Department of Electrical Engineering

California State University, Long Beach

Probability and Statistics with Applications to Computing

Duc Tran

Gaussian Distributions

Submitted by

Devin Suy

2020 December 03

Introduction

The purpose of this laboratory was to utilize the Python scripting language along with a few additional numerical libraries to perform three different probability simulations. These simulations modeled the Normal Distribution, the Central Limit Theorem, and Distribution of Exponential Random Variables. The simulations conducted were: Implementing the probability density function (pdf) and cumulative distribution function (cdf) of a normal distribution and comparing behavior for differing $f(x; \mu, \sigma^2)$ mean and variance parameters; Modeling the thickness of a book as a random variable to observe central limit theorem in summation of the random variables; Modeling the lifetime of a battery as a random variable that is exponentially distributed and observing the pdf and cdf of a carton of such batteries in relation to the normal distribution.

Methods

In this first experiment the following equations are implemented in Python:

$$f(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \qquad F(x;\mu,\sigma^2) = \frac{1}{2}\operatorname{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right) + \frac{1}{2}$$

Representing the probability density function f(x) and the cumulative distribution function F(x) of a Gaussian random variable x. in which *mu* represents the mean and is the location parameter, and the variance can be used to determine the standard deviation which is the scale parameter of our standard normal distribution. Additionally, the error function *erf* is defined as follows:

$$\operatorname{erf} z = \frac{2}{\sqrt{\pi}} \int_0^z e^{-} t^2 \, dt$$

$\operatorname{erf}(z)$ = Gauss error function

t = time

z = a complex number

The *erf*() function implementation is used from the *math* standard library and is available in Python 2.7 and subsequent versions. Here the error function (erf) is used the express the CDF F(x) since integration along [-inf, x] will yield our desired output of this function. Plots are then generated and compared for varying *(mu, var*) parameter combinations.

The next experiment involves the sum S_n of widths in a stack of books, since width W is an independent random variable in [a=1, b=3], the uniform distribution is defined as follows:

$$f(x) = \frac{1}{b-a} \text{ for } a \le x \le b \qquad \text{Mean } \mu = \frac{a+b}{2} \qquad \text{Variance } \sigma^2 = \frac{(b-a)^2}{12}$$

Functions *uniform_mean(a,b)* and *uniform_sdev(a,b)* use the above equations to implement mean and standard deviation of the width of a book. Since S_n for a stack is simply the sum of the book's widths, we can also define the following: $\mu_{S_n} = n * u; \quad \sigma_{S_n} = \sigma\sqrt{n}$ which are implemented in the functions *stack_mean(n,a,b)* and *stack_sdev(n,a,b*). To sample our values from a uniform distribution we simply pass the same arguments to the numpy function: *numpy.random.uniform(a,b,n)*. The histogram bars are then generated using *numpy.histogram* with our sampled values. This is repeated for stack size *n = {1,5,15}* over *N =100,000* samples and plotted using *matplotlib* along with the probability density function in which values are obtained using the *pdf()* function implemented in the first experiment.

In this last experiment a carton is defined as *n = 24* batteries each with a lifetime that is our exponentially distributed random variable *T* which compose our carton *C* and is defined as:

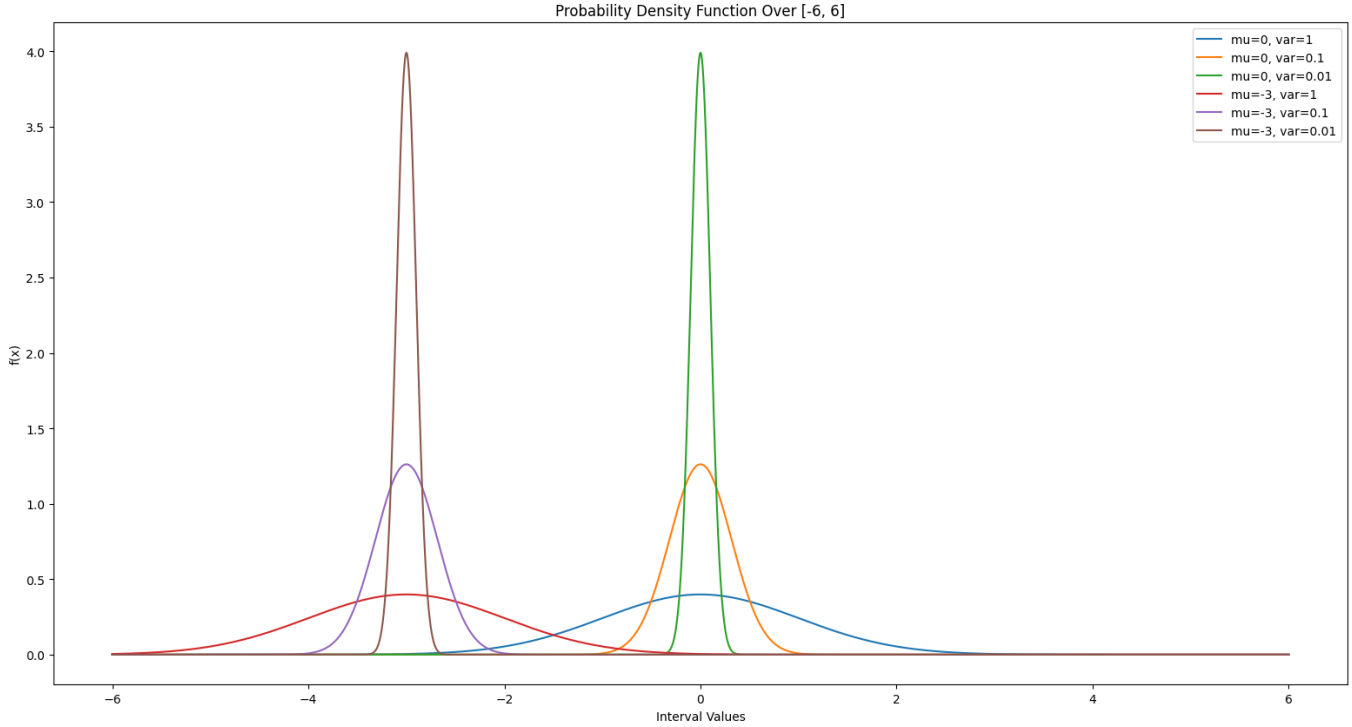
$$f_T(t;\beta) = \begin{cases} \frac{1}{\beta}\exp(-\frac{1}{\beta}t), & t \ge 0 \\ 0, & t < 0 \end{cases} \qquad \sigma_T = \beta \qquad \mu_T = \beta$$

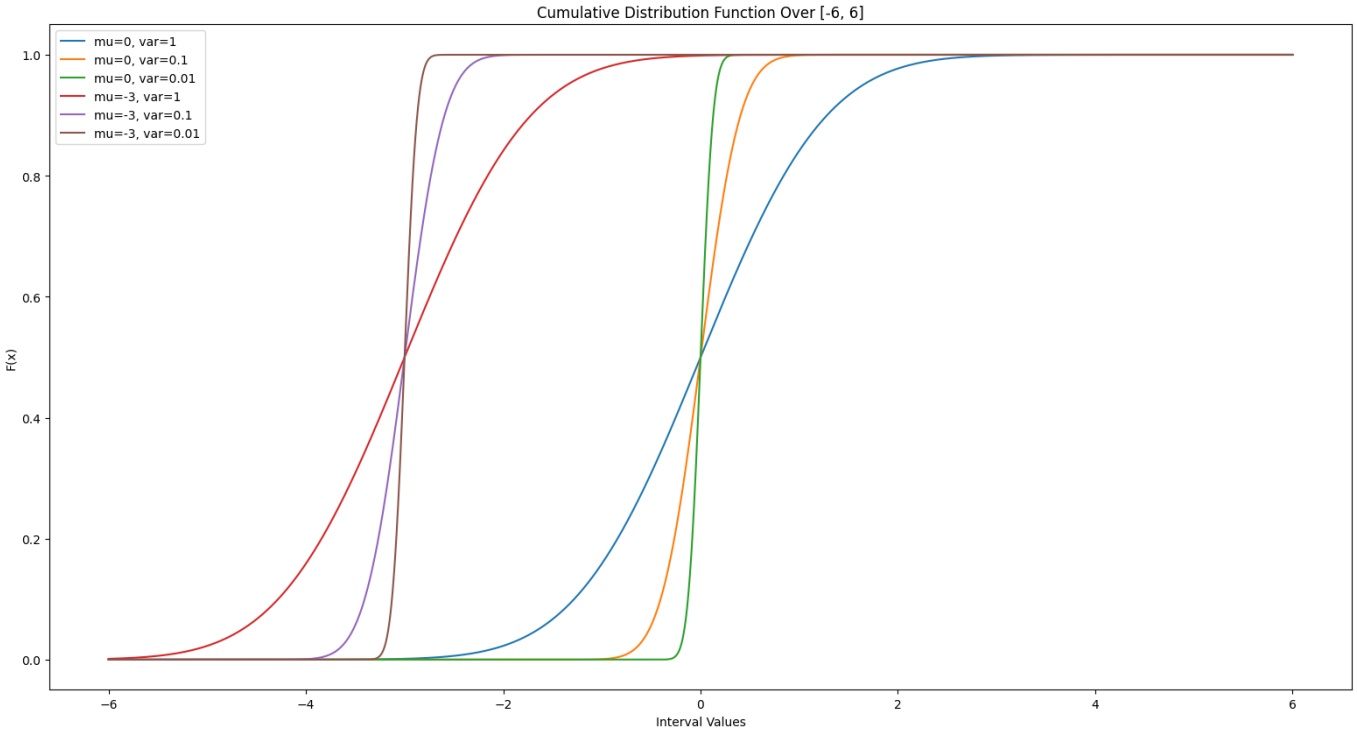
$$\sigma_C = \beta\sqrt{n} \qquad \mu_C = n\beta$$

In implementation, a carton is therefore represented as a list of *n = 24* values sampled by passing *β = 45 days* and n to the *numpy* function *random.exponential*(). The values of this list are summed and stored in a list *C_vals* representing the life of the carton. Similar to the previous experiment, histogram bars are generated and plotted against the normal distribution with the calculated mean and variance passed to the *pdf()* function from the first experiment. An additional plot is created for the cumulative distribution function by using *numpy.cumsum()* in which the histogram values multiplied by the width of the bars is passed as the argument, effectively giving us the CDF by means of calculating the area or the integral of the probability density function. Viewing the plot of the CDF allows us to estimate the probability of any given lifetime of a carton by observing the probability of the different regions of the distribution.

Results and Discussion

Experiment One:





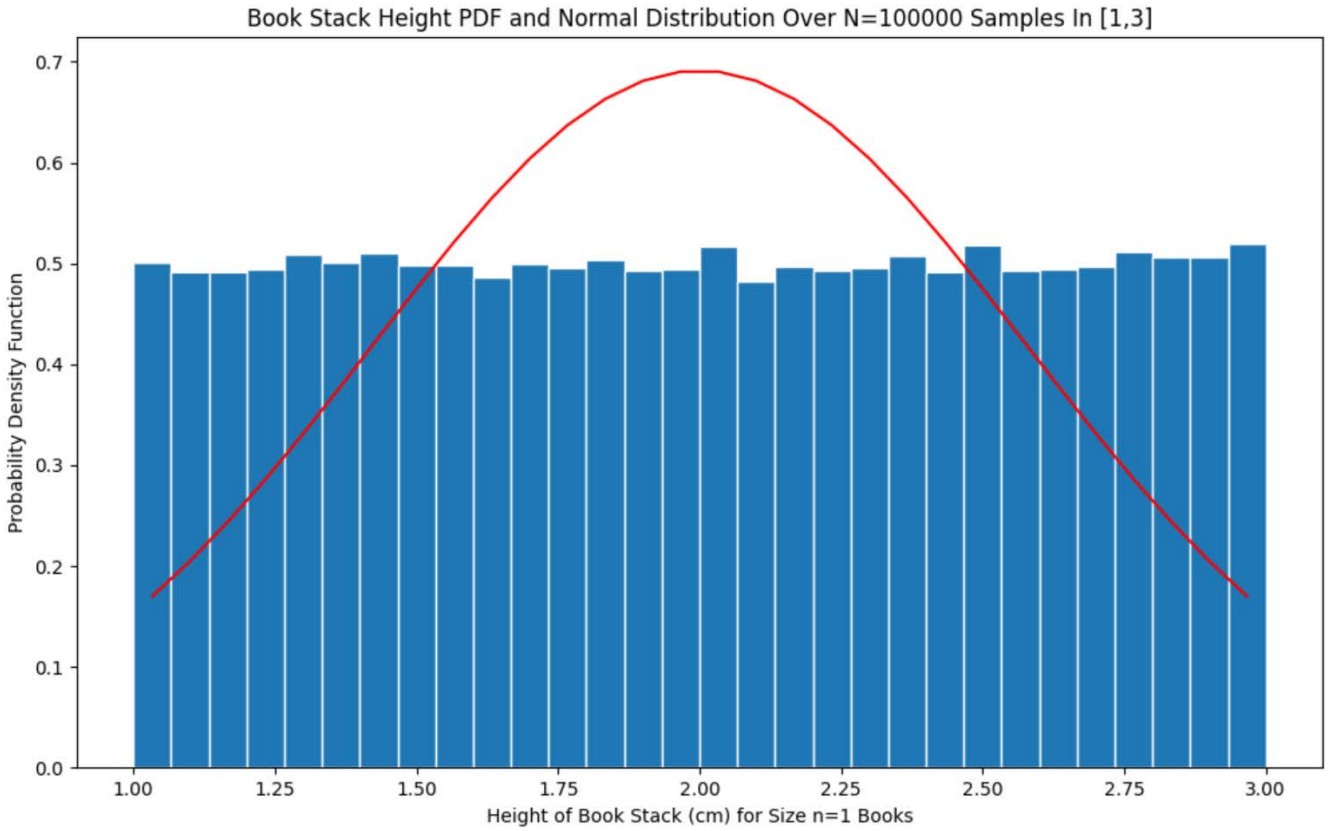
```
PDF Implementation
------------------
Probability Density Function @ x=1, mu=0, variance=1
   Using self implementation : 0.24197072451914337
   Using scipy.stats.norm.pdf: 0.24197072451914337

CDF Implementation
------------------
Cumulative Distribution Function @ x=1, mu=0, variance=1
   Using self implementation : 0.8413447460685428
   Using scipy.stats.norm.cdf: 0.8413447460685429
```

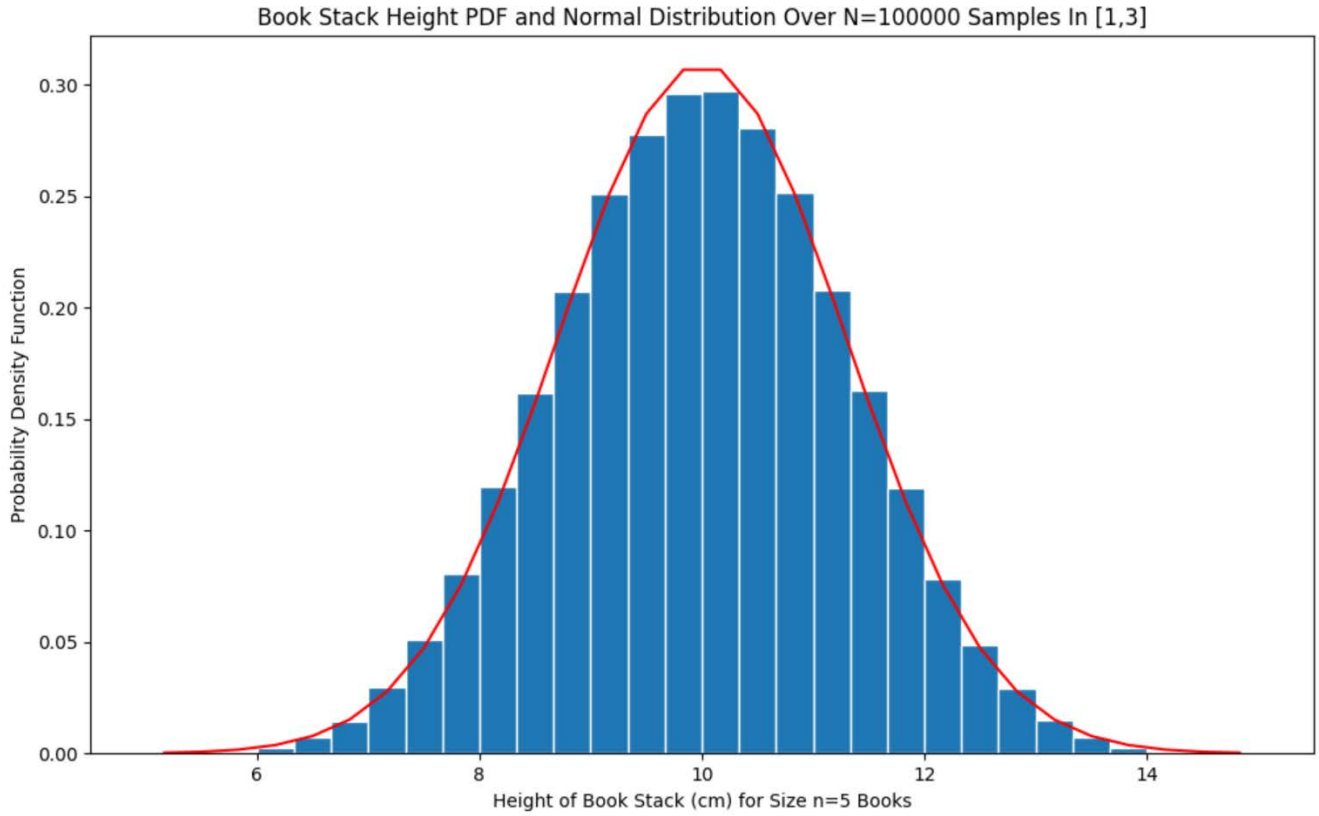
As discussed, the mean in our plot represents the location parameter and standard deviation is our scale parameter. Therefore, as shown in the graph of the probability density function a larger value of *mu* translates to a shift to the right along the x axis. We also observe that for smaller values of *var*, giving us a smaller standard deviation, for identical values of *mu* the values grow the same, but the scale increases with higher peak values compared to plots with larger values of *var*. This is also reflected in the cumulative distribution function where we where and how sharply the values increase as taking the gradient would yield us the PDF. Implementation is verified for validity by testing against the respective functions using the *scipy.stats.norm* module.

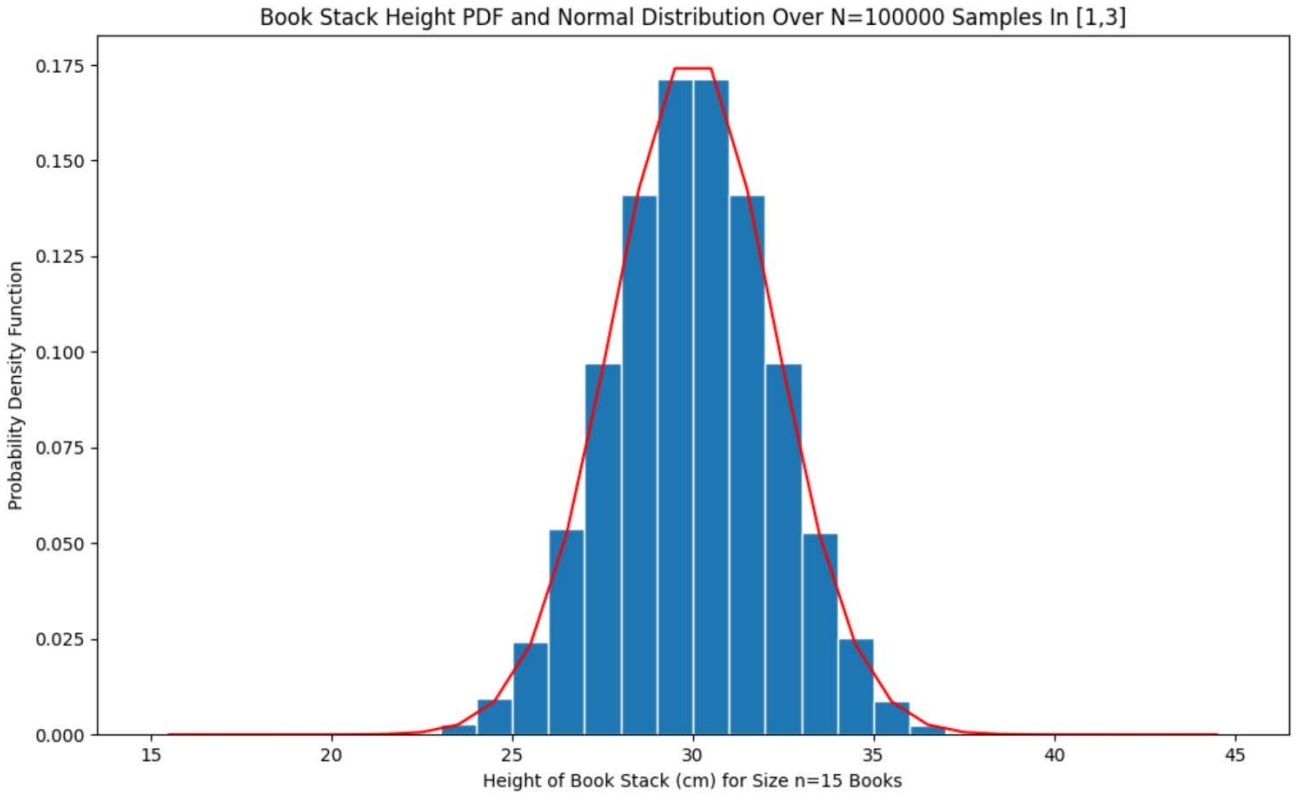
Experiment Two:

```
Mean And Sdev Of Book Stack
---------------------------
   Stack @ n=1 books:
      Sn Mean: 2.0 cm
      Sn Sdev: 0.577 cm
   Stack @ n=5 books:
      Sn Mean: 10.0 cm
      Sn Sdev: 1.29 cm
   Stack @ n=15 books:
      Sn Mean: 30.0 cm
      Sn Sdev: 2.235 cm
```

```
For A Single Book Over [1,3]
----------------------------
   Mean: 2.0 cm
   Sdev: 0.577 cm
```


Book Stack Height PDF and Normal Distribution Over N=100000 Samples In [1,3]
Probability Density Function
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Height of Book Stack (cm) for Size n=1 Books


Book Stack Height PDF and Normal Distribution Over N=100000 Samples In [1,3]
Probability Density Function
0.00
0.05
0.10
0.15
0.20
0.25
0.30
6
8
10
12
14
Height of Book Stack (cm) for Size n=5 Books

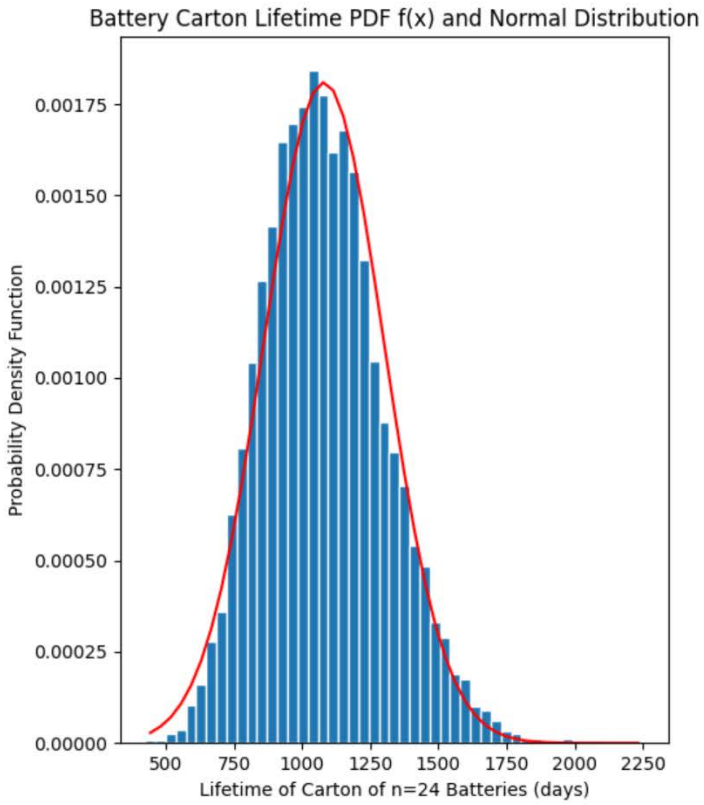


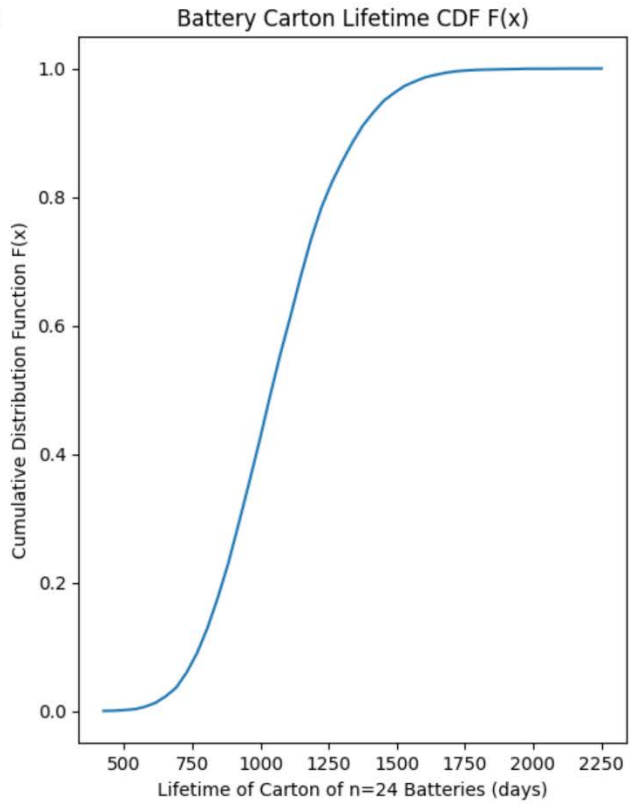
Since our experiment involves independent random variables, the width of each book, central limit establishes that the normalized sum will approach a normal distribution. Even in situations in which the variable itself does not follow the normal distribution, for a sufficiently large sample size the sum does in fact appear to resemble the gaussian curve. This is further demonstrated when we compare the plot of *n=1* and *n=15* since the former shows that the book widths are not normally distributed, and the latter shows that despite this the sum of the widths in a stack of books still approaches the normal distribution drawn in red. The size of "sufficient large sample" can vary but generally $n \geq 30$ can be considered large enough for central limit theorem to hold.

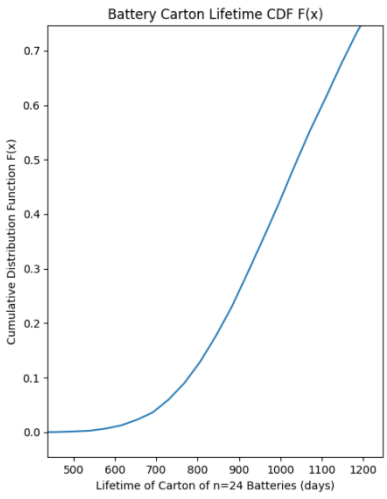
Mean Thickness of a Single Book (cm)	**Standard Deviation of the Thickness for *n* books**
$\mu_{S_n} = 2.0$	$\sigma_{S_n} = 0.58$

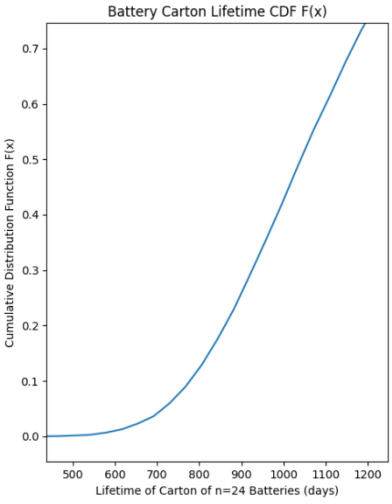
Number of Books *n*	**Mean Thickness of a Stack of Books (cm)**	**Standard Deviation of the Thickness for *n* books**
n=1	$\mu_{S_n} = 2.0$	$\sigma_{S_n} = 0.58$
n=**5**	$\mu_{S_n} = 10.0$	$\sigma_{S_n} = 1.29$
n=15	$\mu_{S_n} = 30.0$	$\sigma_{S_n} = 2.24$

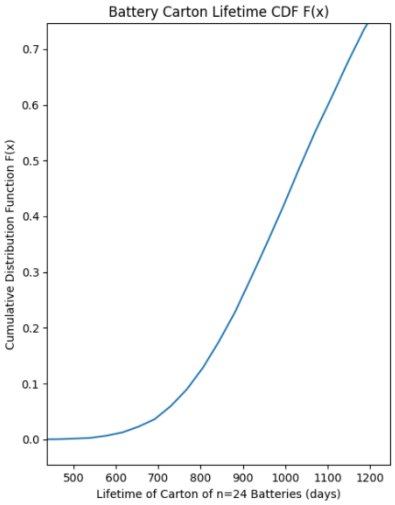
Experiment Three:











Similar to the previous experiment in which the book widths were uniformly distributed, the same properties of central limit theorem hold true for the exponentially distributed battery lifetimes as shown. The battery lifetime *T* is an independent random variable which is not normally distributed, yet for a carton of batteries C the lifetime of the carton, that is the sum of the independent random variables T, is shown to approach the normal distribution. Applying these properties, by zooming in on the cumulative distribution function for the carton lifetime boxed in red we can approximate the probability of any given carton lifespan in our range as shown below:

P(S > 3 years)

=> P(S > 1095)

=> 1 - P(S $\leq$ 1095)

=> 1 - F(1095)

~ **0.411**

P(2.0 years < S < 2.5 years)

=> P(730 < S < 912)

=> F(912) - F(730)

=> 0.279 - 0.061

~ **0.218**

Conclusion

Performing this laboratory was a useful meansof being able to apply the fundamental concepts of the uniform distribution, exponential distribution, and central limit theorem. We have previously studied the probability density function and cumulative distribution function, mean and standard deviation, but it was interesting to be able apply these concepts to actual scenarios and observe how each factor contributes to the behavior exhibited in the data and resulting plots.

Setting up some of these plots has become easier after a bit more exposure to working with matplotlib, notably this laboratory and consulting the available api documentation helped build a better understanding of histograms and setting up bins for plots. It is fascinating to be able to observe how the summation of independent random variables, despite having a different type of distribution whether it be uniform or even exponential, still approaches the normal distribution. Such a property can be very desirable in statistics as we have seen the useful applications of working with the normal distribution.

Appendix

Dependencies:

```
from scipy import stats
from math import erf, floor, ceil, sqrt, pi, e
from matplotlib import pyplot as plt
from collections import OrderedDict
import numpy as np
```

Experiment 1(a) Implementation:

```
'''
---------------------------
Normal Gaussian Distribution
---------------------------
'''
# Implementation of probability density function f(x)
def pdf(x, mu=0, var=1):
    exponent = (-1 * pow((x-mu), 2)) / (2 * var)
    return (1 / sqrt(2*pi*var)) * pow(e, exponent)

# Implementation of cumulative distribution function F(x)
def cdf(x, mu=0, var=1):
    return 0.5 + (0.5 * erf((x-mu) / sqrt(2*var)))

# Verify values obtained against scipy.stats.normal module
def show_verification():
    print("PDF Implementation\n------------------")
    print("Probability Density Function @ x=1, mu=0, variance=1")
    print("   Using self implementation :", pdf(1))
    print("   Using scipy.stats.norm.pdf:", stats.norm.pdf(1))

    print("\nCDF Implementation\n------------------")
    print("Cumulative Distribution Function @ x=1, mu=0, variance=1")
    print("   Using self implementation :", cdf(1))
    print("   Using scipy.stats.norm.cdf:", stats.norm.cdf(1))

show_verification()
```

Experiment 1(b) Implementation:

```
# Given a range [start_val, end_val] returns a data dictionary
# mapping input -> pdf/cdf output vals for each val in the range
def get_plot(start_val, end_val, use_pdf=True, mu=0, var=1):
    output = {}
    # Map using probability density function
    if use_pdf:
        for val in np.linspace(start_val, end_val, 10000): output[val] = pdf(val,
 mu, var)

    # Map values using cumulative distribution function
    else:
        for val in np.linspace(start_val, end_val, 10000): output[val] = cdf(val,
 mu, var)

    return output

# Generate pdf/cdf plots for the specified (mu, variance) pairs
def generate_all_plots(start_val=-6, end_val=6):
    # Maps argument ID to mu, variance arguments
    mu_var = {
        0 : [0,1], 1 : [0, 0.1], 2 : [0, 0.01], 3 : [-3, 1], 4 : [-
3, 0.1], 5 : [-3, 0.01]
    }
    # Maps mu_var argument ID to the corresponding plot values
    plot_vals_pdf = OrderedDict()
    plot_vals_cdf = OrderedDict()
    x_vals = np.linspace(start_val, end_val, 10000)

    # Generate plots for each of our mu, variance settings
    for i in range(len(mu_var)):
        params = mu_var[i]
        plot_vals_pdf[i] = get_plot(start_val, end_val, use_pdf=True, mu=params[0
], var=params[1])
        plot_vals_cdf[i] = get_plot(start_val, end_val, use_pdf=False, mu=params[
0], var=params[1])

    # Plot PDF results for each parameter
    for i in range(len(mu_var)):
        mu, var = mu_var[i]
        plt.plot(x_vals, plot_vals_pdf[i].values(), label=("mu=" + str(mu) + ", v
ar=" + str(var)))
    plt.title("Probability Density Function Over [" + str(start_val) + ", " + str
(end_val) + "]")
    plt.xlabel("Interval Values")
```

```
    plt.ylabel("f(x)")
    plt.legend()
    plt.show()

    # Plot CDF results for each parameter
    for i in range(len(mu_var)):
        mu, var = mu_var[i]
        plt.plot(x_vals, plot_vals_cdf[i].values(), label=("mu=" + str(mu) + ", v
ar=" + str(var)))
    plt.title("Cumulative Distribution Function Over [" + str(start_val) + ", " +
 str(end_val) + "]")
    plt.xlabel("Interval Values")
    plt.ylabel("F(x)")
    plt.legend()
    plt.show()

generate_all_plots()
```

Experiment 2(a) Implementation:

```
'''
--------------------
Central Limit Theorem
--------------------
'''

# Returns the mean of uniform distribution over [a,b]
def uniform_mean(a, b):
    return round(0.5*(a+b), 3)

# Return the standard deviation over the uniform distr
def uniform_sdev(a, b):
    return round(sqrt((1/12) * pow(b-a, 2)), 3)

# Display the mean and sdev of a single book (cm) over [a,b]
def single_book(a=1, b=3):
    print("For A Single Book Over [" + str(a) + "," + str(b) + "]"
    + "\n---------------------------")
    print("   Mean:", uniform_mean(a,b), "cm")
    print("   Sdev:", uniform_sdev(a,b), "cm")

single_book()
```

Experiment 2(b) Implementation:

```
# For a stack of n books, return the mean thickness
def stack_mean(n, a, b):
    return round(n * uniform_mean(a,b), 3)

# For a stack of n books, return the standard deviation thickness
def stack_sdev(n, a, b):
    return round(uniform_sdev(a,b) * sqrt(n), 3)

# Display the mean and sdev of a stack of books
def book_stack(n_vals=[1,5,15], a=1, b=3):
    print("Mean And Sdev Of Book Stack" +
    "\n---------------------------")
    for n in n_vals:
        print("   Stack @ n=" + str(n) + " books:")
        print("      Sn Mean:", stack_mean(n, a, b), "cm")
        print("      Sn Sdev:", stack_sdev(n, a, b), "cm")

book_stack()
```

Experiment 2(c) Implementation:

```
# Perform N samples of size n to generate probability
# histogram and plot of normal PDF f(x)
def run_simu_util(n, a=1, b=3, N=100000):
    mean = stack_mean(n, a, b)
    s_dev = stack_sdev(n, a, b)

    # Randomly sample n values from [a,b]
    s_vals = []
    for _ in range(N): s_vals.append(np.sum(np.random.uniform(a, b, n)))

    # Generate histogram bars
    hist, b_edges = np.histogram(a=s_vals, bins=np.linspace(n*a, n*b, 31), densit
y=True)
    bar = (b_edges[:-1]+b_edges[1:]) / 2
    w = bar[1]-bar[0]

    return [mean, s_dev, hist, b_edges, bar, w]

# Run the simulation for each n value
def run_book_simu(n_vals=[1,5,15], a=1, b=3, N=100000):
    plt_data = []
    for n in n_vals:
        mean, s_dev, hist, b_edges, bar, w = run_simu_util(n, a, b, N)

        # Plot results
        plt.bar(x=bar, height=hist, width=w, edgecolor='w')
        pdf_vals = [pdf(x, mu=mean, var=pow(s_dev,2)) for x in bar]
        plt.plot(bar, pdf_vals, 'r')
        plt.xlabel("Height of Book Stack (cm) for Size n=" + str(n) + " Books")
        plt.ylabel("Probability Density Function")
        plt.title("Book Stack Height PDF and Normal Distribution Over N=" + str(N
)
        + " Samples In [" + str(a) + "," + str(b) + "]")
        plt.show()

run_book_simu()
```

Experiment 3 Implementation:

```
'''
-------------------------------------------------------
Distribution of the sum of exponential random variables
-------------------------------------------------------
- A battery lasts an average of β = 45 days
- Batteries are purchased in a carton of 24
'''

# Generate the n values with the given beta in an exponential
# distribution, representing our "batteries" comprising a "carton"
def generate_carton(beta, n):
    return np.random.exponential(beta, n)

def run_carton_simu(beta=45, n=24, N=10000):
    # Run N simulations of cartons, track lifetime sum C
    C_vals = []
    for _ in range(N): C_vals.append(sum(generate_carton(beta, n)))

    # Central Limit: Calculate mean and s_dev for gaussian plot
    mean = n * beta
    s_dev = beta * sqrt(n)

    # Find the min and max values for bins
    min_val = float('inf')
    max_val = float('-inf')
    for c_val in C_vals:
        if c_val < min_val: min_val = c_val
        if c_val > max_val: max_val = c_val
    min_val = floor(min_val-1)
    max_val = ceil(max_val+1)
    print(min_val)

    # Plot PDF of carton lifetime f(c)
    plt.subplot(1,2,1)
    hist, b_edges = np.histogram(C_vals, bins=np.linspace(min_val, max_val), dens
ity=True)
    bar = (b_edges[:-1]+b_edges[1:]) / 2
    w = bar[1]-bar[0]
    plt.bar(x=bar, height=hist, width=w, edgecolor='w')

    # Plot normal distribution with calculated mean and s_dev
    norm_vals = [pdf(x, mu=mean, var=pow(s_dev,2)) for x in bar]
    plt.plot(bar, norm_vals, 'r')
    plt.xlabel("Lifetime of Carton of n=" + str(n) + " Batteries (days)")
```

```
    plt.ylabel("Probability Density Function")
    plt.title("Battery Carton Lifetime PDF f(x) and Normal Distribution")

    # Plot cumulative distribution function
    plt.subplot(1,2,2)
    cdf = np.cumsum(hist*w)
    plt.plot(np.linspace(min_val, max_val, len(cdf)), cdf)
    plt.xlabel("Lifetime of Carton of n=" + str(n) + " Batteries (days)")
    plt.ylabel("Cumulative Distribution Function F(x)")
    plt.title("Battery Carton Lifetime CDF F(x)")
    plt.show()

run_carton_simu()
```