigma,b1)
plt.plot(b1,f,'r')

#CALCULATE THE MEAN AND STANDARD DEVIATION
mu_x=np.mean(x)
sig_x=np.std(x)

print("Mu_x: = ", mu_x )
print("Sig_x: = ", sig_x )
```

Problem 2:

- **Introduction**

  In Problem 2 we use the central limit theorem to calculate the mean thickness of book stacks as well as the standard deviation when limited between the boundaries of 2 and 5. Later, we increase the amount of books and repeat the process.

- **Methodology**

  We begin this experiment by calculating the theoretical Mean and STD to compare with later. This can be done with the given a and b (2 and 5) and using a quick uniform distribution generator.

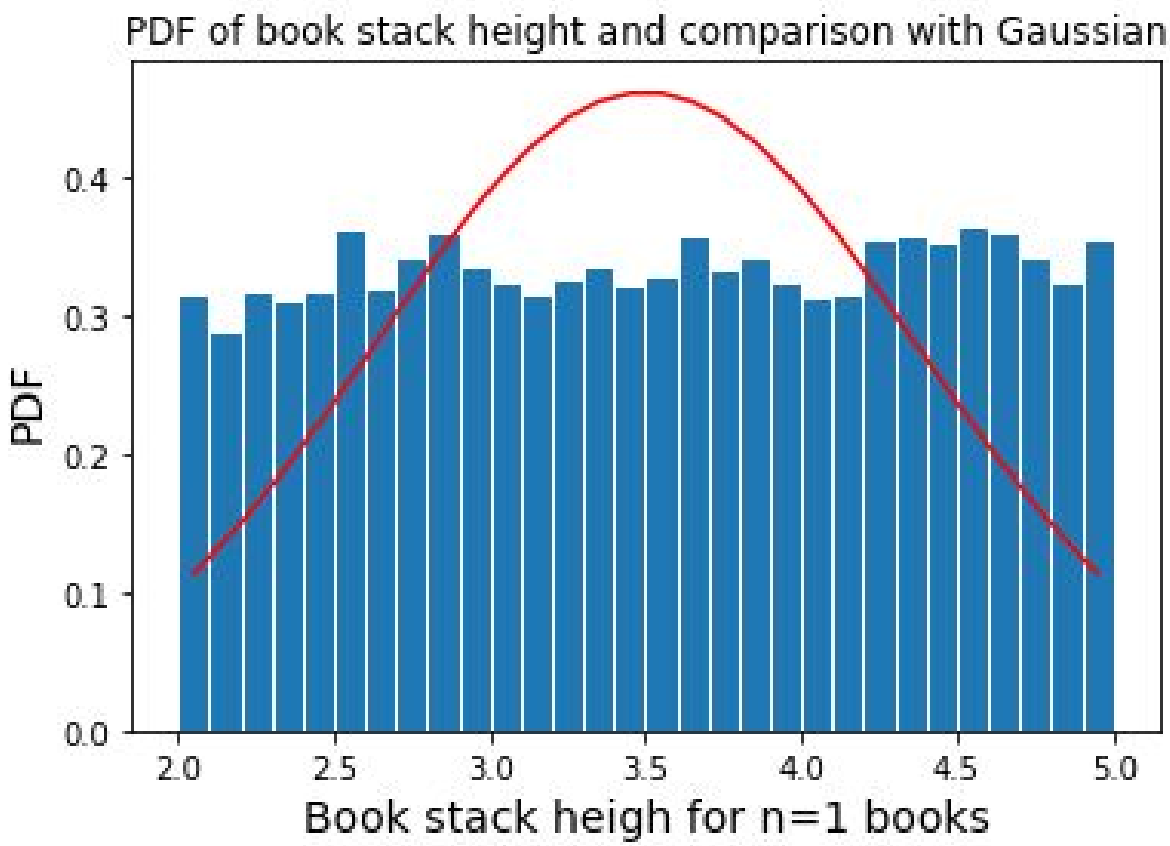
  We then run the given modified code for stacks of books of size 1, 2, and 15, graphing the results, and calculating the mean and STD for each one to be presented in a table.

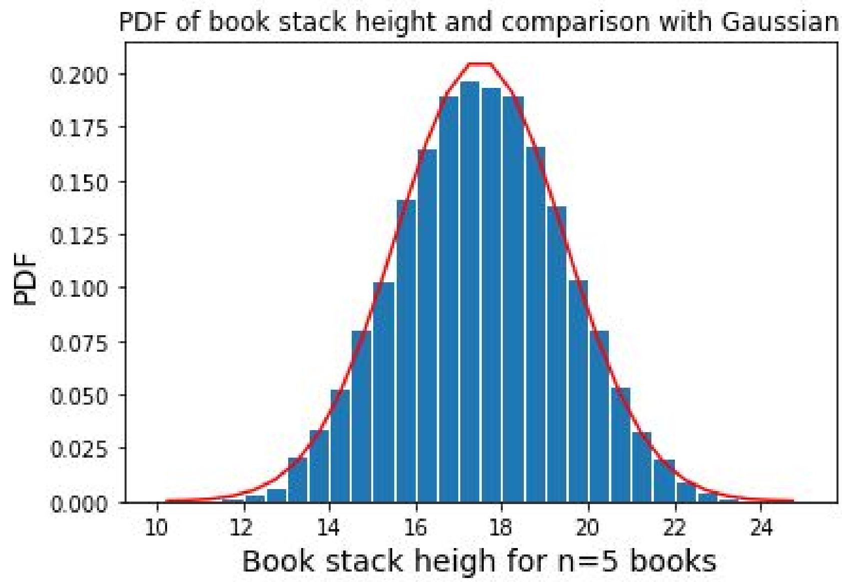
  For each bar graph, we plot a theoretical normal distribution line (in red) to compare with the probability histogram using this formula:

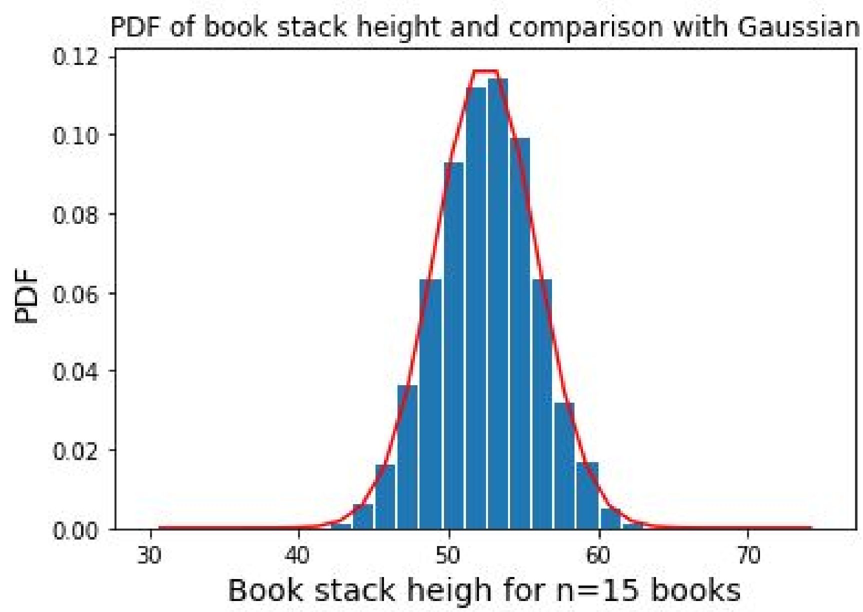
$$f(x) = \frac{1}{\sigma_S \sqrt{2\pi}} \exp\{-\frac{(x-\mu_S)^2}{2\sigma_S^2}\}$$

- **Results**

Mean thickness of a single Book (cm)	Standard deviation of thickness (cm)
~3.5005	~0.8644







Number of books n	**Mean thickness of a stack of n books (cm)**	**Standard Deviation of the thickness for n books**
n=1	~3.5245	~0.8639
n=5	~17.5095	~1.9324
n=15	~52.4779	~3.3384

- **Appendix/Source Code**
  - **Mean, STD, and n=1**

```
import numpy as np
import matplotlib.pyplot as plt


a=2.0
b=5.0
n=10000

x= np.random.uniform(a,b,n)

#CALCULATE THE MEAN AND STANDARD DEVIATION
mu_x=np.mean(x)
sig_x=np.std(x)

print("Mu_x: = ", mu_x)

print("Sig_x: = ", sig_x, '\n\n\n')


# Generate the values of the RV X
N=10000
nbooks=1

mu_x=(a+b)/2
sig_x=np.sqrt((b-a)**2/12)
X=np.zeros((N,1))

for k in range(0,N):
    x=np.random.uniform(a,b,nbooks)
    w=np.sum(x)
    X[k]=w

# Create bins and histogram
nbins=30; # Number of bins
edgecolor='w'
# Color separating bars in the bargraph #
bins=[float(x) for x in np.linspace(nbooks*a, nbooks*b,nbins+1)]
h1, bin_edges = np.histogram(X,bins,density=True)
# Define points on the horizontal axis
be1=bin_edges[0:np.size(bin_edges)-1]
be2=bin_edges[1:np.size(bin_edges)]
```

```
b1=(be1+be2)/2
barwidth=b1[1]-b1[0] # Width of bars in the bargraph

# PLOT THE BAR GRAPH

plt.close('all')
fig1=plt.figure(1)
plt.bar(b1,h1, width=barwidth, edgecolor=edgecolor)


#PLOT THE GAUSSIAN FUNCTION
def gaussian(mu,sig,z):
    f=np.exp(-(z-mu)**2/(2*sig**2))/(sig*np.sqrt(2*np.pi))
    return f

f=gaussian(mu_x*nbooks,sig_x*np.sqrt(nbooks),b1)
plt.title('PDF of book stack height and comparison with
Gaussian')
plt.xlabel('Book stack heigh for n=1 books',fontsize=14)
plt.ylabel('PDF',fontsize=14,)
plt.plot(b1,f,'r')

mu_x=np.mean(X)
sig_x=np.std(X)

print("Mu_x: = ", mu_x)

print("Sig_x: = ", sig_x)
```

- **n=5**

```
import numpy as np
import matplotlib.pyplot as plt


a=2.0
b=5.0

# Generate the values of the RV X
N=10000
nbooks=5

mu_x=(a+b)/2
sig_x=np.sqrt((b-a)**2/12)
X=np.zeros((N,1))

for k in range(0,N):
    x=np.random.uniform(a,b,nbooks)
    w=np.sum(x)
    X[k]=w

# Create bins and histogram
nbins=30; # Number of bins
edgecolor='w'
# Color separating bars in the bargraph #
bins=[float(x) for x in np.linspace(nbooks*a, nbooks*b,nbins+1)]
h1, bin_edges = np.histogram(X,bins,density=True)
# Define points on the horizontal axis
be1=bin_edges[0:np.size(bin_edges)-1]
be2=bin_edges[1:np.size(bin_edges)]
b1=(be1+be2)/2
barwidth=b1[1]-b1[0] # Width of bars in the bargraph

# PLOT THE BAR GRAPH
plt.close('all')
fig1=plt.figure(1)
plt.bar(b1,h1, width=barwidth, edgecolor=edgecolor)


#PLOT THE GAUSSIAN FUNCTION
def gaussian(mu,sig,z):
    f=np.exp(-(z-mu)**2/(2*sig**2))/(sig*np.sqrt(2*np.pi))
    return f

plt.title('PDF of book stack height and comparison with
Gaussian')
```

```
plt.xlabel('Book stack heigh for n=5 books',fontsize=14)
plt.ylabel('PDF',fontsize=14,)
f=gaussian(mu_x*nbooks,sig_x*np.sqrt(nbooks),b1)
plt.plot(b1,f,'r')

mu_x=np.mean(X)
sig_x=np.std(X)

print("Mu_x: = ", mu_x)

print("Sig_x: = ", sig_x)
```

- **n=15**

```
import numpy as np
import matplotlib.pyplot as plt


a=2.0
b=5.0

# Generate the values of the RV X
N=10000
nbooks=15

mu_x=(a+b)/2
sig_x=np.sqrt((b-a)**2/12)
X=np.zeros((N,1))

for k in range(0,N):
    x=np.random.uniform(a,b,nbooks)
    w=np.sum(x)
    X[k]=w

# Create bins and histogram
nbins=30; # Number of bins
edgecolor='w'
# Color separating bars in the bargraph #
bins=[float(x) for x in np.linspace(nbooks*a, nbooks*b,nbins+1)]
h1, bin_edges = np.histogram(X,bins,density=True)
# Define points on the horizontal axis
be1=bin_edges[0:np.size(bin_edges)-1]
be2=bin_edges[1:np.size(bin_edges)]
b1=(be1+be2)/2
barwidth=b1[1]-b1[0] # Width of bars in the bargraph

# PLOT THE BAR GRAPH
plt.close('all')
fig1=plt.figure(1)
plt.bar(b1,h1, width=barwidth, edgecolor=edgecolor)


#PLOT THE GAUSSIAN FUNCTION
def gaussian(mu,sig,z):
    f=np.exp(-(z-mu)**2/(2*sig**2))/(sig*np.sqrt(2*np.pi))
    return f

plt.title('PDF of book stack height and comparison with
Gaussian')
```

```
plt.xlabel('Book stack heigh for n=15 books',fontsize=14)
plt.ylabel('PDF',fontsize=14,)
f=gaussian(mu_x*nbooks,sig_x*np.sqrt(nbooks),b1)
plt.plot(b1,f,'r')

mu_x=np.mean(X)
sig_x=np.std(X)

print("Mu_x: = ", mu_x)

print("Sig_x: = ", sig_x)
```

Problem 3:

- **Introduction**
  In problem 3 we will take what we've learned from the previous problems and apply them to represent battery life. We will do so by generating a PDF of a single battery's life, as well as making a CDF of carton life.
  With these graphs we can answer questions about the lifespan probabilities of these batteries.

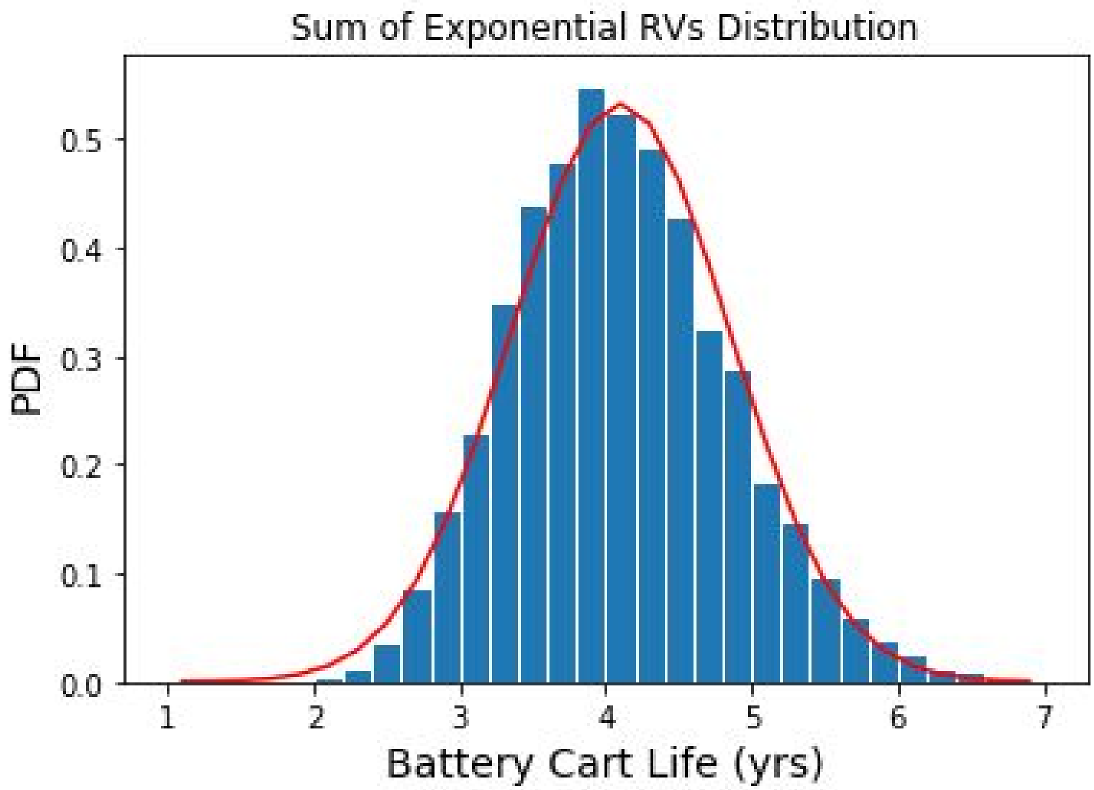
- **Methodology**
  To generate the PDF, we generate an exponential distribution of the given beta (50) into an array of the given n (30). This is done 10,000 and the sum of each array is added to each index of another array. Plotting this array as a bar graph gives us an accurate PDF. To redline a theoretical calculation over it, we use the Gaussian Distribution formula:

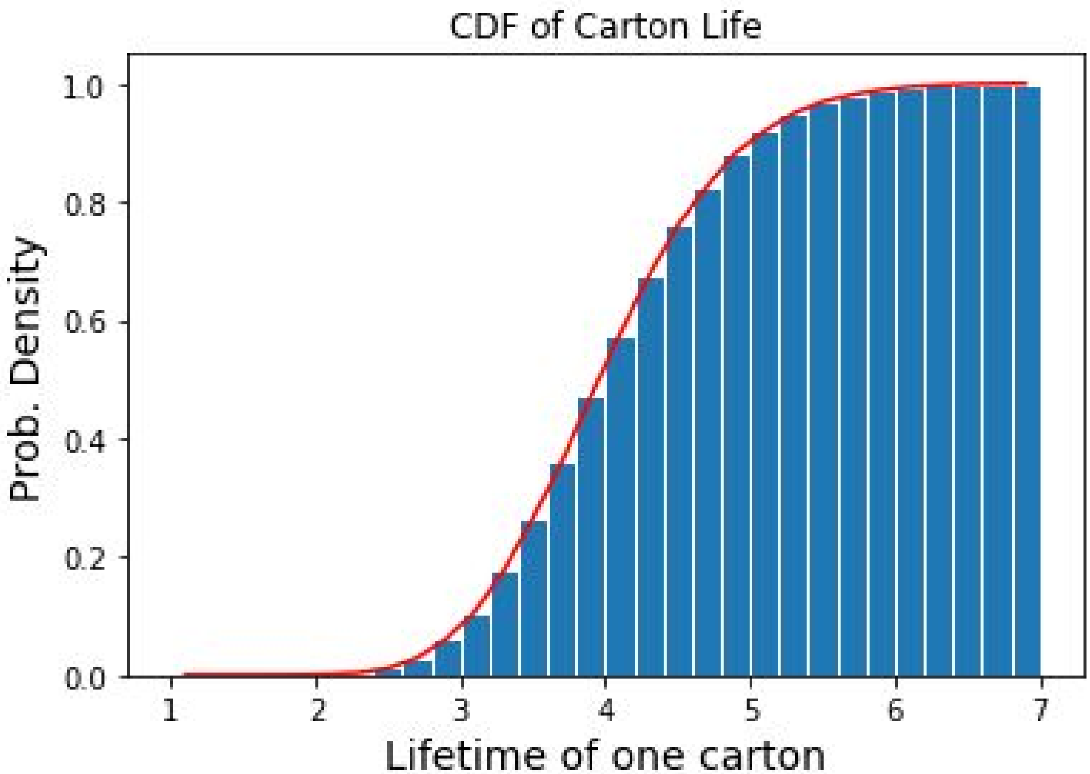
$$f_T(t\,;\beta) = \begin{cases} \frac{1}{\beta}\exp(-\frac{1}{\beta}t), & t \geq 0 \\ 0, & t < 0 \end{cases}$$

  After we have this, we generate the Cumulative distribution function, which will help us answer further probability questions about battery life.

- **Results**

Question	Ans.
1. Probability the carton will last longer that Y1 (3) years	P = ~**0.9**
2. Probability that the carton will last between Y2 (2) and Y3 (4) years	P = **~0.5**




CDF of Carton Life
Prob. Density
Lifetime of one carton
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
5
6
7

- **Appendix/Source Code**

```
import numpy as np
import matplotlib.pyplot as plt


beta =50 #days
n=30 #batteries
Y1=3 #yrs
Y2=2 #yrs
Y3=4 #yrs;
N=10000


# Generate the values of the RV X
X=np.zeros((N,1))

for k in range(0,N):
    x=np.random.exponential(beta,n)
    X[k]= np.sum(x) /365

# Create bins and histogram
nbins=30        # Number of bins
edgecolor='w'       # Color separating bars in the bargraph #



bins=[float(y) for y in np.linspace(1,7,nbins+1)]
h1, bin_edges = np.histogram(X,bins,density=True)


# Define points on the horizontal axis
be1=bin_edges[0:np.size(bin_edges)-1]
be2=bin_edges[1:np.size(bin_edges)]
b1=(be1+be2)/2
barwidth=b1[1]-b1[0] # Width of bars in the bargraph

# PLOT THE BAR GRAPH

plt.close('all')
fig1=plt.figure(1)
plt.bar(b1,h1, width=barwidth, edgecolor=edgecolor)
plt.title('Sum of Exponential RVs Distribution')
plt.xlabel('Battery Cart Life (yrs)',fontsize=14)
plt.ylabel('PDF',fontsize=14,)
```

```
mu_x=np.mean(X)
sig_x=np.std(X)

#PLOT THE GAUSSIAN FUNCTION
def gaussian(mu,sig,z):
    f=np.exp(-(z-mu)**2/(2*sig**2))/(sig*np.sqrt(2*np.pi))
    return f

f = gaussian(mu_x, sig_x, b1)
plt.plot(b1, f, 'r')
plt.show()

print("Mu_x: = ", mu_x)

print("Sig_x: = ", sig_x)

CDF = np.cumsum(h1 * barwidth)
plt.bar(b1, CDF, width = barwidth, edgecolor = edgecolor)
plt.title('CDF of Carton Life')
plt.xlabel('Lifetime of one carton',fontsize=14)
plt.ylabel('Prob. Density',fontsize=14,)
#plt.grid(True, color='#dfdfdf', dashes=(1,2))
#plt.yticks(np.arange(0,1.1,step=.1))


plt.plot (b1, CDF, 'r')
plt.show()
```