Kevin Long

Professor Chassiakos

EE 381

8 April 2019

Project 4 – Central Limit Theorem: Simulating Continuous Random Variables with Various Distributions

1. Simulate Continuous Random Variables with Selected Distributions

1.1 Simulate a Uniform Random Variable

a. Introduction

The PDF of a random variable uniformly distributed in $[a,b)$ is defined as following:

$$f(x)=\begin{cases}\frac{1}{(b-a)}, & a\le x\le b\\ 0, & \text{otherwise}\end{cases} \quad ; \quad \text{and} \quad P(X\le x)=F(x)=\begin{cases}0, & x<a\\ \frac{(x-a)}{(b-a)}, & a\le x<b\\ 1, & x\ge b\end{cases}$$

It is noted that the mean and variance of a uniformly distributed random variable X are given by:

$$\mathrm{E}(X)=\mu_X=\frac{a+b}{2} \quad ; \quad \mathrm{Var}(X)=\sigma_X^2=\frac{(b-a)^2}{12}$$

b. Methodology

A random variable *X* is created with a uniform distribution. The function "np.random.uniform(a, b, n)" is called to generate *n* values of the random variable *X* with uniform probability distribution in the open interval [a,b). The values used for parameters are: a = 1.0, b = 4.0, n = 10000.

The expectation and standard deviation of the random variable *X* is done via the "np.mean()" and "np.std()" functions. These values are compared to the theoretical values given by:

$$\mu_X=\frac{a+b}{2} \quad ; \quad \sigma_X^2=\frac{(b-a)^2}{12}$$

The histogram function from matplotlib.pyplot is used to plot a bar graph of the experimental values of the random variable *X*. On the same graph is the probability density function for the random variable as given by:

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

The graph given by the above function is compared to the bar graph plot. The above function is translated into code in the function "unifPDF()."

c. **Result(s) and Conclusion(s)**

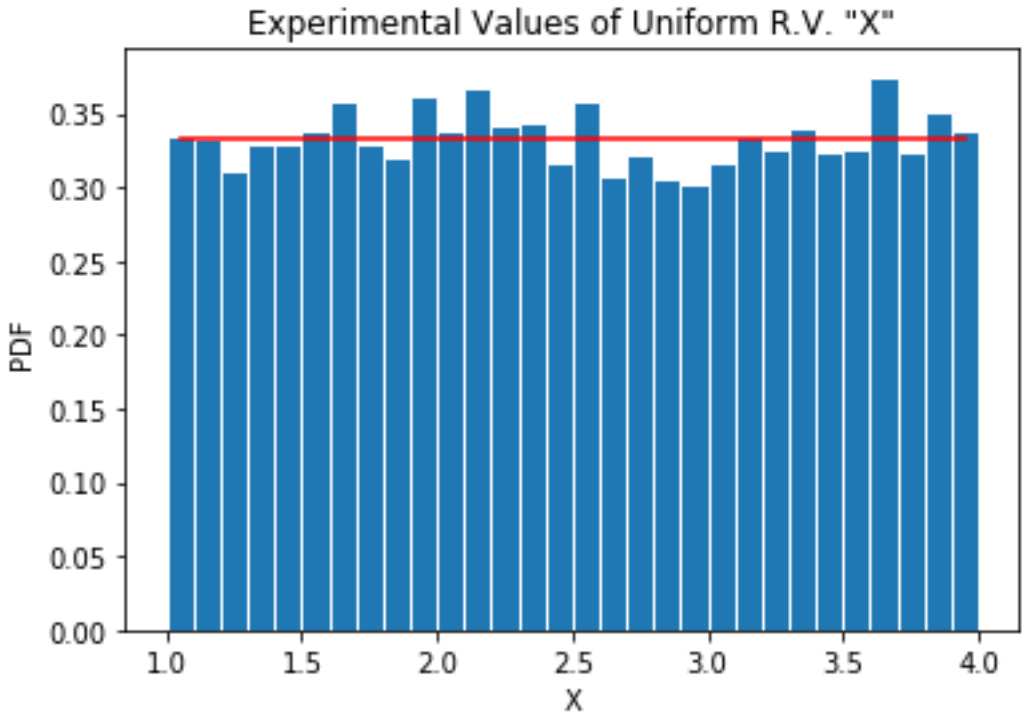

Figure 1. Bar Graph of Uniform Probability Distribution

Table 1: Statistics for a Uniform Distribution			
Expectation		**Standard Deviation**	
Theoretical Calculation	Experimental Measurement	Theoretical Calculation	Experimental Measurement
2.5	**2.501488023239**	**0.75**	**0.8628900647665687**

d. **Source Code**

```
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.  #1.1
5.  def uniRV(a, b, n):
6.      X = np.random.uniform(a, b, n)
7.
8.      # Create bins and histogram
9.      nbins = 30; # Number of bins
10.     edgecolor = 'w'; # Color separating bars in the bargraph
11.     bins = [float(X) for X in np.linspace(a, b, nbins+1)]
12.     h1, bin_edges = np.histogram(X, bins, density=True)
13.
14.     # Define points on the horizontal axis
15.     be1 = bin_edges[0:np.size(bin_edges)-1]
16.     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
    fig1 = plt.figure(1)
    plt.bar(b1, h1, width=barwidth, edgecolor=edgecolor)
    plt.title('GRAPH TITLE') #PROPER LABELS
    plt.xlabel('X')
    plt.ylabel('PDF')

    #PLOT THE UNIFORM PDF
    def unifPDF(a, b, X):
        f = (1/abs(b-a))*np.ones(np.size(X))
        return f

    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('Mean:', mu_x)
    print('Standard deviation:', sig_x)

if __name__ == '__main__':
    a = 1.0
    b = 4.0
    n = 10000
    beta = 40
    mu = 2.5
    sigma = 0.75

    uniRV(a, b, n)
```

1.2 Simulate a Uniform Random Variable

a. **Introduction**

The PDF of a random variable exponentially distributed is defined as following:

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

From the above definition, the CDF of T is found as:

$$P(T \leq t) = F(t) = \begin{cases} 0, & t < 0 \\ 1 - \exp(-\frac{1}{\beta}t), & t \geq 0 \end{cases}$$

It is noted that the mean and standard deviation of the exponentially distributed random variable T are given by:

$$\mu_T = \beta \quad ; \quad \sigma_T = \beta$$

b. **Methodology**

A random variable *T* is created with exponential distribution. The function "np.random.exponential(beta, n)" is used to generate *n* values of the random variable *T* with exponential probability distribution. The values are: beta = 40, n = 10000.

The expectation and standard deviation of the random variable *X* is done by the functions "np.mean()" and "np.std." The values are compared to the theoretical values given by:

$$\mu_T = \beta \quad ; \quad \sigma_T = \beta$$

Where β = 40.

The histogram function from matplotlib.pyplot is used to plot a bar graph of the experimental values of the random variable *T*. On the same graph is the probability density function for the random variable as given by:

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

The graph given by the above function is compared to the bar graph plot. The above function is translated into code in the function "expPDF()."

c. **Result(s) and Conclusion(s)**

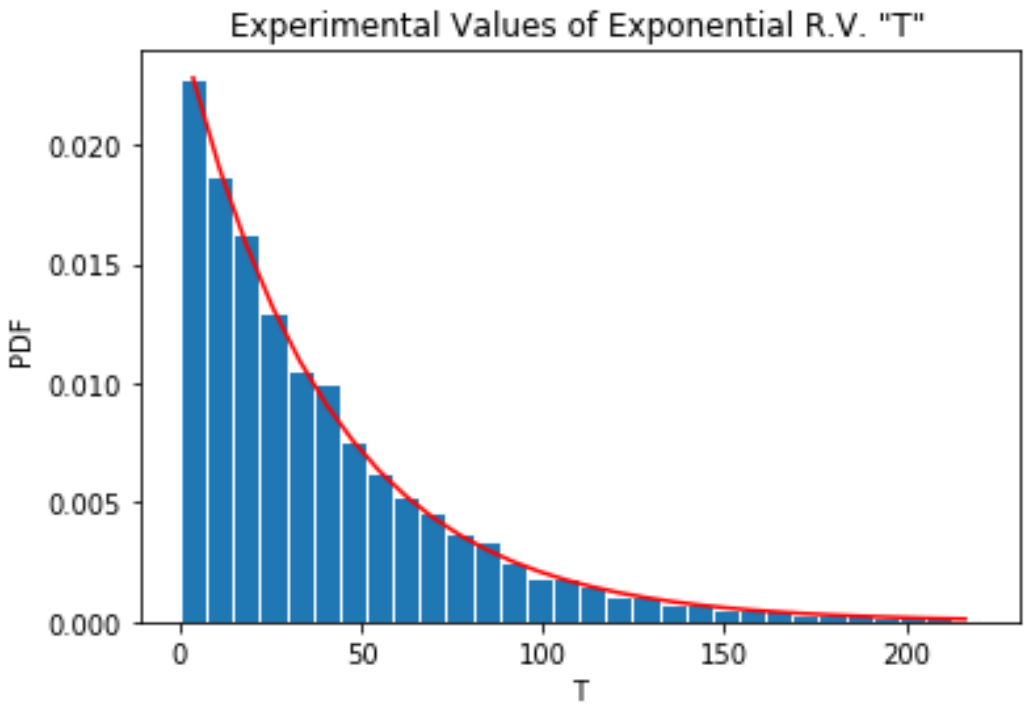

Figure 2. Bar Graph of Exponential Probability Distribution

Table 2: Statistics for an Exponential Distribution			
Expectation		**Standard Deviation**	
Theoretical Calculation	Experimental Measurement	Theoretical Calculation	Experimental Measurement
40	**40.14455120207611**	**40**	**40.559484795753015**

d. **Source Code**

```
52. import numpy as np
53. import matplotlib.pyplot as plt
54. #1.2
55. def expRV(beta, n):
56.     T = np.random.exponential(beta, n)
57.
58.     # Create bins and histogram
59.     nbins = 30; # Number of bins
60.     edgecolor = 'w'; # Color separating bars in the bargraph
61.     bins = [float(T) for T in np.linspace(0, 220, nbins+1)]
62.     h1, bin_edges = np.histogram(T, bins, density=True)
63.
64.     # Define points on the horizontal axis
65.     be1 = bin_edges[0:np.size(bin_edges)-1]
66.     be2 = bin_edges[1:np.size(bin_edges)]
67.     b1 = (be1+be2)/2
68.     barwidth = b1[1]-b1[0] # Width of bars in the bargraph
```

```
    plt.close('all')

    # PLOT THE BAR GRAPH
    fig1 = plt.figure(1)
    plt.bar(b1, h1, width=barwidth, edgecolor=edgecolor)
    plt.title('GRAPH TITLE 2') #PROPER LABELS
    plt.xlabel('T')
    plt.ylabel('PDF')

    #PLOT THE UNIFORM PDF
    def expPDF(beta, T):
        f = ((1/beta)*np.exp(-(1/beta)*T))*np.ones(np.size(T))
        return f

    f = expPDF(beta, b1)
    plt.plot(b1, f, 'r')

    #CALCULATE THE MEAN AND STANDARD DEVIATION
    mu_t = np.mean(T)
    sig_t = np.std(T)

    print('Mean:', mu_t)
    print('Standard deviation:', sig_t)

if __name__ == '__main__':
    a = 1.0
    b = 4.0
    n = 10000
    beta = 40
    mu = 2.5
    sigma = 0.75
        expRV(beta, n)
```

1.3 Simulate a Normal Random Variable

a. **Introduction**

The PDF of a normal random variable X with mean μ_X and standard deviation σ_X is defined as following:

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

It is noted that the mean and variance of the normally distributed random variable X are given by:

$$E(X) = \mu_X \quad ; \quad \mathrm{Var}(X) = \sigma_X^2$$

b. **Methodology**

A random variable *X* is created with normal distribution. The function "np.random.normal(mu, sigma, n)" is used to generate *n* values of the random variable *X* with normal probability distribution. The values are: mu = 2.5, sigma = 0.75, n = 10000.

The expectation and standard deviation of the random variable *X* is done by the functions "np.mean()" and "np.std." The values are compared to the theoretical values given by:

$$\mu_X = \mu \quad ; \quad \sigma_X = \sigma$$

The values are listed previously above.

The histogram function from matplotlib.pyplot is used to plot a bar graph of the experimental values of the random variable *X*. On the same graph is the probability density function for the random variable as given by:

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

The graph given by the above function is compared to the bar graph plot. The above function is translated into code in the function "unifPDF()."

c. **Result(s) and Conclusion(s)**

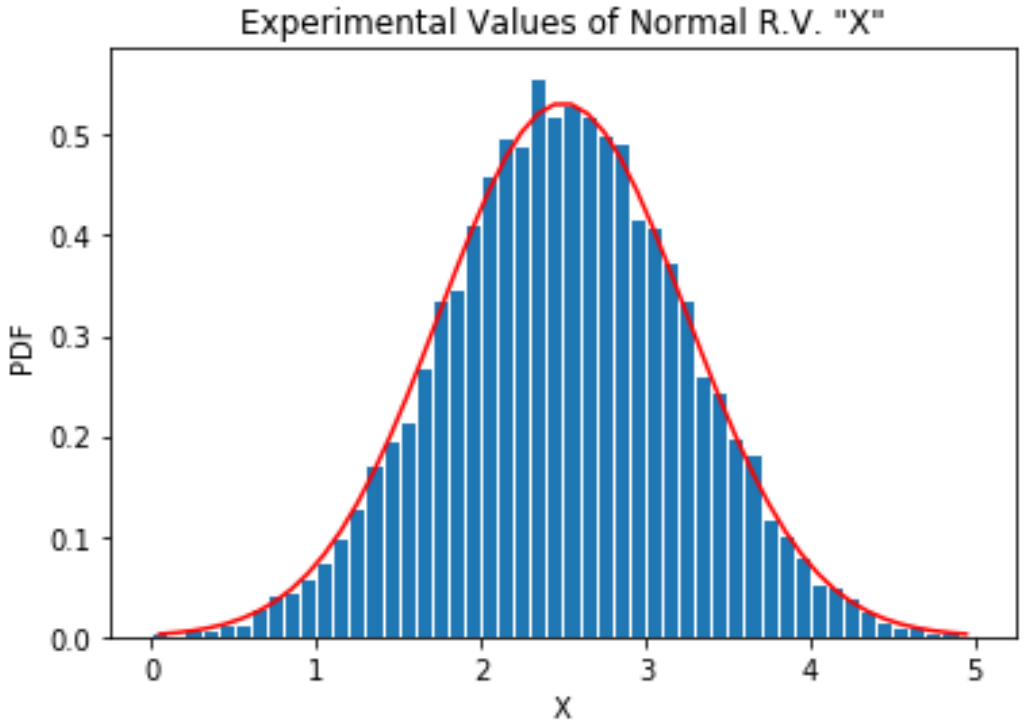

Figure 3. Bar Graph of Normal Probability Distribution

Table 3: Statistics for a Normal Distribution

Expectation		Standard Deviation	
Theoretical Calculation	Experimental Measurement	Theoretical Calculation	Experimental Measurement
2.5	**2.5123353912744584**	**0.75**	**0.7539313082945683**

d. **Source Code**

```
import numpy as np
import matplotlib.pyplot as plt
import math
#1.3
def normRV(mu, sigma, n):
    X = np.random.normal(mu, sigma, n)

    # Create bins and histogram
    nbins = 50; # Number of bins
    edgecolor = 'w'; # Color separating bars in the bargraph
    bins = [float(X) for X in np.linspace(0, 5, 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
    fig1 = plt.figure(1)
    plt.bar(b1, h1, width=barwidth, edgecolor=edgecolor)
    plt.title('GRAPH TITLE') #PROPER LABELS
    plt.xlabel('X')
    plt.ylabel('PDF')

    #PLOT THE UNIFORM PDF
    def unifPDF(mu, sigma, X):
        f = ( 1/(sigma*math.sqrt(2*math.pi) ) )*np.exp( -( (X-mu)**2)/(2*(sigma)**2) )*np.ones(np.size(X) )
        return f

    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)

```

```
140.        print('Mean:', mu_x)
141.        print('Standard deviation:', sig_x)
142.
143.    if __name__ == '__main__':
144.        a = 1.0
145.        b = 4.0
146.        n = 10000
147.        beta = 40
148.        mu = 2.5
149.        sigma = 0.75
150.        normRV(mu, sigma, n)
```

2. Central Limit Theorem

a. Introduction

If $X_1, X_2, \ldots X_n$ are independent random variables having the same probability distribution with mean μ and standard deviation σ, consider the sum $S_n = X_1 + X_2 + \ldots X_n$.

This sum S_n is a random variable with mean $\mu_{S_n} = n\mu$ and standard deviation $\sigma_{S_n} = \sigma\sqrt{n}$.

The Central Limit Theorem states that as $n \to \infty$ the probability distribution of the R.V. S_n will approach a normal distribution with mean μ_{S_n} and standard deviation σ_{S_n}, *regardless of the original distribution* of the R.V. $X_1, X_2, \ldots X_n$.

It is noted that the PDF of the normally distributed R.V. S_n is given by:

$$f(s_n) = \frac{1}{\sigma_{S_n}\sqrt{2\pi}} \exp\{-\frac{(x-\mu_{S_n})^2}{2\sigma_{S_n}^2}\}$$

Imagine that there is a collection of books, each of which has thickness *W*. The thickness *W* is a random variable, uniformly distributed between a minimum of *a* and a maximum of *b* centimeters. The values of *a* and *b* are respectively 1.0 and 4.0.

The books are piled in stacks of *n* = 1, 5, 10, or 15 books. The width *Sn* of a stack of *n* books is a random variable (the sum of the widths of the *n* books). The random variable has a mean:

$\mu_{S_n} = n\mu_w$ and a standard deviation of $\sigma_{S_n} = \sigma_w\sqrt{n}$. The mean and the standard deviation of the stacked books are calculated for the different values of *n* = 1, 5, 10, or 15.

b. Methodology

A random variable *X* is created with normal distribution. The function "np.random.uniform(a, b, n)" is used to generate *n* values of the random variable

X with normal probability distribution. The values are: a = 2.5, b = 0.75, n = 1, 5, 10, and 15. 10,000 experiments are run.

The following values are calculated as such: rho_w = m.sqrt(((b-a)**2) / 12), rho_s = rho_w*m.sqrt(n), mu_w = (a+b)/2, mu_s = n*mu_w. μ_x and σ_x of the random variable X is calculated by the functions "np.mean()" and "np.std."

The histogram function from matplotlib.pyplot is used to plot a probability histogram of the random variable *X*. On the same graph is the normal probability function. The probability histogram is compared to the plot of the following function:

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

The above function is translated into code in the function "gaussian()."

c. **Result(s) and Conclusion(s)**

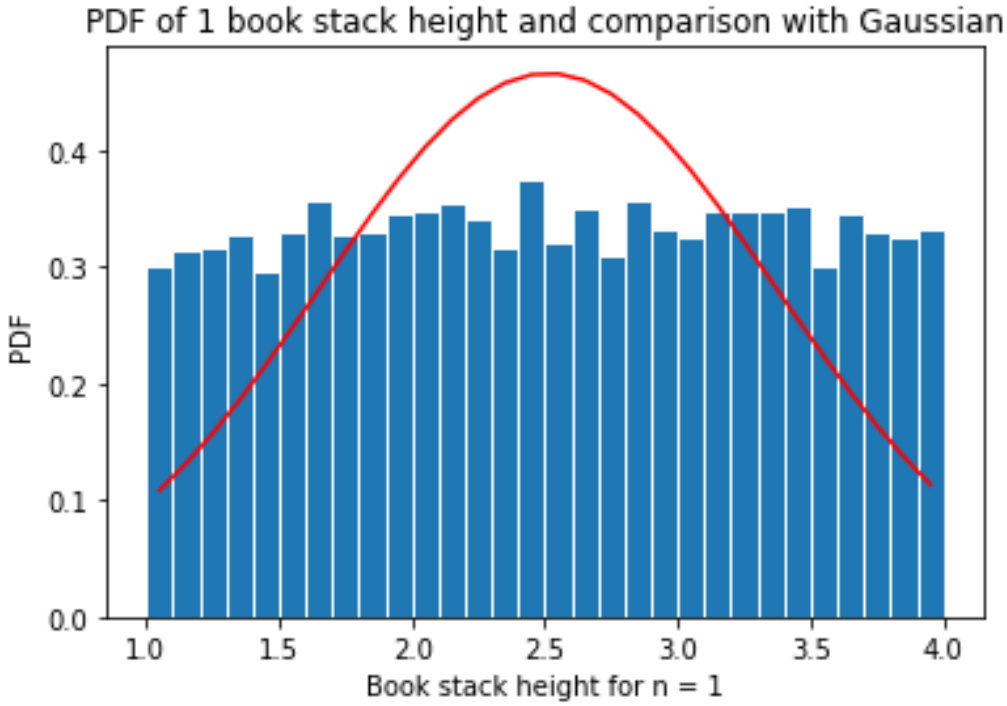

Figure 4. 1 Book Stack

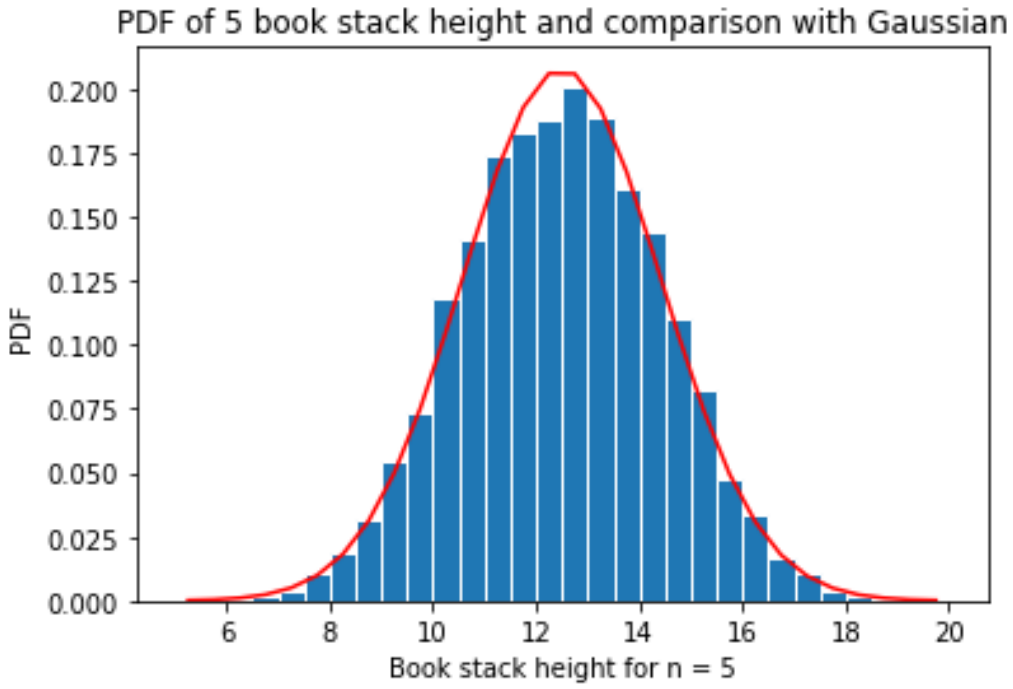

Figure 5. 5 Book Stack

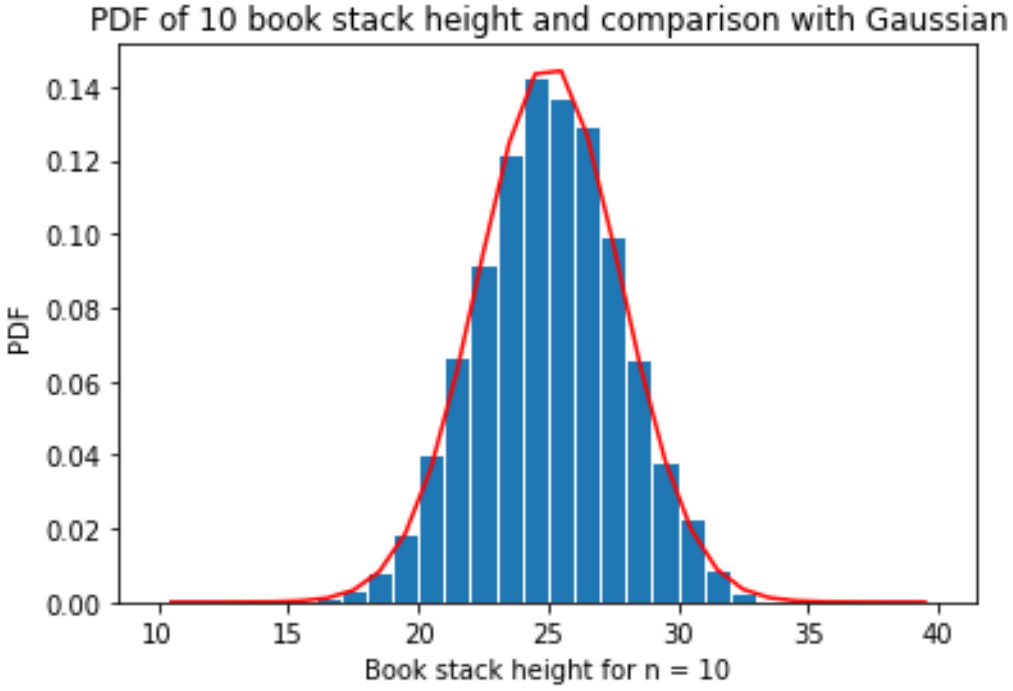

Figure 6. 10 Book Stack

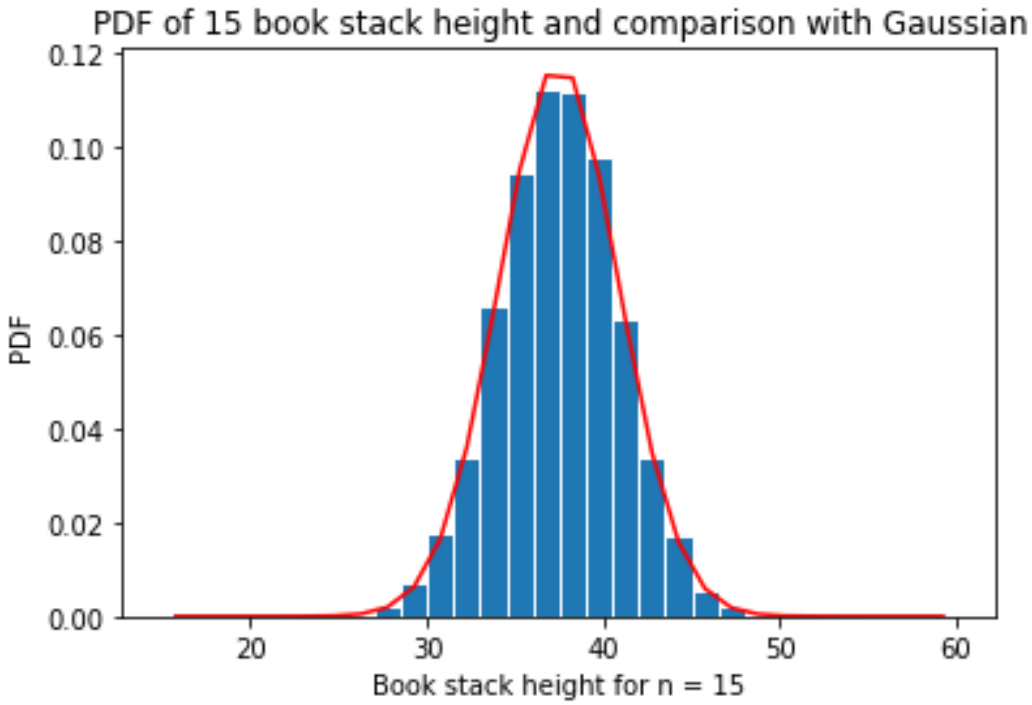

Figure 7. 15 Book Stack

Table 4: Mean Thickness & Standard Deviation of Thickness	
Mean thickness of a single book (cm)	Standard deviation of thickness (cm)
μ_w = 2.4985222591247767	σ_w = 0.866895848906263

Table 5: Statistics for Central Limit Theorem		
Numbers of books *n*	Mean thickness of a stack of *n* books (cm)	Standard deviation of the thickness for *n* books
n = 1	μ_w = 2.4985222591247767	σ_w = 0.866895848906263
n = 5	μ_w = 12.487806461551116	σ_w = 1.9354575109705072
n = 15	μ_w = 37.52411603501356	σ_w = 3.367934302183977

d. **Source Code**

```
21 import numpy as np
22 import matplotlib.pyplot as plt
23 import math as m
24
25 def centralLimit(a, b, n):
26     X = [None] * 10000
27     for i in range(0, 10000):
28         w = np.random.uniform(a, b, n)
29         X[i] = np.sum(w)
```

```
30
31  #Calculations
32  rho_w = m.sqrt(((b-a)**2) / 12)
33  print("rho_w:", rho_w)
34
35  rho_s = rho_w*m.sqrt(n)
36  print("rho_s:", rho_s)
37
38  mu_w = (a+b)/2
39  print('mu_w:', mu_w)
40
41  mu_s = n*mu_w
42  print('mu_s:', mu_s)
43
44  mu_x=np.mean(X)
45  sig_x=np.std(X)
46
47  print('mean:', mu_x)
48  print('std:', sig_x)
49
50  # Create bins and histogram
51  nbins = 30  # Number of bins
52  edgecolor = 'w' # Color separating bars in the bargraph
53  bins = [float (x) for x in np.linspace(n*a,n*b,nbins+1)]
54  h1, bin_edges = np.histogram(X,bins,density=True)
55
56  be1 = bin_edges[0:np.size(bin_edges)-1]
57  be2 = bin_edges[1:np.size(bin_edges)]
58  b1 = (be1+be2)/2
59  barwidth = b1[1]-b1[0] # Width of bars in the bargraph
60  plt.close('all')
61
62  # PLOT THE BAR GRAPH
63  fig1 = plt.figure(1)
64  plt.bar(b1,h1, width=barwidth, edgecolor = edgecolor)
65  label = 'Book stack height for n = %d' % n
66  plt.title('PDF of %d book stack height and comparison with Gaussian' % n)
67  plt.xlabel(label)
68  plt.ylabel('PDF')
69
70  def gaussian(mu,sig,z):
71      f=np.exp(-(z-mu_x)**2/(2*sig_x**2))/(sig_x*np.sqrt(2*np.pi))
72      return f
73  f=gaussian(mu_x*n, sig_x*np.sqrt(n),b1)
74  plt.plot(b1,f,'r')
75
```

```
76 if __name__ == '__main__':
77     a = 1
78     b = 4
79     #centralLimit(a, b, 1)
80     #centralLimit(a, b, 2)
81     #centralLimit(a, b, 5)
82     #centralLimit(a, b, 10)
83     centralLimit(a, b, 15)
```

3. Distribution of the Sum of Exponential Random Variables

a. Introduction

Consider a problem that involved a battery-operated medical monitor. The lifetime (T) of the battery is a random variable with an exponentially distributed lifetime. A battery lasts an average of β, which is 40 days. Under such conditions, the PDF of the battery lifetime is given by:

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

As mentioned before, the mean of the variance of the random variable T are:

$$\mu_T = \beta \quad ; \quad \sigma_T = \beta$$

When a battery fails, it is replaced immediately by a new one. Batteries are purchased in a carton of 24 batteries.

1) Find the probability that the carton will last longer than three years, i.e. $P(S > 3 * 365) = 1 - P(S \leq 3 * 365) = 1 - F(1095)$.

2) Find the probability that the carton will last between 2.0 and 2.5 years (i.e. between 730 and 912 days): $P(730 \leq S \leq 912) = F(912) - F(730)$.

b. Methodology

A vector representing the 24 batteries are created and is named as “carton.” For each experiment ran, the batteries are calculated via the function “np.random.exponential().” The value is temporarily stored into the variable “t” and then summed via the function “np.sum().” This value is stored into the variable “carton” in the sequential order. The variables passed into the function are defined and initialized to the following: a = 300, b = 2000, N = 10,000, numBatteries = 24, beta = 40.

The following variables are then calculated as such: mu_t = beta, mu_c = numBatteries * beta, sig_t = beta, sig_c = beta * np.sqrt(numBatteries), rho_t = beta.

Finally, the variables are graphed via the matplotlib.pyplot library.

c. **Result(s) and Conclusion(s)**

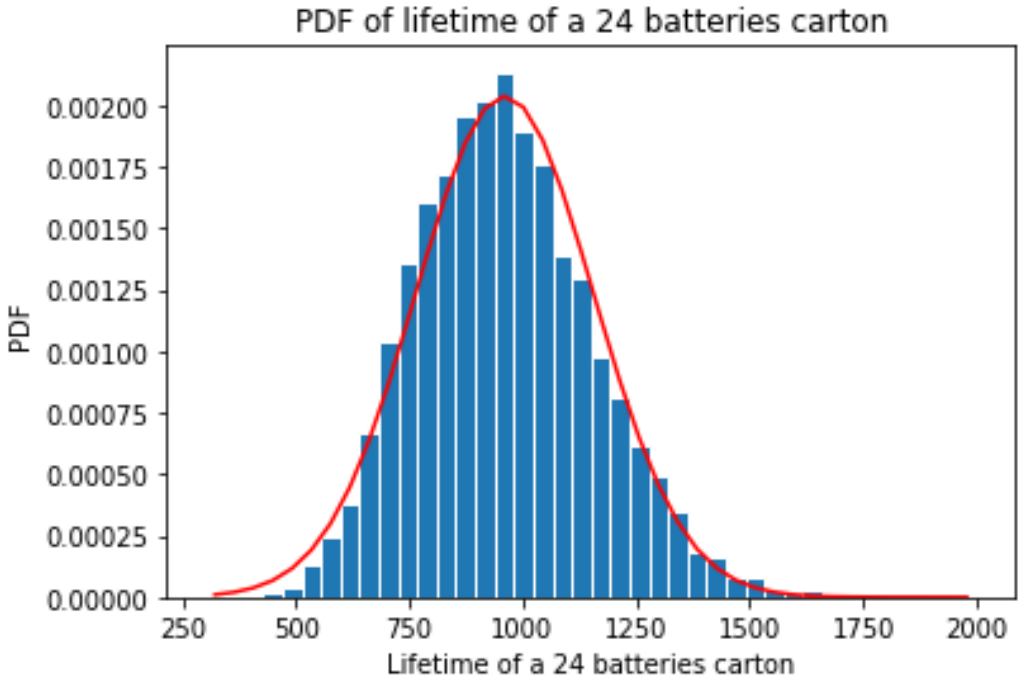

Figure 8. Lifetime PDF

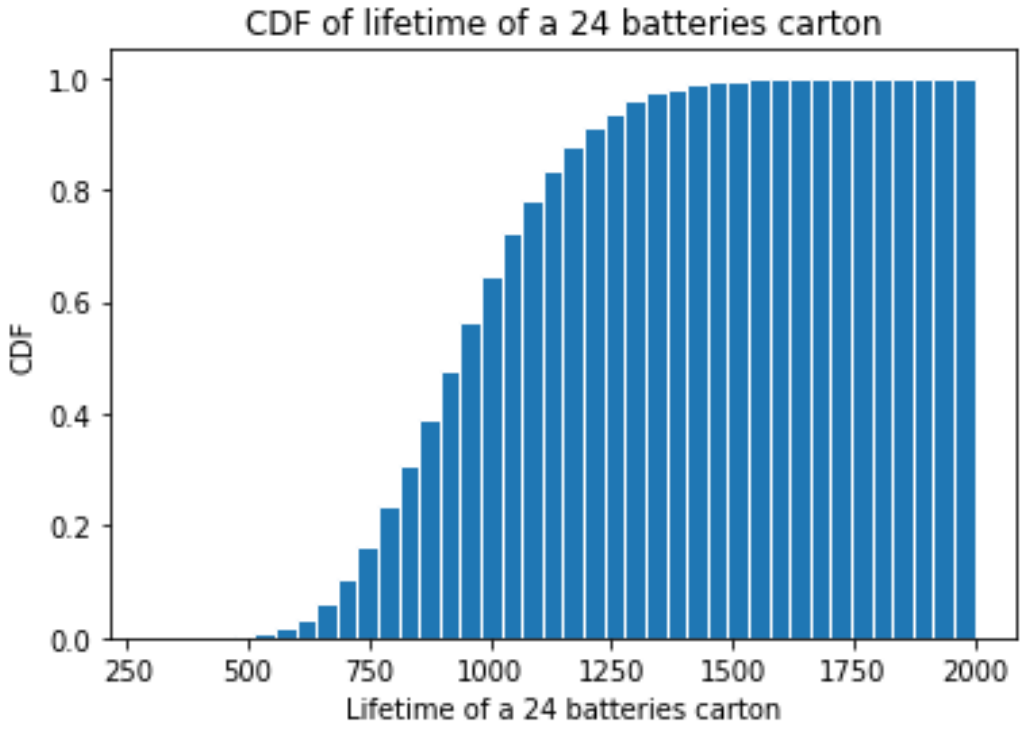

Figure 9. Lifetime CDF

Table 6:	
QUESTION	**ANS.**
1. Probability that the carton will last longer than three years	**0.213**
2. Probability that the carton will last between 2.0 and 2.5 years	**0.3326**

d. **Source Code**

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

def exponentialRV(beta):
    a = 300
    b = 2000
    N = 10000
    numBatteries = 24

    carton = np.zeros((N,1))
    for i in range(0, N):
        t = np.random.exponential(beta, numBatteries)
        carton[i] = np.sum(t)

    mu_t = beta
    mu_c = numBatteries * beta
    sig_t = beta
    sig_c = beta * np.sqrt(numBatteries)
    rho_t = beta

    nbins=40; # 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(carton,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
    fig1=plt.figure(1)
    plt.bar(b1,h1, width=barwidth, edgecolor=edgecolor)

    plt.title('PDF of lifetime of a 24 batteries carton')
    plt.xlabel('Lifetime of a 24 batteries carton')
    plt.ylabel('PDF')
```

```
43.
44.     def gaussian(mu_x,sig_x,z):
45.         f = np.exp(-(z-mu_x)**2/(2 * sig_x**2))/(sig_x * np.sqrt(2*np.pi))
46.         return f
47.     f=gaussian(mu_c,sig_c,b1)
48.     plt.plot(b1,f,'r')
49.
50.     fig2 = plt.figure(2)
51.     h2 = np.cumsum(h1)*barwidth
52.     plt.bar(b1,h2, width=barwidth, edgecolor = edgecolor)
53.     plt.title('CDF of lifetime of a 24 batteries carton')
54.     plt.xlabel('Lifetime of a 24 batteries carton')
55.     plt.ylabel('CDF')
56.
57. if __name__ == '__main__':
58.     beta = 40
59.     exponentialRV(beta)
```