Department of Electrical Engineering

California State University, Long Beach

Probability and Statistics with Applications to Computing

Duc Tran

Uniform Distributions

Submitted by

Devin Suy

2020 November 01

Introduction

The purpose of this laboratory was to utilize the Python scripting language along with a few additional numerical libraries to perform four different probability simulations. The simulations conducted were: Modeling the uniform distribution for a continuous random variable, implementing the probability density function and cumulative distribution function over a given interval. As well as generating a circle and randomly selecting three points on the circumference, simulating the probability that all three points lie on the same semicircle.

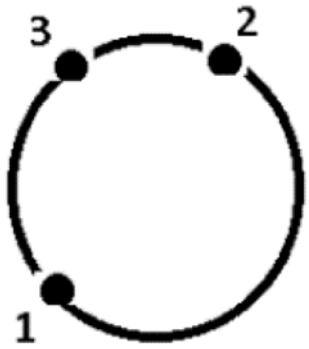
Methods

In this first experiment the modeled interval for the probability density function and cumulative distribution function was performed over (1,10). The respective functions are defined as follows:

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

This experiment was simulated with two different methods: manual implementation of the above functions through the use of conditional statements, as well as utilization of the Python PDF and CDF functions of the *scipy.stats.uniform* library. *Numpy.linspace()* was utilized to generate *n=1000000* evenly space values over our interval. Finally, matplotlib was used to create subplots side by side of the values in our interval mapped to their respective PDF and CDF values defined by their functions and equations.

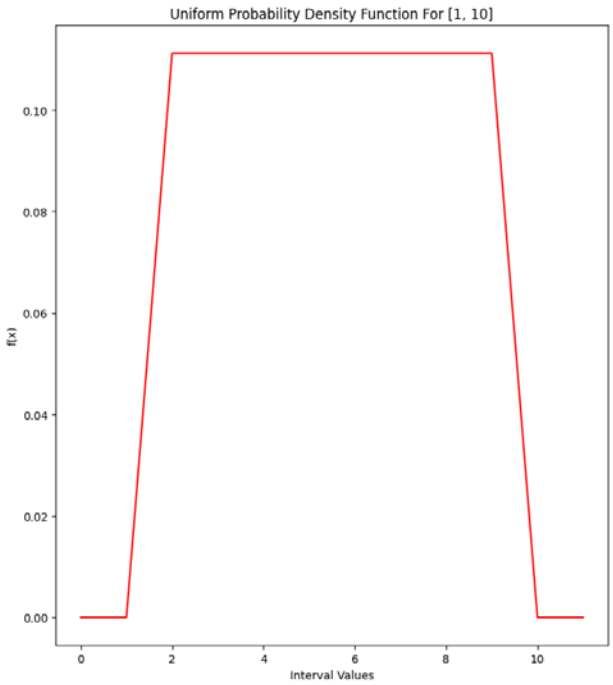
The following experiment was implemented by simulating a circle by calculating its circumference with *radius=3* using the following equation: $C = 2\pi \cdot r$ the length of the semicircle *semi_len* is simply defined as C/2 or half the length of the calculated circumference. In the model utilized, the location *radius* distance directly above the center of the circle is considered to assume the value zero. Moving clockwise from this location along the circumference, the position value increases all the way up until the circumference value. In essence, points randomly selected along the circumference of our circle assume the value *[0, C]* and were randomly selected using *numpy.random.uniform()*.

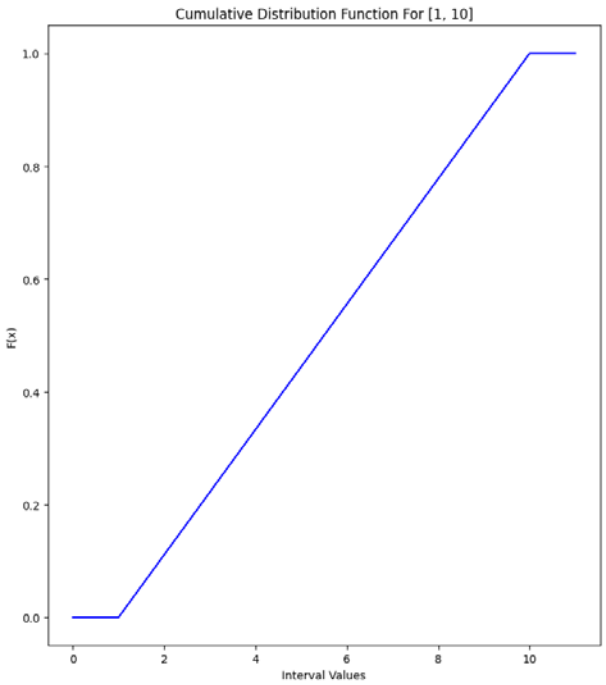


From here we can determine if the three selected points lie on the same semicircle by first sorting the point location values in ascending order. We can then simply locate the middle point and compare it to the two ends, the point location at index 0, and the point location at the last index. Our selected points lie on the same semicircle if the distance from the middle point to both end points is no more than the length of our the calculated *semi_len* as the randomly selected points will not be evenly spaced and there are many different semicircles we positions we can consider.

Results and Discussion

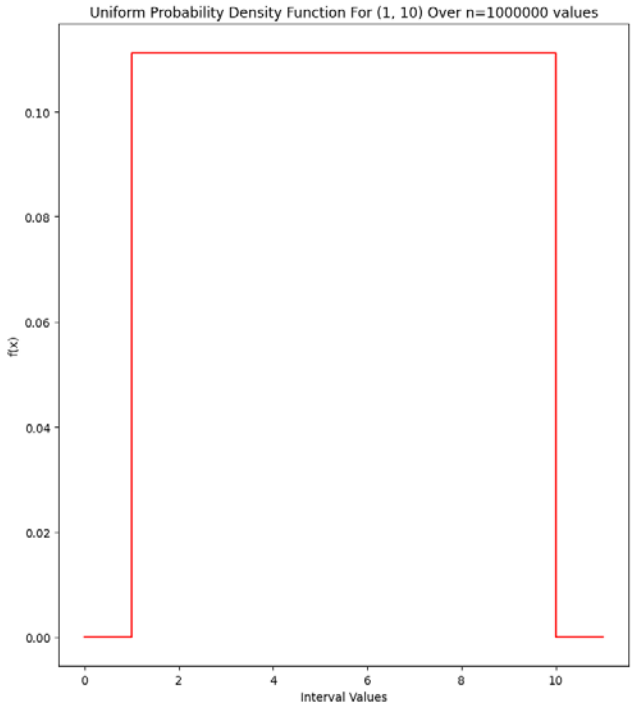
Experiment One:

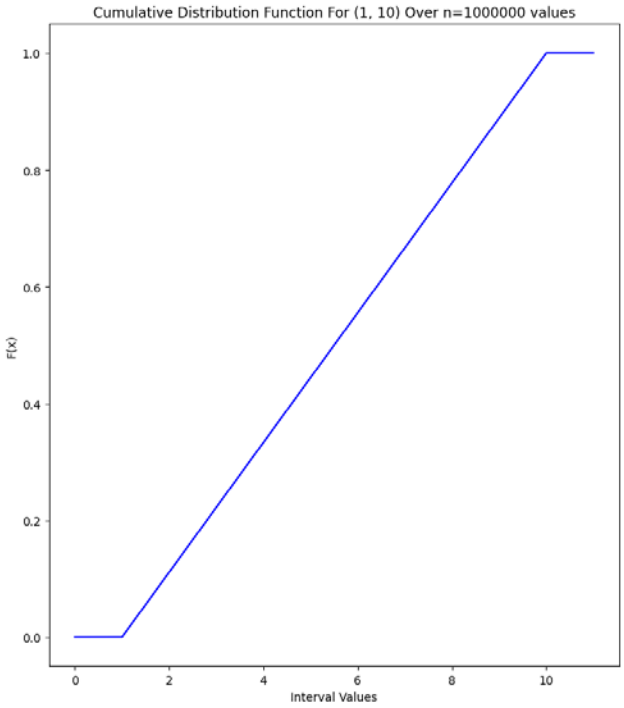




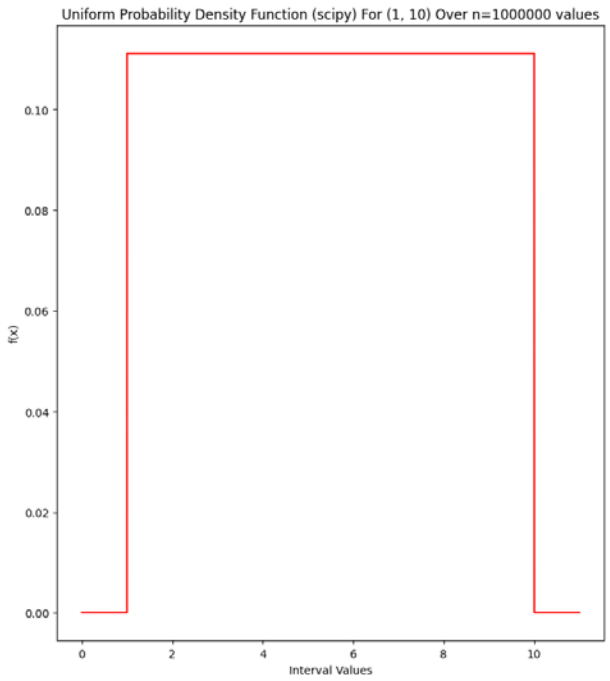
```
Results
-------
Uniform Implementation
Values on interval [1, 10]
PDF f(x): {0: 0, 1: 0, 2: 0.1111, 3: 0.1111, 4: 0.1111, 5: 0.1111, 6: 0.1111, 7: 0.1111, 8: 0.1111, 9: 0.1111, 10: 0, 11: 0}
CDF F(x): {0: 0, 1: 0, 2: 0.1111, 3: 0.2222, 4: 0.3333, 5: 0.4444, 6: 0.5556, 7: 0.6667, 8: 0.7778, 9: 0.8889, 10: 1, 11: 1}
```

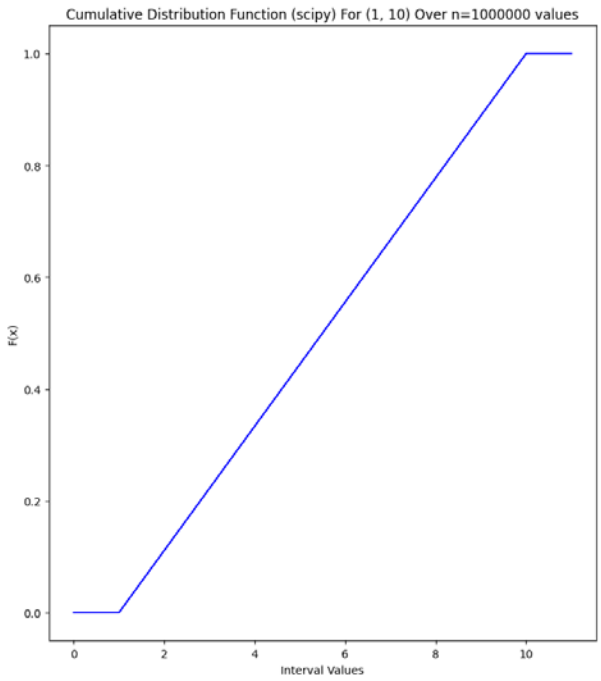
The following plot was obtained from selecting exactly 12 values, [0, 11] that includes just outside of our interval. As we can see from the shape of the graph, we should select more values for a true continuous distribution.





After we select *n=1,000,000* values, we begin to see the expected graph shape for PDF and CDF.





Using the *scipy.stats.uniform* library implementation of the PDF and CDF functions, we see the same graph shape as expected. The sharp drop for values 1 and 10 represent the nature of the continuous distribution, as P(a) = P(b) = 0 since it P(X=x) = 0 for the continuous distribution. In addition as shown by the two graphs, the relationship between the PDF and CDF can be observed as differing by a single order for the same values such that integrating the PDF f(x) will yield the CDF F(x). Likewise, obtaining the gradient of the CDF F(x) function will give us the PDF f(x).

Experiment Two:

```
Results
-------
Randomly selected points: 3
Circle Radius: 3
Circumference: 18.85
Semicircle length: 9.425

Number of trials: n = 100000
Probability of same semicircle: 0.753 (75265/100000)
```

In this last experiment three points are selected randomly on the circumference of the modeled circle. After n=100,000 trials the average probability in our sample of three randomly selected points lying on the same semicircle is found as *same_semi_count* / *n* at about *0.75*. This value is

in line with the theoretical value in which for the first point can be selected in *3* ways, and the following point has a fifty percent chance of being on the same semi-circle as the first, and the last point has fifty percent chance as well. This yields a theoretical value of 3 * 0.5 * 0.5 = 0.75 as modeled in the results of this experiment.

Conclusion

Performing this laboratory was a hands-on way of being able to apply the fundamental concepts of the uniform distribution of a continuous random variable. We have previously studied the probability density function and cumulative distribution function, but it was useful to take these abstract equations and implement a simulation using these concepts.

Previous laboratories we have done have use matplotlib in the past, however prior to this laboratory subplots have not been used. It was useful to consult the available documentation to see how to use the different functions and become a bit more familiar with the data visualization aspect as well. By plotting the results of the experiment it is far more clear to see the relationship between the probability density function and the cumulative distribution function.

Appendix

Experiment 1(a) Implementation:

```
'''
A random variable that is uniformly distributed over the interval (a, b) follows
the probability density
function (pdf) given by:
    f(x; a, b) = 1/(b-a);
    a < x < b

The cumulative distribution function (cdf) for a uniform random variable is:
    F(x) = {
                0           : x ≤ a
                (x-a)/(b-a) : a < x < b
                1           : x ≥ b
            }
Python implementation for (a, b) = (1, 10)
'''

# Utility functions implementing the above equations
def get_uni_pdf(a, b):
    return round(1/(b-a), 4)
def get_uni_cdf(x, a, b):
    return round((x-a)/(b-a), 4)

def uniform_eq(a=1, b=10, n=1000000):
    x_vals = np.linspace(a-1, b+1, n)
    # PDF map values to uniform distribution f(x)
    pdf_pcts = {}
    pdf_val = get_uni_pdf(a, b)
    for x in x_vals:
        if x > a and x < b: pdf_pcts[x] = pdf_val
        else: pdf_pcts[x] = 0

    # CDF map values to uniform distrubtion F(x)
    cdf_pcts = {}
    for x in x_vals:
        if x > a and x < b: cdf_pcts[x] = get_uni_cdf(x, a, b)
        elif x <= a: cdf_pcts[x] = 0
        elif x >= b: cdf_pcts[x] = 1

    # Output results
    print("Results\n-------")
    print("Uniform Implementation")
```

```
    print("Values on interval (" + str(a) + ", " + str(b) + ")")
    print("Number of values: n =", n)
    # print("PDF f(x):", pdf_pcts)
    # print("CDF F(x):", cdf_pcts)

    # Plot PDF and CDF horizontally
    plt.subplot(1,2,1)
    plt.title("Uniform Probability Density Function For (" + str(a) + ", " + str(
b) + ") Over n=" + str(n) + " values")
    plt.plot(pdf_pcts.keys(), pdf_pcts.values(), 'r-')
    plt.xlabel("Interval Values")
    plt.ylabel("f(x)")

    plt.subplot(1,2,2)
    plt.title("Cumulative Distribution Function For (" + str(a) + ", " + str(b) +
 ") Over n=" + str(n) + " values")
    plt.plot(cdf_pcts.keys(), cdf_pcts.values(), 'b-')
    plt.xlabel("Interval Values")
    plt.ylabel("F(x)")
    plt.show()

uniform_eq()
```

Experiment 1(b) Implementation:

```
'''
Implementation using scipy.stats.uniform pdf() and cdf() functions
over specified n evenly spaced values
'''
def uniform_stats(a=1, b=10, n=1000000):
    x = np.linspace(a-1, b+1, n)
    pdf_y = uni.pdf(x, a, b-a)
    cdf_y = uni.cdf(x, a, b-a)

    # Plot PDF and CDF horizontally
    plt.subplot(1,2,1)
    plt.title("Uniform Probability Density Function (scipy) For (" + str(a) + ",
" + str(b) + ") Over n=" + str(n) + " values")
    plt.plot(x, pdf_y, 'r-')
    plt.xlabel("Interval Values")
    plt.ylabel("f(x)")

    plt.subplot(1,2,2)
    plt.title("Cumulative Distribution Function (scipy) For (" + str(a) + ", " +
str(b) + ") Over n=" + str(n) + " values")
    plt.plot(x, cdf_y, 'b-')
    plt.xlabel("Interval Values")
    plt.ylabel("F(x)")
    plt.show()

uniform_stats()
```

Experiment 2 Implementation:

```
'''
Points are selected at random from the circumference of a circle.
Simulate the probability that the three points lie on the same semicircle
'''
# All points are are on the same semi circle if the distance from the middle
# point to the min and max point is no more than the semicricle length
def same_semi(points, semi_len):
    points.sort()
    mid = int(len(points) / 2)
    return points[mid] - points[0] < semi_len and points[-
1] - points[mid] < semi_len

def semi_circle(num_points=3, r = 3, n=100000):
    # Calculate the circumference, and length of a semicircle
    circumference = 2 * np.pi * r
    semi_len = circumference / 2
    same_semi_count = 0

    # Perform n trials of randomly selecting num_points
    for _ in range(n):
        # Randomly select three values on the circumference, assuming
        # 12 o'clock position as 0 with increasing values moving clockwise
        # rand_pts = np.random.uniform(0, 360, num_points)
        rand_pts = np.random.uniform(0, circumference, num_points)
        if same_semi(rand_pts, semi_len): same_semi_count += 1
    p_same_semi = round(same_semi_count / n, 3)

    # Output results
    print("Results\n-------")
    print("Randomly selected points:", num_points)
    print("Circle Radius:", round(r, 3))
    print("Circumference:", round(circumference, 3))
    print("Semicircle length:", round(semi_len, 3))
    print("\nNumber of trials: n =", n)
    print("Probability of same semicircle:", p_same_semi,
    "(" + str(same_semi_count) + "/" + str(n) + ")")

semi_circle()
```