Matthew Horowitz

A20377155

CS 430- Section 5 (Online)

Girish Rajani-Bathija

A20503736

CS 430- Section 4 (Live)

Bhavesh Rajesh Talreja

A20516822

CS 430-Section 4 (Live)

Muhammad Sheheryar Qureshi

A20517229

CS 430-Section 4 (Live)

Homework Two

QUESTION ONE:

A.

The continuous uniform distribution is a probability distribution in which each element in the domain has an equal chance of selection, yielding a PDF that is a constant. As such, the probability density function (PDF) of the uniform distribution is defined as:

$f(x) = \frac{1}{b-a}$, for $0 \leq a \leq b$

And where $f(x) = 0$, for all values outside interval a and b.[1]

The PDF of the uniform distribution gives information about the likelihood that an element exists for a specific value. However, what it does not do is tell the probability of selecting an event, given that selecting a specific value from a continuous distribution is immeasurable.[2] As such, it is more useful to discuss probabilities that an element is selected is at most some value, such that a measurement of area, under the PDF is given from a to X (the value of some element in the interval (a,b)). In order to do this, the cumulative distribution function (CDF), must be calculated and is found by taking the integral of the PDF.[3]

Below are the calculations presented for finding the CDF.

$$F(X) = \int_a^x \frac{1}{b-a} dx$$

$$\frac{1}{b-a}x - \frac{1}{b-a}a = {}^{x-a}/_{b-a}$$

, thus yielding the CDF of the uniform distribution:

$$\mathrm{F(x)} = \frac{x-a}{b-a}$$

, where $a \geq 0$ and $b > a$, and x falls on the interval $0 \leq a \leq X \leq b$.

Using the CDF formula probability for any uniformly distributed data X can be computed by using the above CDF equation.

[1] Wapole, R.E.; Myers, R.H.; Myers, S.L.; Ye, k (2017), Probability and Statistics for Engineers, Ninth Edition, Pearson: Boston, pp. 171.
[2] Ibid., pp. 89.
[3] Ibid., pp. 89.

B.

The inverse transform method is an important application in statistics that enables one to generate random samples for any distribution, in order to test probabilities. In general, finding perfectly random samples for complex distributions can be difficult, and as such, the inverse transform method is often utilized to achieve such results.[4]

The inverse transform process works as such:

1. Samples are drawn uniformly from 0 to 1.
2. Since probabilities are of interest, the CDF of the desired distribution is calculated
3. Then, using the desired distribution's CDF, the inverse of that CDF is found.
4. Uniformly generated samples are fed into the inverse of the CDF to find approximate X values for the desired distribution. [5]

Mathematical Justification:

Mathematically, the inverse transform from the uniform distribution does in fact approximate the probability of another distribution's CDF including the uniform distribution itself.

Using the proof from Devroye's text, it can be said that:

$P(F^{-1}(U) \leq x) = P(inf\ \{y: F(y) = U) \leq x)$

$P(U \leq F(x)) = F(x)$

Applying the bounds of $0 \leq U \leq 1$

$P(F(x) \leq u) = P(X \leq F^{-1}(u))$

$F(F^{-1}(u))=u$ [6]

[4] Field, A; Miles, J; Field, Z (2012), Discovering Statistics Using R, Sage: London.
[5]Xiannong, M. (2018), "Uniform Distribution and Inverse Transform", Bucknell University, Available at: https://www.eg.bucknell.edu/~xmeng/Course/CS6337/Note/master/node51.html.
Devroye, L (1986), Non-Uniform Random Variate Generation, Springer-Verlag: New York, Available at: http://luc.devroye.org/rnbookindex.html. pp. 27.
[6] Ibid., 28.

Calculating Inverse:

Since the inverse transform will be used to find X values of a uniform distribution on the interval $0 \leq a \leq X \leq b$, the first step is to find the inverse of the uniform distribution's CDF. Below are the calculations (note rather than solve for X, the calculation interchanges y and x and then solves for y).

Step 1: Set the CDF

$$\mathrm{F(x)} = \frac{x-a}{b-a}$$

Equal to y, and then interchange y and x.

Such that: $\mathrm{y} = \frac{x-a}{b-a}$ yields $\mathrm{x} = \frac{y-a}{b-a}$

Step 2: Solve for y

$x(b-a) = y-a$, yielding, $y = x(b-a)+a$

Step 3: Set y to equal the inverse of the CDF and x to be U, denoting a uniform distribution entry:

$F^{-1}(U) = U(b-a)+a$, where U is a randomly drawn uniform sample on the interval between (0,1), and whose output is the X value on the interval.

In the next section, a detailed analysis of generating U (uniform numbers is discussed). For this report, X values on arbitrary (a,b) intervals were calculated by feeding uniformly distributed numbers on the interval (0,1) into this inverse function, yielding the requisite X values.

C.

Python code, supplied with this report, was run to conduct the required analysis. The first part of this section details the steps taken to:

1. Generate three sets of 10,000 uniformly distributed random numbers on the interval (0,1) using the Python library.
2. Apply the three inverse transforms to each of the 10,000 uniformly distributed numbers.
3. They were then sorted using a bucket sort.
4. Three subsets were taken from each of the 10,000 values. Justification for this step will be given in the next section.
5. These subsets were then assigned to a uniform distribution whose X values will be calculated, each with different a and b bound values.
6. The inverse transform was then calculated, for these subsets.
7. The output from step 6 was plotted.
8. For theoretical analysis, (subsequent section), 3 extra sets of 10,000 uniformly distributed values were created using Python's library.
9. Each of these 10,000 values was fed into the CDF for uniform distribution, using the same a and b bounds used in step 4. The output was overlayed on top of the plots created in Step 6 to determine how close the inverse transform's observed approximation (empirical) to the theoretical findings obtained by actually running CDF calculations.

Justification for Subsets (k=50):

Given that the CDF of the uniform distribution is linear, and where the line represents the cumulative probability (F(x)=(P(X$\leq x$)) only two samples are needed, are needed to justify a line of displaying cumulative probability.[7] However, to ensure that the empirical findings of in this report hold, the expected linear trendline produced from the inverse transform outputs, large subsamples should be employed. As such, k=50 was selected. Furthermore, to ensure that the linear trendline holds, for each of the three 10,000 samples draw, 3 of these k=50 subsamples was created where the first sample includes the first 50 j elements (j=0-49), the second includes the "median" k samples (j=5000-5049) and the greatest 50 elements (j=9950-9999). In all, 9 subsamples were drawn, 3 per each of the original 10,000 samples.

Following this approach ensures that these samples, once transformed can be definitively approximated to the theoretical uniform distribution CDF.

Justification for Using Bucket Sort:

In order to plot the subsets of transformed uniformly distributed points, a sorting algorithm was needed. For this case, the best algorithm to use, to ensure that run time is minimized is the bucket sort. Below is the justification for using the sort (whose implementation and syntax can be found in the Python code supplied).

[7] Ibid.

Bucket sorts are an efficient sort, that on average generate an asymptotic complexity of O(n), provided the input data is uniformly distributed. [8]This section will now discuss how the bucket sort is implemented, and why the choice was made to implement this source.

The general idea behind a bucket sort is that data can be subdivided into buckets (or mini subarrays), and then the contents of each bucket are sorted, by calling an insertion sort on each of these buckets.[9]

This yields efficiencies as it enables the problem size to be subdivided into smaller problems, where the contents of each bucket are the subdivision.[10] However, to be efficient, each bucket should have a similar number of items compared to every other bucket. Otherwise, should the number of contents in each bucket be unbalanced, when the insertion sort is called on the bucket contents, some buckets will contain too many contents, resulting in an insertion sort that takes O(n^2) complexity, thus yielding an inefficient sort. As such, input must be uniformly distributed, to ensure that subdivisions are approximately equal sizes. Furthermore, to ensure that there is balance between the buckets, the number of buckets should equal the base of the digit system employed. For example, if using base 10, there should be 10 buckets.[11]

Since the input data is uniformly distributed, and since sample size is large, using a bucket sort is an ideal choice for sorting through the uniformly generated numbers.

Implementation of Bucket Sort:

As it can be seen in the Python code, the first step is to identify the number of buckets to be used. To simplify the process, the sort was conducted on each of the sets of uniformly distributed data, with 10,000 samples, prior to being transformed using the inverse transform method. The sort was conducted before the transformation, as it is easier to establish buckets on samples generated between 0 and 1. This is because all one has to do is multiply each sample by 10, resulting in the tenth decimal becoming an integer, where each integer corresponds with a bucket—10 buckets are created ranging from 0 to 9--. Using this leading integer, one can now appropriately place a sample into the correct bucket.

Conducting the sort before applying the inverse transform does not affect the quality of the mathematical outcome. This is because the CDF of the uniform distribution is linear. As a result, the sorted pre-transformed data perfectly map to sorted post-transformed data, and where the ordered properties are maintained during transform.

[8]Corman, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C (2009), Introduction to Algorithms, Third Edition, MIT: Cambridge, MA, pp. 202.
[9] Ibid., pp. 201-202.
[10]Ibid., pp. 202.
[11] Field, A; Miles, J; Field, Z (2012), Discovering Statistics Using R, Sage: London..

Sort that Minimizes Time Complexity: Bucket Sort:

Bucket sorts were used in this analysis because the sorting algorithm minimizes time complexity. As noted above, provided the input data is uniformly distributed, this algorithm is incredibly efficient, with an asymptotic time complexity of O(n).[12]

Below is a table that examines the other sorts, with comments comparing them to bucket sorts[13]

SORT	TIME COMPLEXITY	COMMENTS
Insertion Sort	Best Case: O(n), Worst and General Case $O(n^2)$	The general case is $O(n^2)$, which is worse than O(n) bucket sort. In addition, this sort is used as part of bucket sort.
Merge Sort	Best Case, Average Case, Worst Case: $\Theta(n\log_2 n)$	Merge Sort results in $\Theta(n\log_2 n)$ time complexity which is worse than a bucket sort's O(n)
Heap Sort	Best/Average Case $O(n\log_2 n)$	Average case of $O(n\log_2 n)$ is worse than the O(n) bucket sort.
Quick Sort	Best/Average Case $O(n\log_2 n)$, worst case: $O(n^2)$	Best, avg and worst case all worse than O(n) bucket sort.
Comparison Sort	Best/Average Case: $O(n\log_2 n)$	Best, avg, and worst case all worse than O(n) bucket Sort
Counting Sort	Best/Avg/Worst Case: O(n)	Same time complexity as bucket sort, but due to the large range of numbers and decimals in the uniform distribution, time complexity not as efficient
Radix Sort	Best/Avg/Worst Case: O(n)	Same time complexity as bucket sort, but due to the large number of bits (digits) present, this sort is not as efficient as the bucket.

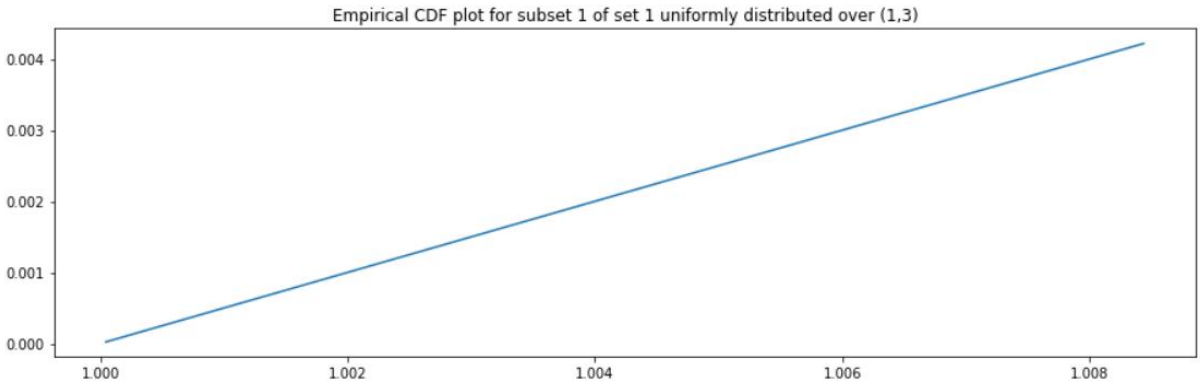
[12] Corman, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C (2009), Introduction to Algorithms, Third Edition, MIT: Cambridge, MA, pp. 202.
[13] Ibid.

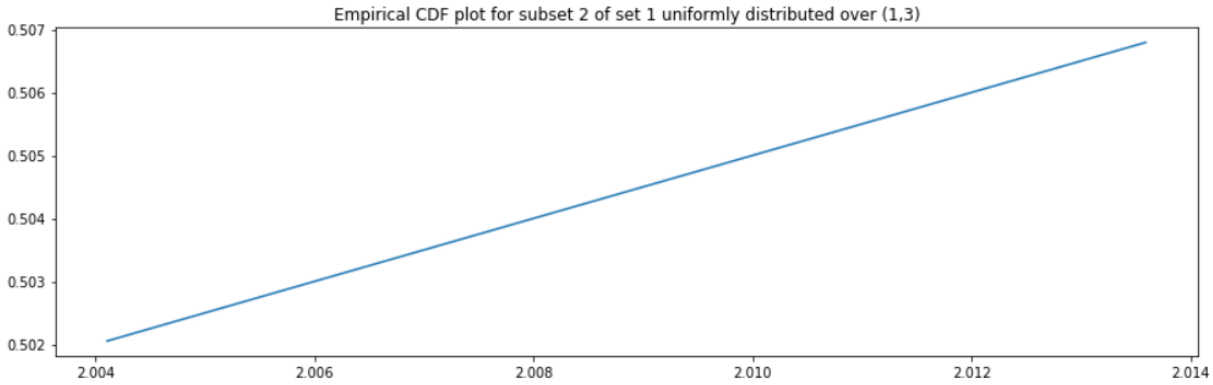
D.

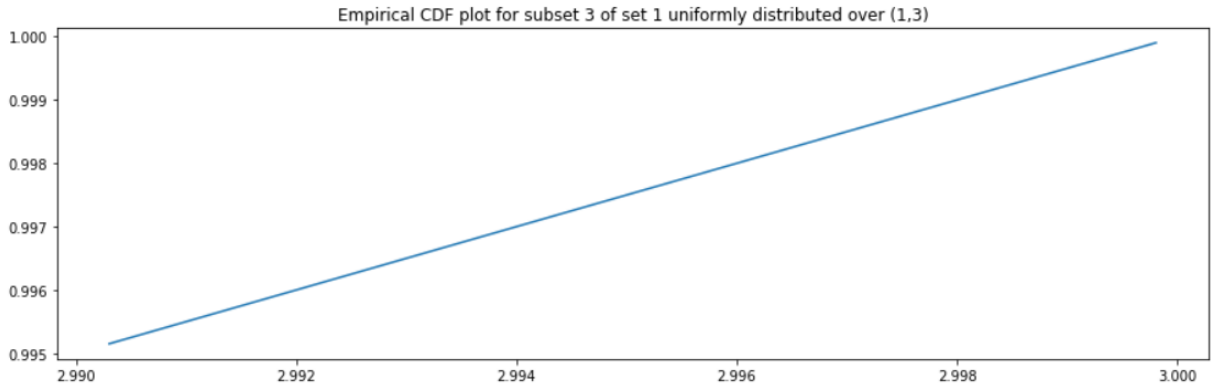
Part One: Plotting empirical outputs

Below are the graphical outputs. For each set 1-3, the three subsets are plotted, in which subset 1, corresponds to values j=0-49, subset 2, j=5000-5049, subset 3, j=9950-9999.

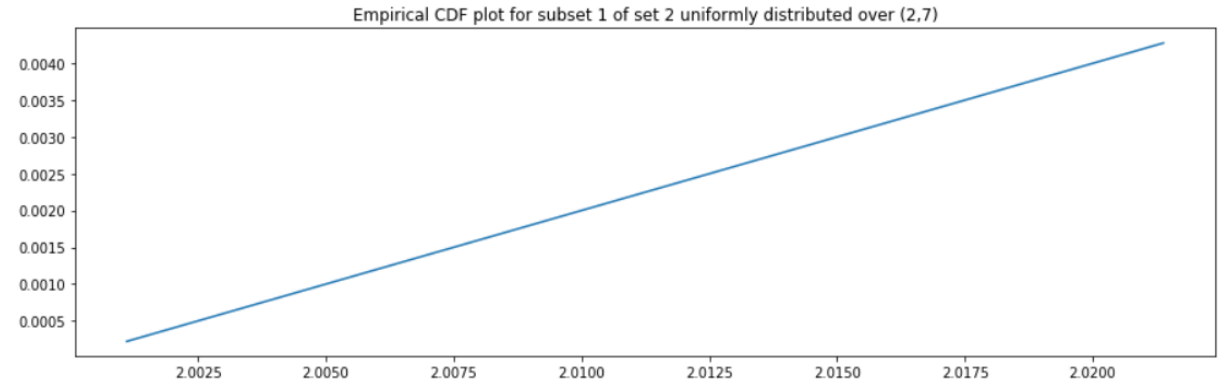
Set 1: a= 1 b= 3

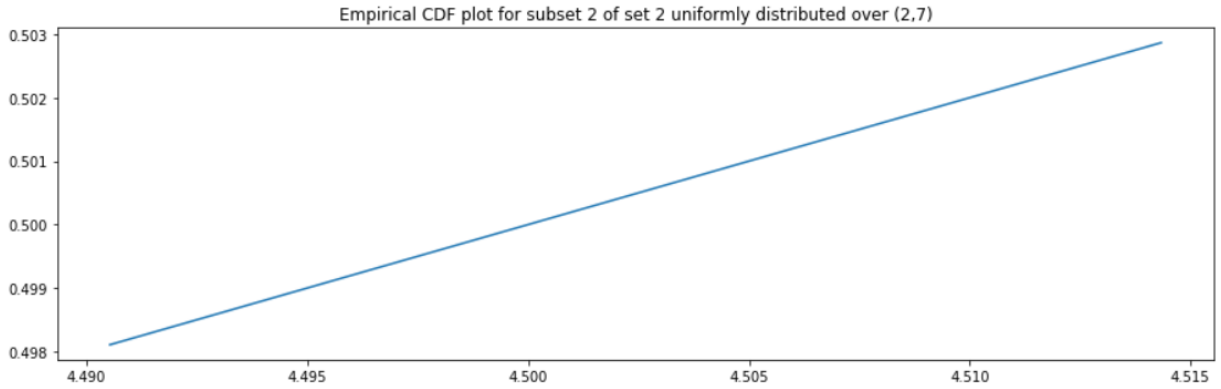


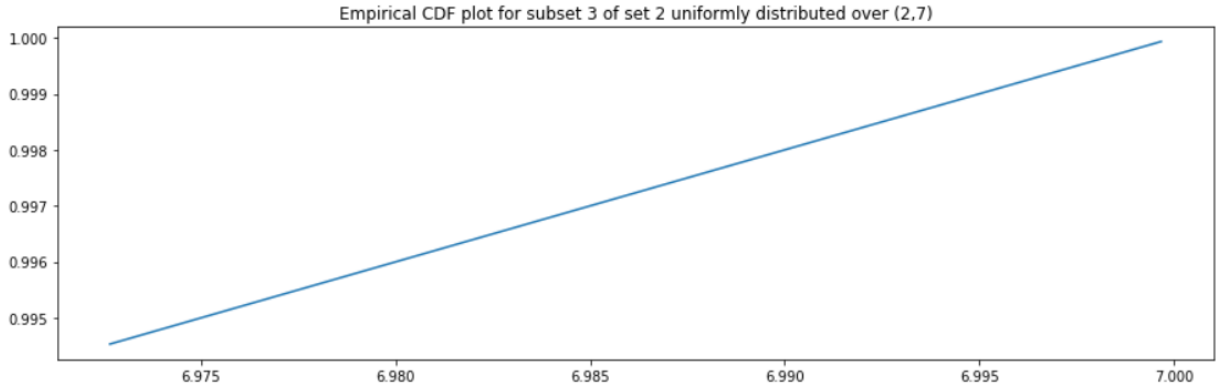




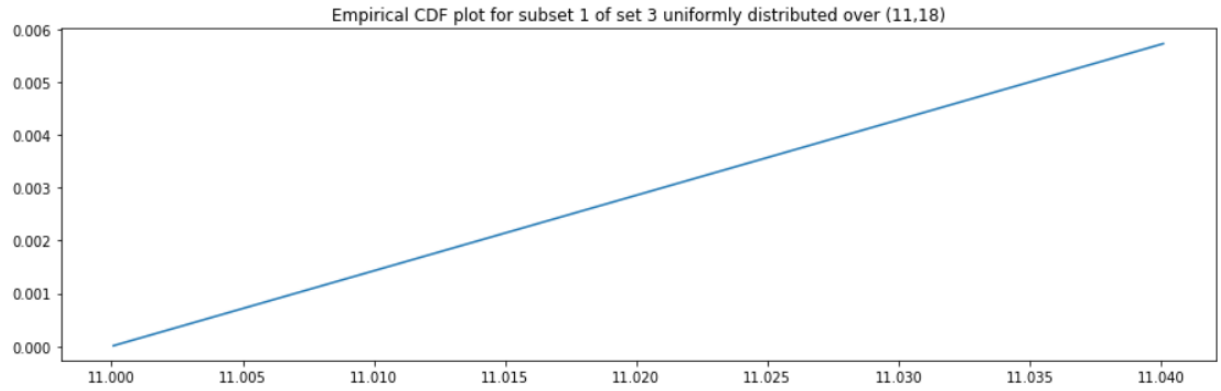
Set 2: a=2 b=7

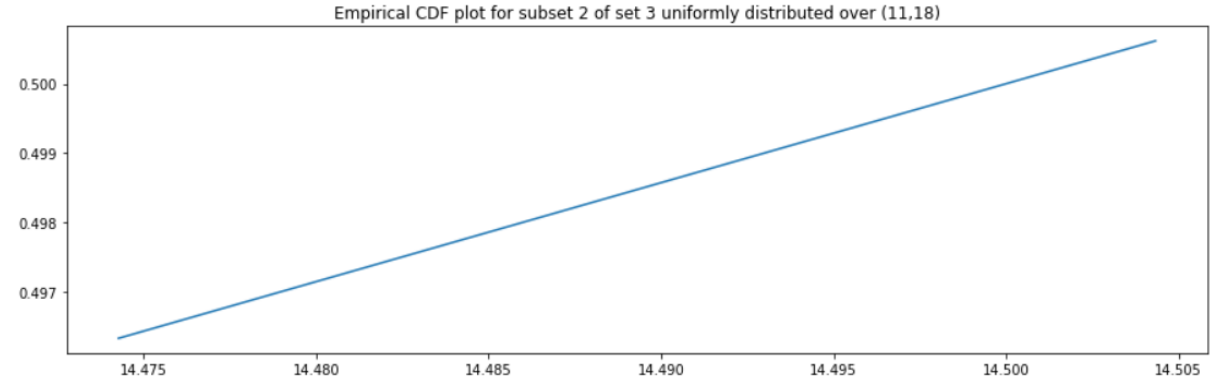

Empirical CDF plot for subset 1 of set 2 uniformly distributed over (2,7)
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
2.0025
2.0050
2.0075
2.0100
2.0125
2.0150
2.0175
2.0200

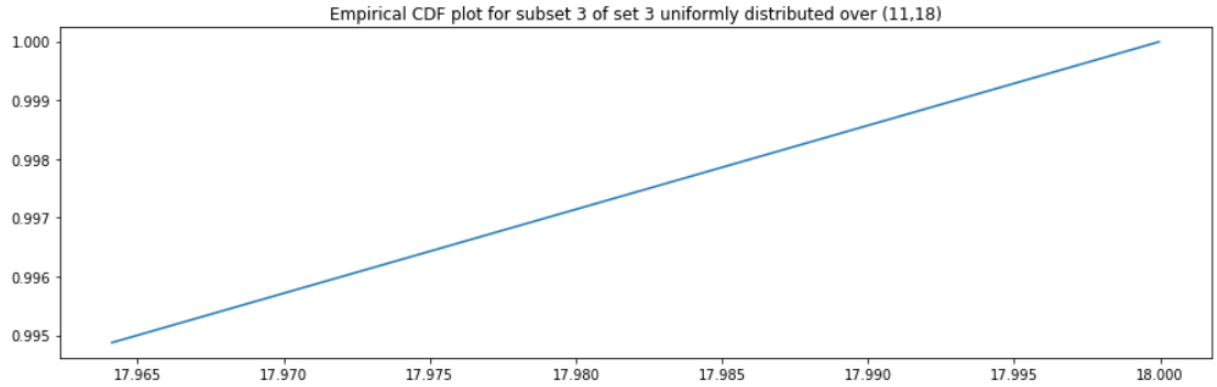

Empirical CDF plot for subset 2 of set 2 uniformly distributed over (2,7)
0.503
0.502
0.501
0.500
0.499
0.498
4.490
4.495
4.500
4.505
4.510
4.515


Empirical CDF plot for subset 3 of set 2 uniformly distributed over (2,7)
1.000
0.999
0.998
0.997
0.996
0.995
6.975
6.980
6.985
6.990
6.995
7.000

Set 3: a= 11 b= 18

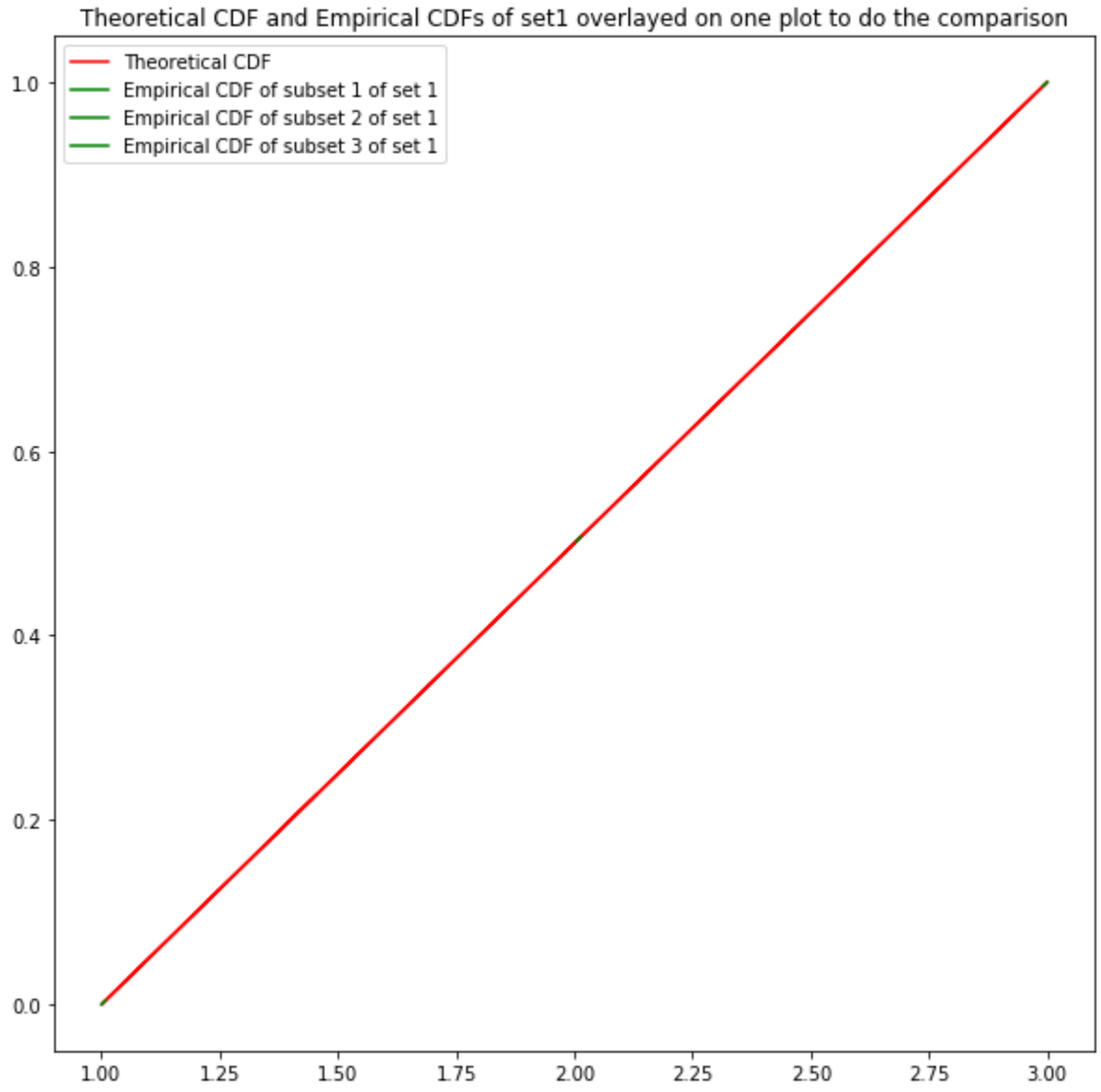

Empirical CDF plot for subset 1 of set 3 uniformly distributed over (11,18)
0.006
0.005
0.004
0.003
0.002
0.001
0.000
11.000
11.005
11.010
11.015
11.020
11.025
11.030
11.035
11.040


Empirical CDF plot for subset 2 of set 3 uniformly distributed over (11,18)
0.500
0.499
0.498
0.497
14.475
14.480
14.485
14.490
14.495
14.500
14.505

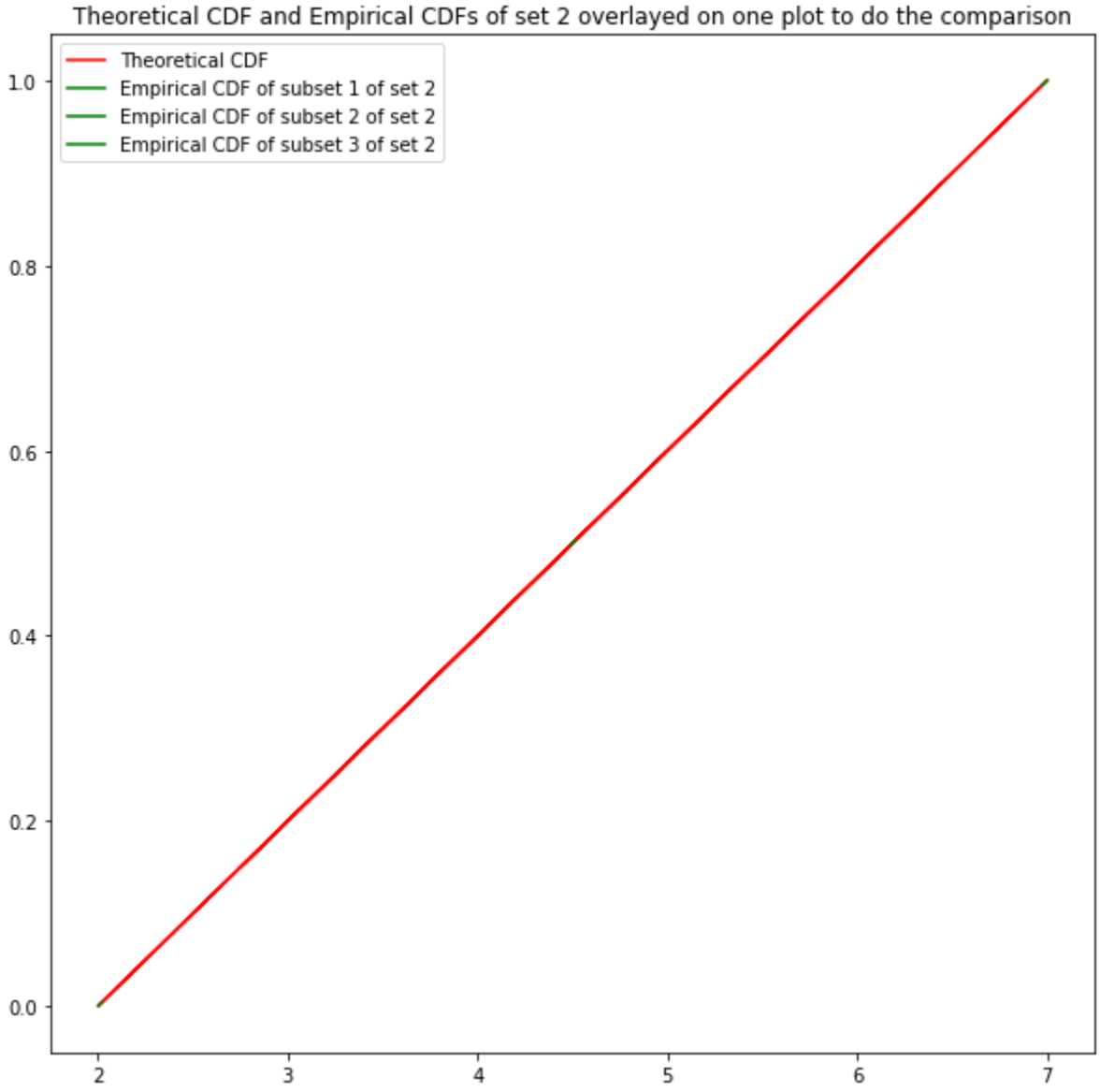

Empirical CDF plot for subset 3 of set 3 uniformly distributed over (11,18)
1.000
0.999
0.998
0.997
0.996
0.995
17.965
17.970
17.975
17.980
17.985
17.990
17.995
18.000

Part Two: Plotting Empirical Overlayed on Theoretical (CDF of Uniform)

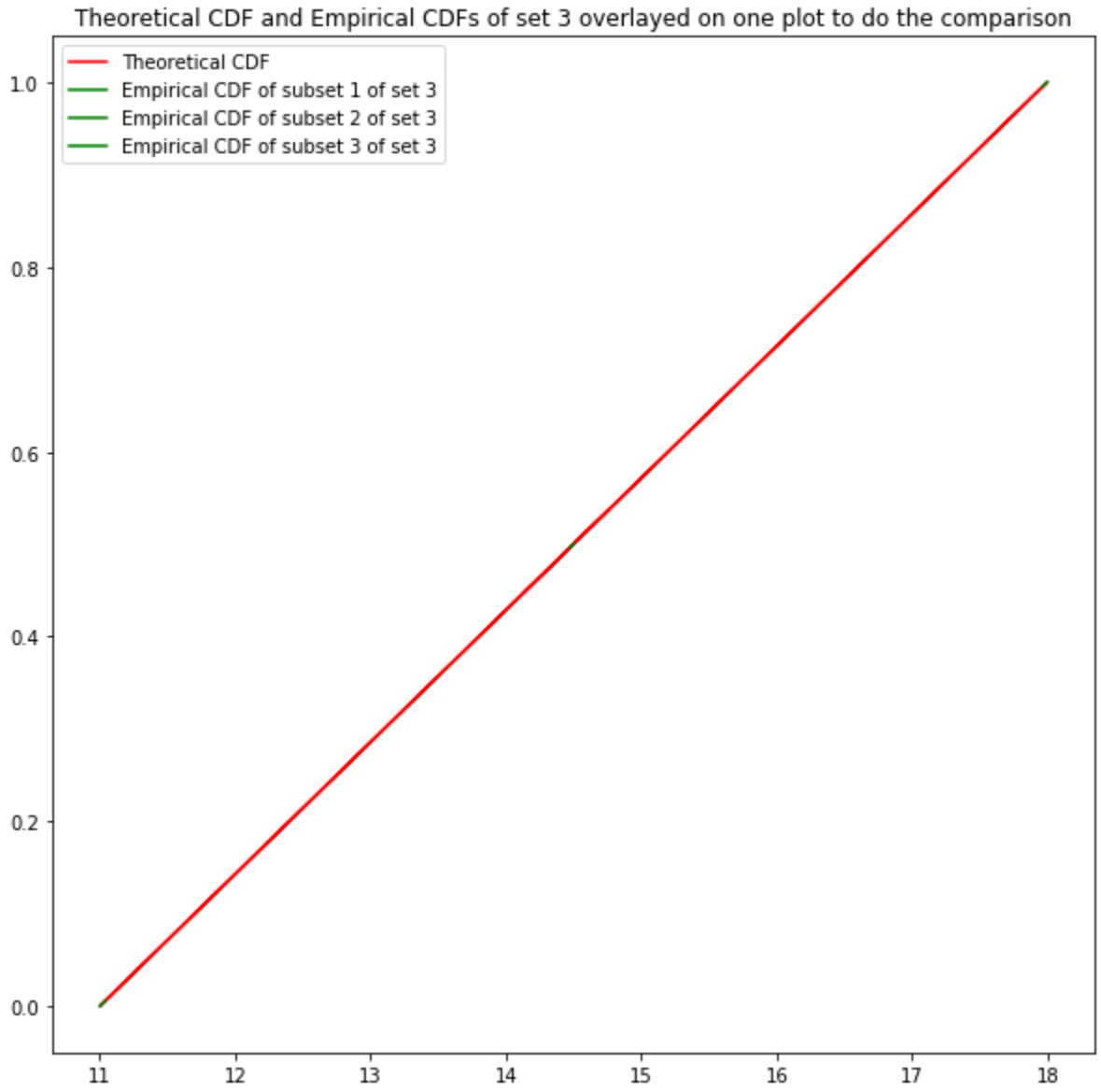
Set 1: a=1 b=3



Set 2: a=2 b=7


Theoretical CDF and Empirical CDFs of set 2 overlayed on one plot to do the comparison
Theoretical CDF
Empirical CDF of subset 1 of set 2
Empirical CDF of subset 2 of set 2
Empirical CDF of subset 3 of set 2
1.0
0.8
0.6
0.4
0.2
0.0
2
3
4
5
6
7

Set 3: a=11 b=18



Discussion:

For each set, the empirical subsets (j=0-49, j=5000-5049, j=9950-9999) were overlayed with a theoretical sample. The theoretical sample is comprised of three additional uniformly generated sets over the following intervals:

Theoretical Set 1: a=1 b=3

Theoretical Set 2: a=2 b=7

Theoretical Set 3: a=11 b=18

These sets were then imputed into the CDF formula for uniform distributions and plotted against the empirical findings from the inverse transform to see how well they aligned. In the above plots, the empirical outputs are in green, while the theoretical are in red. Using the three subsets for each of the set plots, it becomes apparent that employing the inverse transform does in fact approximate the CDF to a very high degree.

Division of Work:

The team worked together to create this report. Matt worked on Part C and did the mathematics for the inverse transform, Bhavesh did the Python code, Girish worked on Part A, and Sheheryar worked on Part B. Together the team then reviewed the sections of the report and worked on final edits.

References:

Corman, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C (2009), Introduction to Algorithms, Third Edition, MIT: Cambridge, MA.

Devroye, L (1986), Non-Uniform Random Variate Generation, Springer-Verlag: New York, Available at: http://luc.devroye.org/rnbookindex.html.

Field, A; Miles, J; Field, Z (2012), Discovering Statistics Using R, Sage: London.

Wapole, R.E.; Myers, R.H.; Myers, S.L.; Ye, k (2017), Probability and Statistics for Engineers, Ninth Edition, Pearson: Boston.

Xiannong, M. (2018), "Uniform Distribution and Inverse Transform", Bucknell University, Available at: https://www.eg.bucknell.edu/~xmeng/Course/CS6337/Note/master/node51.html.