PROJECT 4 – CENTRAL LIMIIT THEOREM

Simulating Continuous Random Variables with Various Distributions

Abstract

This project focuses on the simulation of continuous random variables with different experiments.

Brian Nguyen

015188925

1. Simulate continuous random variables with selected distributions

Introduction

In this section, a continuous random variable is simulated with different means of distributions. These types of distributions are uniform, exponential, and normal.

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

$$f(x) = \begin{cases} \dfrac{1}{b-a}, & a \leq x \leq b \\ 0, & otherwise \end{cases} \; ; \; and \; P(X \leq x) = F(x) = \begin{cases} 0, & x < a \\ \dfrac{x-a}{b-a}, & a \leq x < b \\ 1, & x \geq b \end{cases}$$

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

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

The PDF of a normal random variable *X* with mean μ_x and standard deviation σ_x is defined as the following:

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

We are to calculate the theoretical calculation and the experimental measurement of the expectation and the standard deviation.

Methodology

To create a random variable *X*, I used specific Python functions.

- Uniform: `np.random.uniform(a, b, n)`
  - a = 1, b = 4, n = 10000
- Exponential: `np.random.exponential(beta, n)`
  - beta = 40, n = 10000
- Normal: `np.random.normal(mu, sigma, n)`
  - mu = 2.5, sigma = .75, n = 10000

To calculate the expectation and the standard deviation of the R.V. *X*, I used the Python functions `np.mean` and `np.std`. I then compare them to the theoretical values of each of the following equations:

- Uniform: $\mu_X = \frac{a+b}{2}$; $\sigma_X^2 = \frac{(b-a)^2}{12}$
- Exponential: $\mu_T = \beta$; $\sigma_T = \beta$
- Normal: $\mu_X = \mu$; $\sigma_X = \sigma$

Results and Conclusions

Table 1: Statistics for a Uniform Distribution			
Expectation		**Standard Deviation**	
Theoretical Calculation	Experimental Measure	Theoretical Calculation	Experimental Measure
2.5	2.5017	0.8660	0.8666

Table 1.1: Calculations for a Uniform distribution

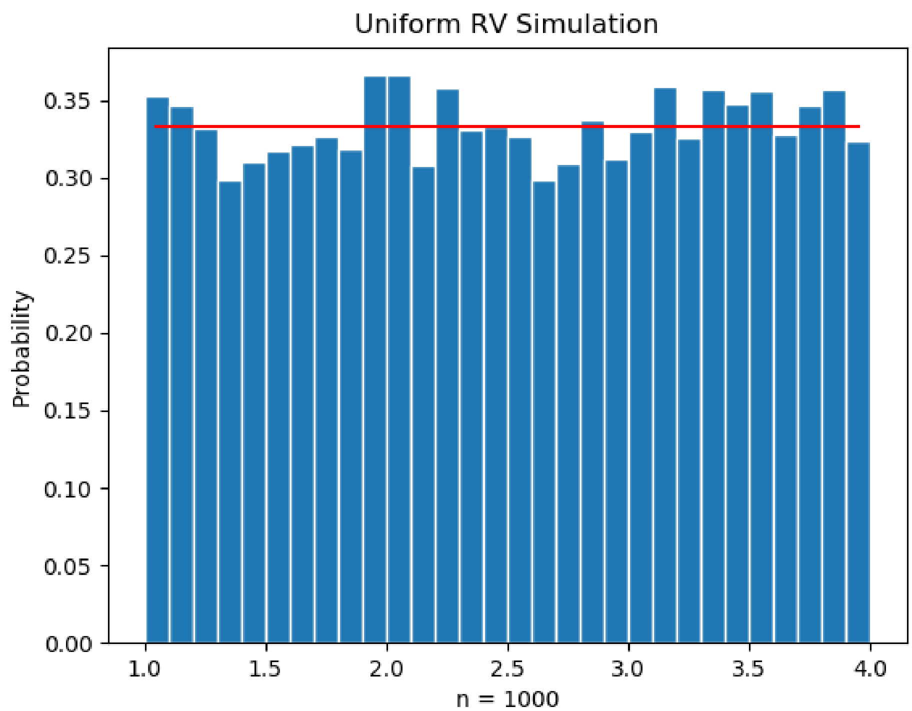

Figure 1.1: Graph of Uniform RV simulation

Table 2: Statistics for an Exponential Distribution			
Expectation		**Standard Deviation**	
Theoretical Calculation	Experimental Measure	Theoretical Calculation	Experimental Measure
40.0	40.0915	40.0	40.4125

Table 1.2: Calculations for an Exponential distribution

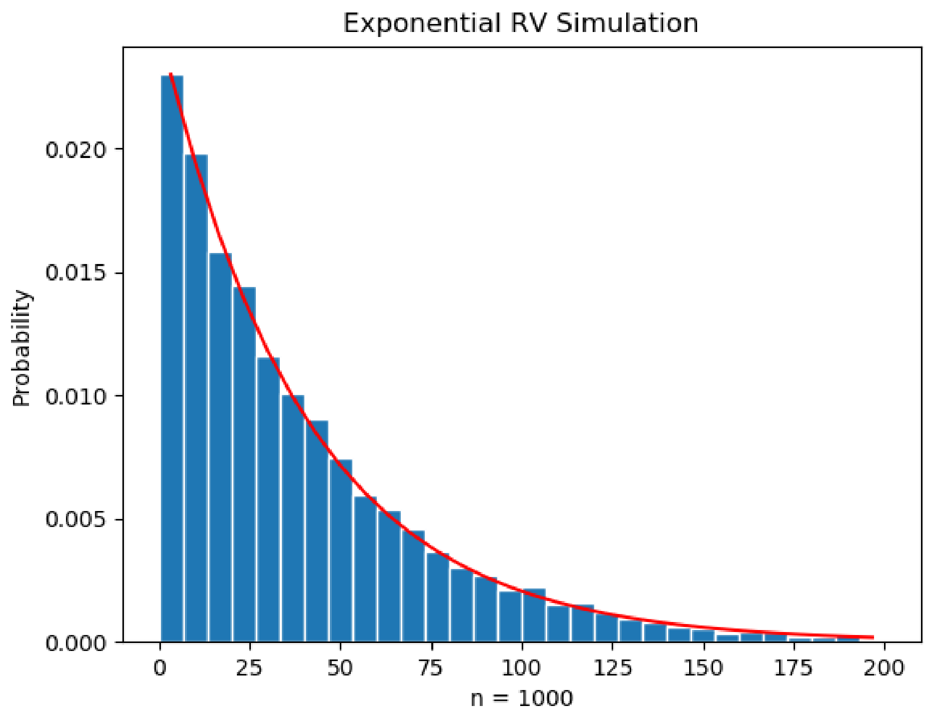

Figure 1.2: Graph of Exponential RV simulation

Table 3: Statistics for a Normal Distribution			
Expectation		**Standard Deviation**	
Theoretical Calculation	Experimental Measure	Theoretical Calculation	Experimental Measure
2.5	2.4986	0.75	0.7501

Table 1.3: Calculations for a Normal distribution

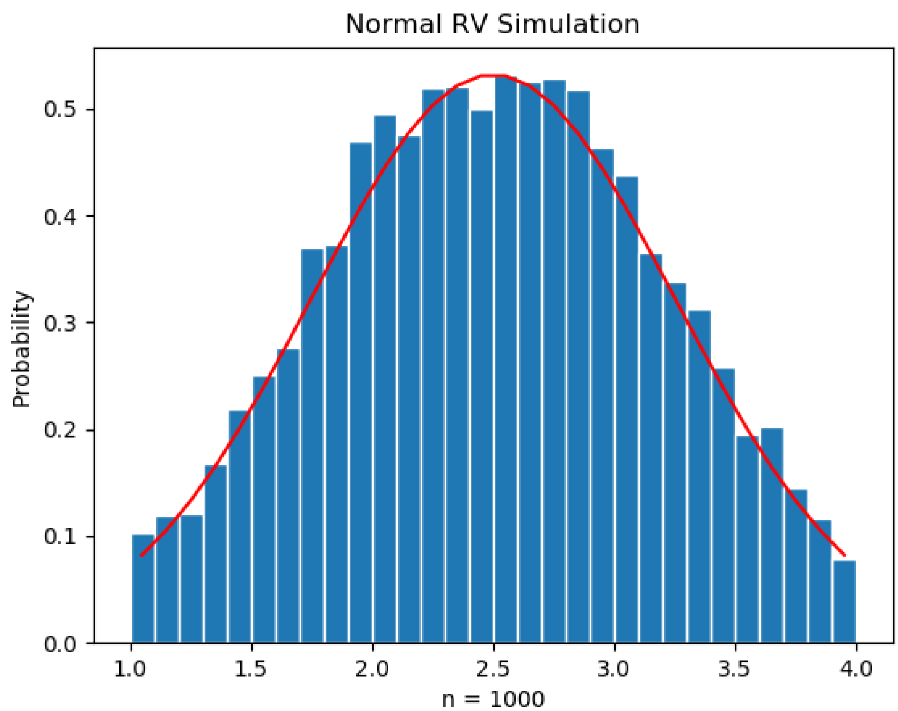

Figure 1.3: Graph of Normal RV simulation

Appendix

```
"""
Name:           Brian Nguyen
Class:          EE 381 - Probability & Stats
Start Date:     3/14/19
End Date:       3/15/19
"""

import numpy as np
import matplotlib.pyplot as plt


def uniform_rv():
    print('---------------\nUNIFORM RV')
    # Generate the values of the RV X
    a = 1
    b = 4
    n = 10000
    x = np.random.uniform(a, b, n)

    # Create bins and histogram
    nbins = 30
    edgecolor = 'w'
    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
    fig1 = plt.figure(1)
    plt.bar(b1,h1, width=barwidth, edgecolor=edgecolor)

    # PLOT THE UNIFORM PDF
    f = unif_pdf(1, 4, b1)
    plt.plot(b1, f, 'r')
    plt.title('Uniform RV Simulation')
    plt.xlabel('n = 1000')
    plt.ylabel('Probability')
    plt.show()

    # 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)


def exponential_rv():
    print('---------------\nEXPONENTIAL RV')
    beta = 40
    n = 10000
    t = np.random.exponential(beta, n)
    nbins = 30
    edgecolor = 'w'
    bins = [float(t) for t in np.linspace(0, 200, nbins + 1)]
```

```
    h1, bin_edges = np.histogram(t, 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)

    # PLOT THE UNIFORM PDF
    f = exp_pdf(beta, b1)
    plt.plot(b1, f, 'r')
    plt.title('Exponential RV Simulation')
    plt.xlabel('n = 1000')
    plt.ylabel('Probability')
    plt.show()

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

    print('mu_t = ', mu_t)
    print('sig_t = ', sig_t)

def normal_rv():
    print('---------------\nNORMAL RV')
    mu = 2.5
    sigma = 0.75
    n = 10000
    x = np.random.normal(mu, sigma, n)
    nbins = 30
    edgecolor = 'w'
    bins = [float(x) for x in np.linspace(1, 4, 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)

    # PLOT THE UNIFORM PDF
    f = normal_pdf(mu, sigma, b1)
    plt.plot(b1, f, 'r')
    plt.title('Normal RV Simulation')
    plt.xlabel('n = 1000')
    plt.ylabel('Probability')
    plt.show()

    # 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)


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


# PLOT THE EXPONENTIAL PDF
def exp_pdf(beta, t):
    f = np.exp((-1 / beta) * t) * (1 / beta)
    return f


# PLOT THE NORMAL PDF
def normal_pdf(mu, sig, z):
    f = np.exp(-(z - mu) ** 2 / (2 * sig ** 2)) / (sig * np.sqrt(2 * np.pi))
    return f


def main():
    uniform_rv()
    exponential_rv()
    normal_rv()


main()
```

2. The Central Limit Theorem

Introduction

This section explores the central limit theorem. In this simulation, we have a collection of books, each with a thickness W. The thickness W is a RV uniformly distributed between a minimum of *a* and a maximum of *b* cm. Using these values, we are to calculate the mean and standard deviation of the thickness.

In addition, we are to calculate books piled in stacks of n = 1, 5, or 15 books. The width S_n of staa ck of *n* books is the RV, and this RV has a mean of $\mu_{s_n} = n\mu_w$ and a standard deviation of $\sigma_{s_n} = \sigma_w\sqrt{n}$

The first simulation runs for $N = 10{,}000$ experiments, simulating the RV $S = W_1$ and creating a histogram that includes the function normal distribution probability function

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

The experiment is repeated for $n = 5$ and $n = 15$.

Methodology

In order to generate a stack of books with thickness W, I used the Python function

```
np.random.uniform(a, b, nbooks)
```

where a and b are the minimum and maximum thickness of a book respectively, and $nbooks$ is the number of books in the stack. The sum of the stack is calculated and appended to a list; this list will be used to plot the data of the uniform distribution. The experiment is repeated for $N = 10{,}000$ times for $n = 1, 5$, and 15.

Results and Conclusion

Mean thickness of a single book (cm)	Standard deviation of thickness (cm)
$\mu_w = 1.3171$	$\sigma_w = 0.0$

Number of books n	Mean thickness of a stack of n books (cm)	Standard deviation of the thickness for n books
$n = 1$	$\mu_w = 1.3171$	$\sigma_w = 0.0$
$n = 5$	$\mu_w = 12.50$	$\sigma_w = 1.9364$
$n = 15$	$\mu_w = 37.5$	$\sigma_w = 3.3541$

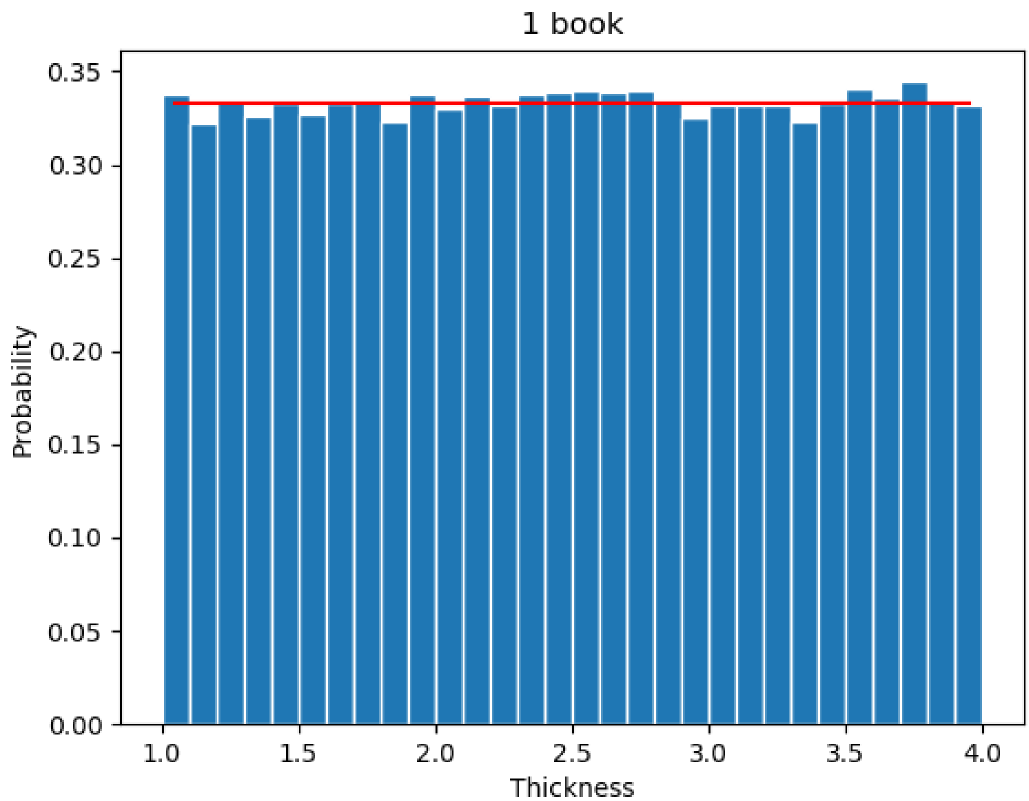

Figure 2.1: Uniform PDF for the Thickness of 1 Book

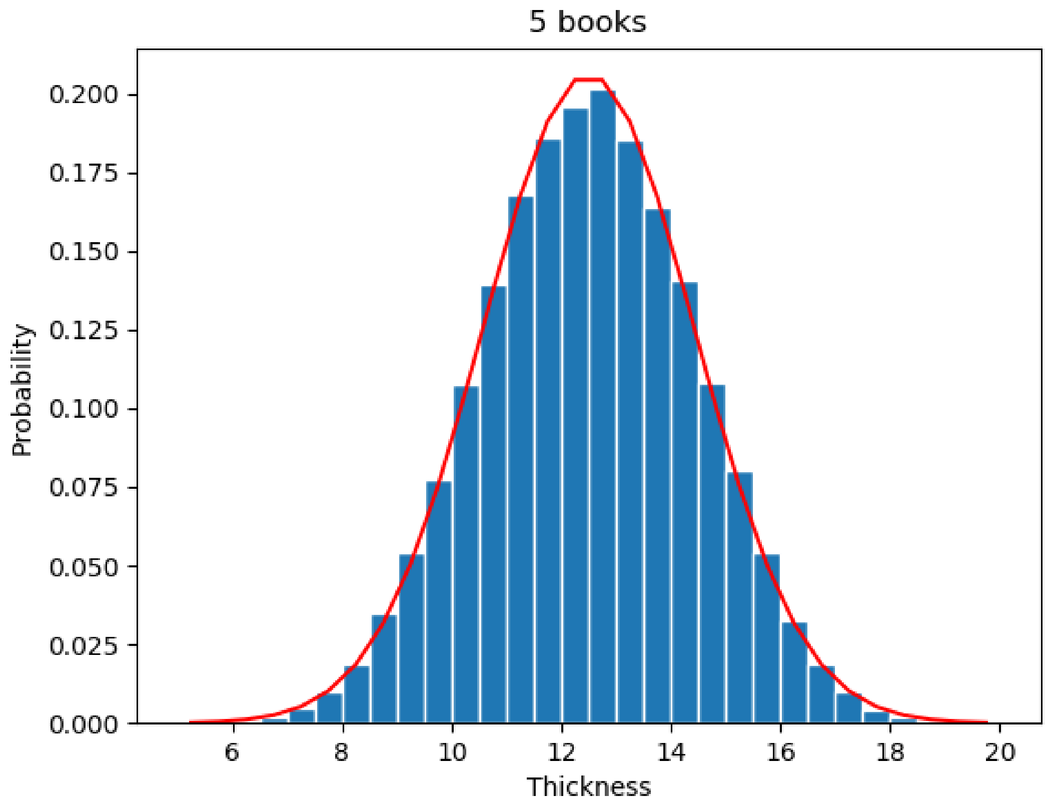

Figure 2.2: Normal PDF for the Thickness of 5 Books

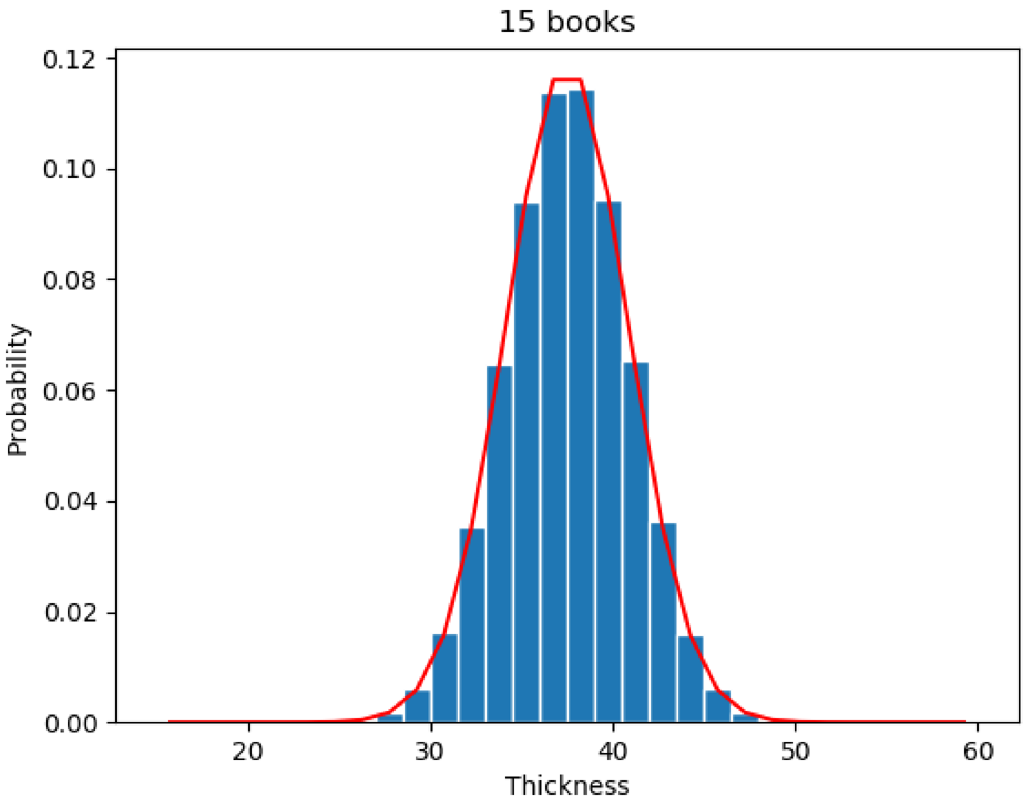

Figure 2.3: Uniform PDF for the Thickness of 15 Books

Appendix

```
"""
Name:           Brian Nguyen
Class:          EE 381 - Probability & Stats
Start Date:     3/15/19
End Date:       3/20/19
"""

import numpy as np
import matplotlib.pyplot as plt


# Generate the values of the RV X
def central_limit(nbooks):
    N = 100000
    a = 1
    b = 4
    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
    plt.close('all')

    # PLOT THE BAR GRAPH AND CALCULATE THE MEAN AND STANDARD DEVIATION
    fig1 = plt.figure(1)
    plt.bar(b1, h1, width=barwidth, edgecolor=edgecolor)
    if nbooks == 1:
        f = unif_pdf(a, b, b1)
        mu_x = np.mean(x)
        sig_x = np.std(x)
    else:
        f = gaussian(mu_x * nbooks, sig_x * np.sqrt(nbooks), b1)
        mu_x = mu_x * nbooks
        sig_x = sig_x * np.sqrt(nbooks)
    plt.plot(b1, f, 'r')
    title = str(nbooks) + " books"
    plt.title(title)
    plt.xlabel('Thickness')
    plt.ylabel('Probability')
    plt.show()

    print('---------------\nnbooks = ', nbooks)
    print('mu_x = ', mu_x)
    print('sig_x = ', sig_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


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


def main():
    central_limit(1)
    central_limit(5)
    central_limit(15)


main()
```

3. Distribution of the Sum of Exponential RVs

Introduction

In this section, we are to calculate the probability that a carton of 24 batteries will last within certain timeframes. The lifetime of a single battery lasts for $\beta = 40$ days. The first experiment is finding the probability that the carton will last longer than 3 years using the PDF graph, and the second experiment is finding the probability that the carton will last between 2.0 and 2.5 years using the CDF graph.

Methodology

In order to create a vector of 24 elements that represents a carton, I used the code

```
sum(np.random.exponential(beta, n))
```

to generate a vector where beta = 40 and n = 24. Each vector is stored in a random variable C.

The PDF is plotted using the values of $\sigma_C = \sigma_T\sqrt{24} = \beta\sqrt{24}$ and $\mu_C = 24\mu_T = 24\beta$ using the normal PDF function.

Using the PDF, the CDF is plotted using the cumulative sum of the PDF and multiplying it with `barwidth`, which produces $F(c)$.

In order to find the probability of how long a carton of batteries last for 3 three years, I created a for-loop using the values of the days the function $F(c)$ and iterated through the loop. In the loop, if a day that is iterated through in b1 is less than 1095 (three years), then the index of that day will be used to calculate the probability $P(S > 3\, x\, 365)$.

A similar method is used for the CDF graph. In this method, I kept track of two days: day_1 and day_2.

- If the value in b1 is greater than or equal to 730 (2 years), then the current index of `b1` will be assigned to `day_1`.
- If the value in b1 is greater than or equal to 912 (2.5 years), then the current index of `b1` will be assigned to `day_2`.

Results and Conclusion

QUESTION	ANS.
1. Probability that the carton will last longer than 3 years	0.2861
2. Probability that the carton will last between 2.0 and 2.5 years	0.02538

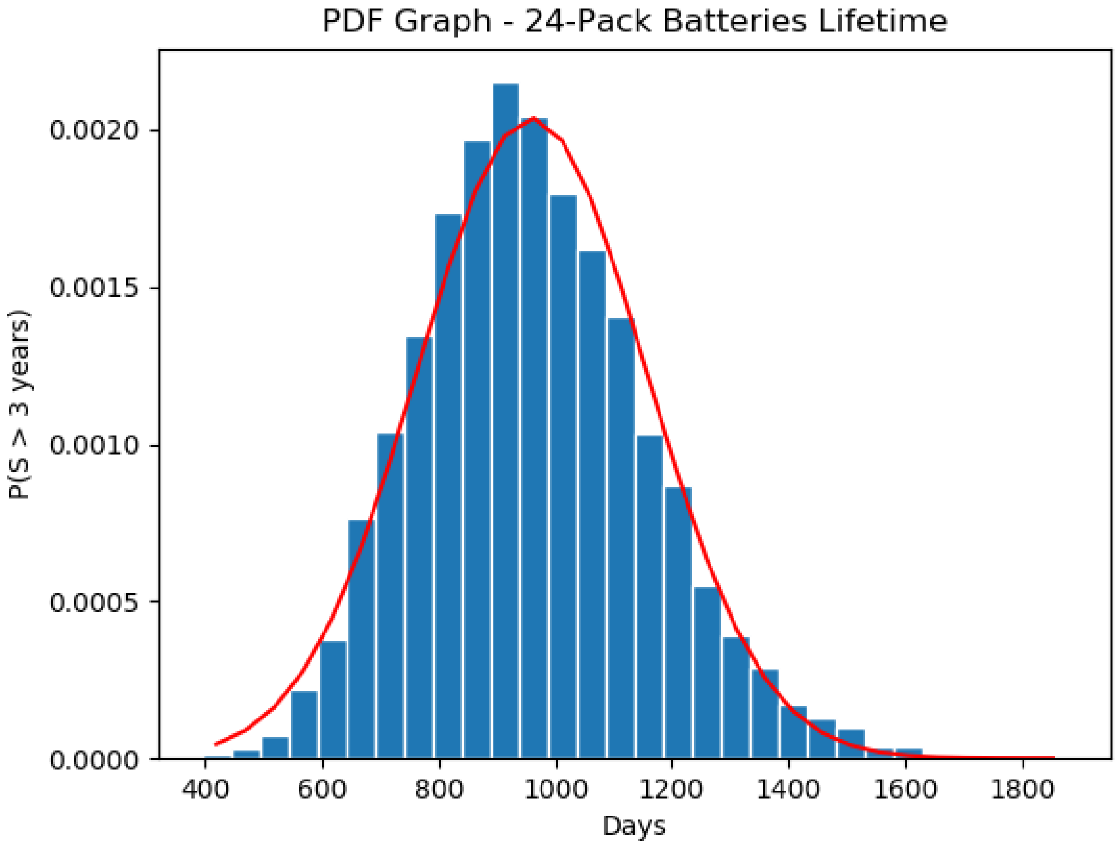

Figure 3.1: PDF Graph for the Lifetime of a Pack of 24 Batteries

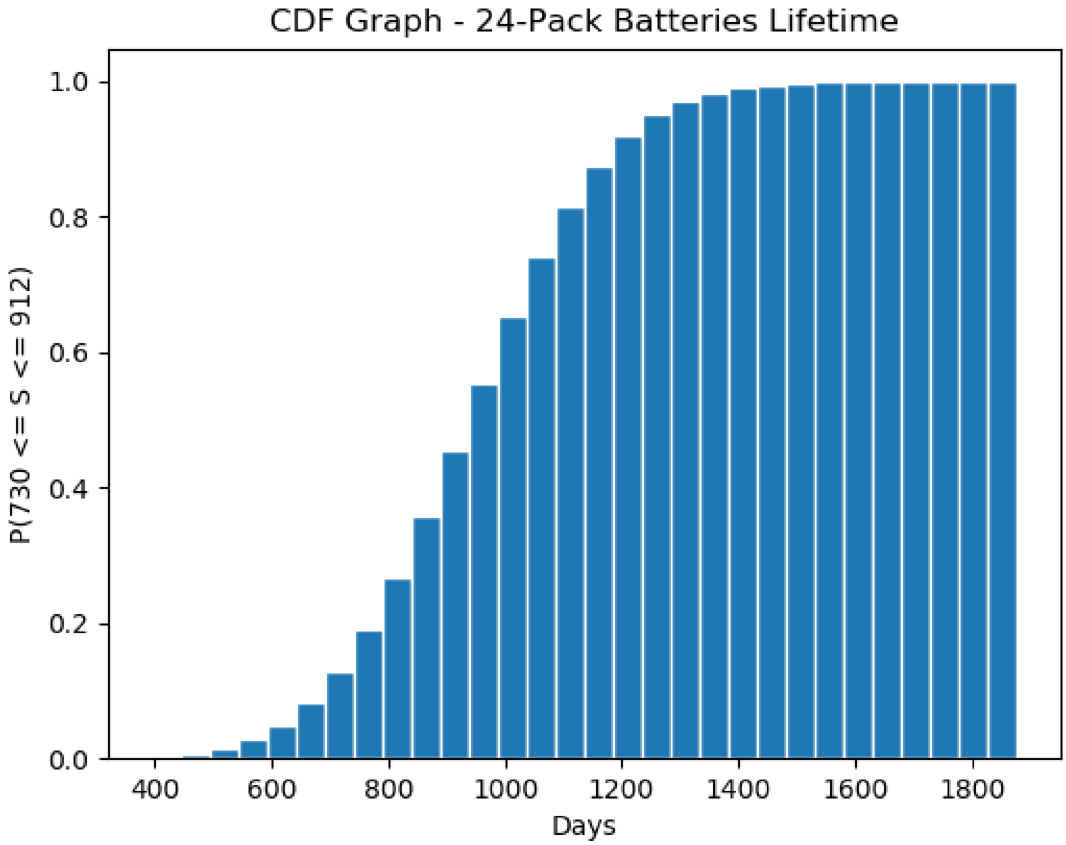

Figure 3.2: CDF Graph for the Lifetime of a Pack of 24 Batteries

Appendix

```
"""
Name:           Brian Nguyen
Class:          EE 381 - Probability & Stats
Start Date:     3/15/19
End Date:       4/5/19
"""


import numpy as np
import matplotlib.pyplot as plt


def distributed_sum():
    beta = 40
    N = 10000
    C = np.zeros((N, 1))
    n = 24
    for T in range(N):
        C[T] = sum(np.random.exponential(beta, n))

    # Plot PDF
    plt.figure(1)
    bins = 30
    edgecolor = "w"
    h1, bin_edges = np.histogram(C, bins, density=True)
    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]
    plt.close("all")
    plt.bar(b1, h1, width=barwidth, edgecolor=edgecolor)
    f = normal_pdf(24 * beta, np.sqrt(24) * beta, b1)
    plt.plot(b1, f, 'r')
    plt.title("PDF Graph - 24-Pack Batteries Lifetime")
    plt.xlabel("Days")
    plt.ylabel("P(S > 3 years)")
    plt.show()

    # Plot CDF
    F = np.cumsum(f) * barwidth
    plt.figure(2)
    plt.bar(b1, F, width=barwidth, edgecolor='w')
    plt.title("CDF Graph - 24-Pack Batteries Lifetime")
    plt.xlabel("Days")
    plt.ylabel("P(730 <= S <= 912)")
    plt.show()
    question_1(b1, F)
    question_2(b1, F)


def question_1(b1, F):
    day = 0
    for i, days in enumerate(b1):
        if days <= 1095:
            day = i
    # 1 - F(1095)
    p = 1 - F[day]
    print("P(S > 3 years):", p)
```

```
def question_2(b1, F):
    day_1, day_2 = 0, 0
    for i, days in enumerate(b1):
        if days >= 730:
            day_1 = i
            break
    for i, days in enumerate(b1):
        if days <= 912:
            day_2 = i
    # F(912) - F(730)
    p = F[day_2] - F[day_1]
    print("P(730 <= S <= 912)", p)


def normal_pdf(mu, sig, z):
    f = np.exp(-(z - mu) ** 2 / (2 * sig ** 2)) / (sig * np.sqrt(2 * np.pi))
    return f


def main():
    distributed_sum()


main()
```