Jeffrey Rodriguez 110733867

AMS 326

Report 2

3/2/2018

2.1

The initial task is to generate 100,000,000 random numbers in the range (-10,10). Partitioning this into 20 subintervals, we first wish to see what percentage of total numbers compose each of these intervals. Then, we partition the numbers into groups of 20 and sum over each, referring to these as g numbers. We then extend the interval to a wider range, (-200,200) for the sums. Taking these sums, and seeing what percentage they compose for each interval, we can then plot a histogram of the percentages. Ideally, this histogram will appear normally distributed.
After this, we again generate random numbers, this time $M = 20,000,000$, and perform a Box-Mueller transformation on them to obtain values which will also follow a normal distribution.
To generate the numbers, two arbitrarily small values between 0 and 1 are chosen, such that they do not sum to 0 or 1. They are then applied to the recursive formula
$x_{i+1} = (x_i + x_{i-1}) \mod 1$. Once a value is created, it is stored in another variable, $y = 20x - 10$, so that we can examine numbers in the $(-10, 10)$ range instead. Furthermore, a counter is kept, so that we can calculate the g numbers for every 20 x values. The algorithm used for this program is as follows:

```
1. Set $i,\ j \in \mathbb{Z}$ equal to 0.
2. while $i < N = 100,000,000$, and while $j < 20$, generate a new random number
   $x$, and corresponding $y$ value.
3. Assign $y$ to a bin based on the rounded down value of $\frac{y-(-10)}{1}$, where -10 is
   the minimum value y can attain, and 1 is the bin width.
4. Increment $g$ by $y$ until $j$ reaches 20.
5. Assign $g$ to a bin based on the rounded down value of $\frac{g-(-200)}{20}$, with similar
   reasoning as in the case of the $y$ bins.
```

The algorithm for the second part of this was similar, except that we work with two random numbers x and y, with the z transforms as:

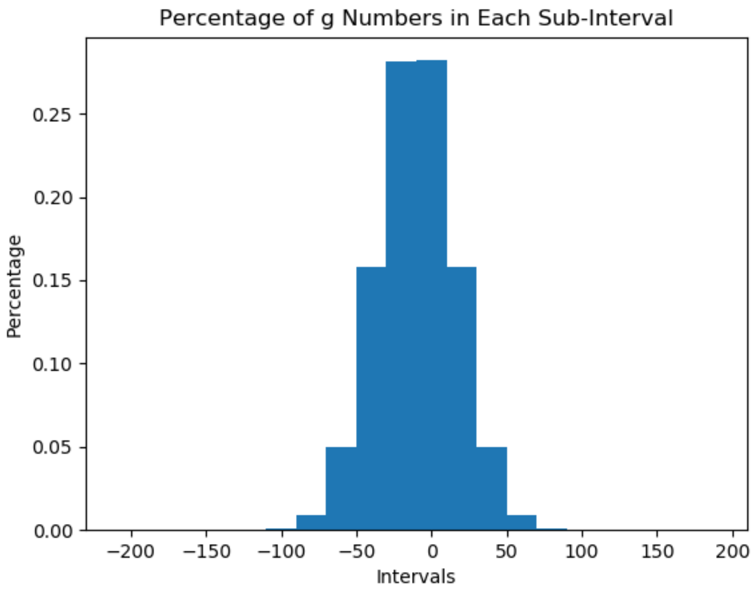
$$z_1 = \sqrt{-2\ln x}\cos(2\pi y)$$

$$z_2 = \sqrt{-2\ln x}\cos(2\pi y)$$

Some values appear to fall outside the range (-5,5) and we refer to these as ghosts. The

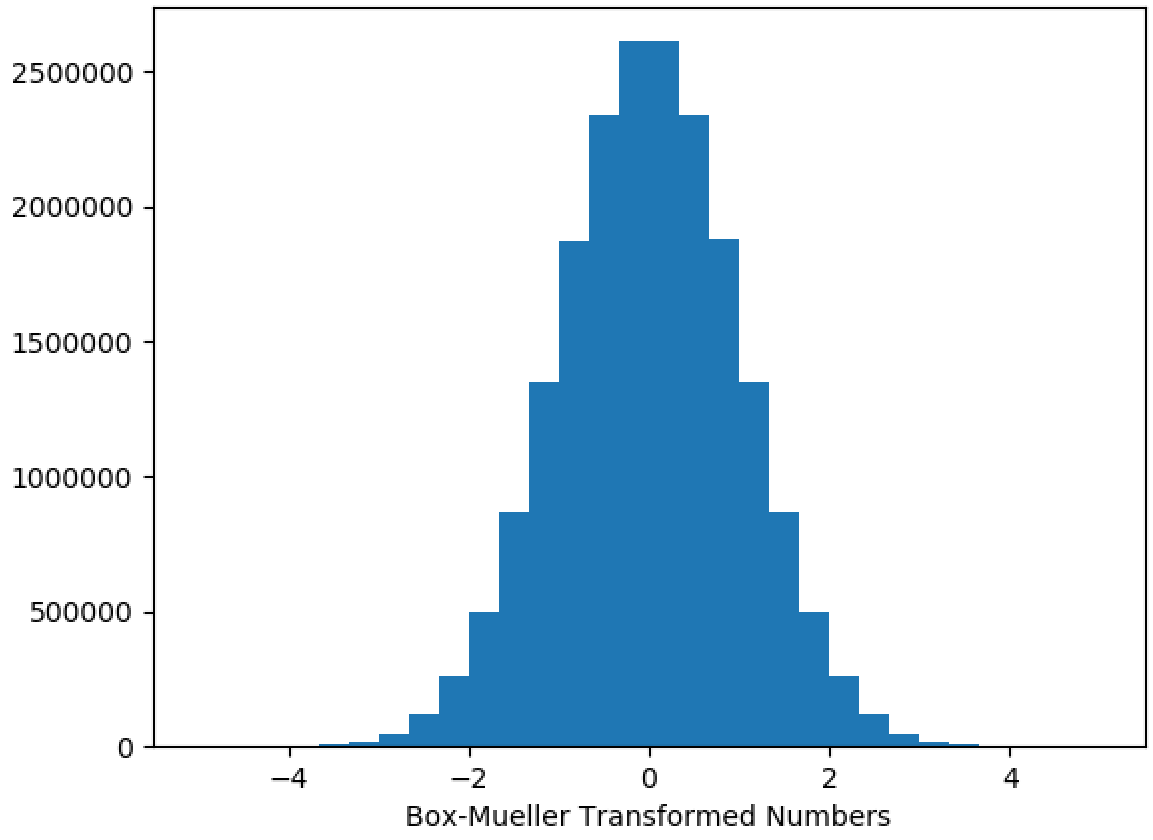
remaining values will be plotted in a histogram.
Both of these were evaluated in Python, in the attached program '326HW2-1.py'. For the first algorithm, initial values for the random generator are 0.396184 and 0.265781, but as stated previously, any two values can be used so long as the sum lies in the interval (0,1). The y values take up roughly 5% of each bin, with arithmetic average of roughly 0.05. Of course, we can expect an equal distribution of percentages as $N \to \infty$ as stated in the Law of Large Numbers.
A table was generated in Python and shows the percentage of g values in each partition of the interval (-200,200). This along with a histogram of the percentages of g values contained in each bin can both be found on the following page.

```
+----------+------------+
| Interval | Percentage |
+==========+============+
| -200     | 0          |
+----------+------------+
| -180     | 0.00000    |
+----------+------------+
| -160     | 0.00000    |
+----------+------------+
| -140     | 0.00001    |
+----------+------------+
| -120     | 0.00009    |
+----------+------------+
| -100     | 0.00099    |
+----------+------------+
| -80      | 0.00921    |
+----------+------------+
| -60      | 0.04995    |
+----------+------------+
| -40      | 0.15804    |
+----------+------------+
| -20      | 0.28158    |
+----------+------------+
| 0        | 0.28204    |
+----------+------------+
| 20       | 0.15780    |
+----------+------------+
| 40       | 0.04999    |
+----------+------------+
| 60       | 0.00919    |
+----------+------------+
| 80       | 0.00102    |
+----------+------------+
| 100      | 0.00009    |
+----------+------------+
| 120      | 0.00001    |
+----------+------------+
| 140      | 0.00000    |
+----------+------------+
| 160      | 0          |
+----------+------------+
| 180      | 0          |
+----------+------------+
```



We see that this does in fact appear to follow a normal distribution. For smaller sized n values, (such as 100,000 or less), the program is quite fast, but after that it takes noticeably longer.
For the second, we use initial values 0.396184 and 0.265781 for one random variable, and 0.221573 and 0.665431 for the other. After applying this function, we see that the ghost values are (after sorting): (-5.430471, -5.278879, -5.262297, -5.241019, -5.229572, -5.071456,

5.015548, 5.099233, 5.169329, 5.186372, 5.194119, 5.213180, 5.271024, 5.431485, 5.492506). The histogram for the remaining values can be seen below:



The graph does indeed appear to be normal distributed, and moreso centered around 0 than the first plot. Due to M being much smaller than N, this function also performs much faster.

2.3

Next, we look at the NASDAQ indices during the first full week of February 2018 (2/5 to 2/9). This was during a period where the stock market underwent some sharp declines. We obtained the values from Yahoo!Finance[1] as seen in the following table:

Date	x	y
2/5/2018	1	6967
2/6/2018	2	7115
2/7/2018	3	7051
2/8/2018	4	6777
2/9/2018	5	6874

With these values, we wish to find two curves connecting the data points. One via interpolation with a 4th order approximation, the other via a quadratic curve. To obtain the interpolation curve, we set $P(x_i) = y_i$, with $P(x_i)$ being a polynomial curve, and $0 \leq i \leq 4$. We write $P(x_i) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$. Then with 5 x values and 5 y values, we can write this in matrix form as the equation:

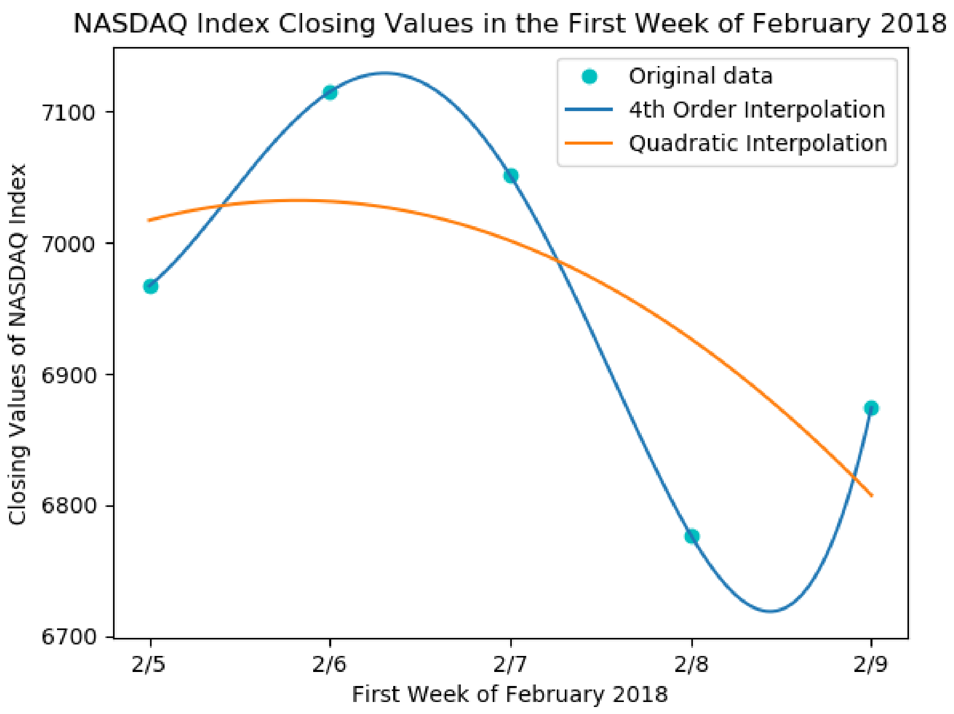
$$\begin{pmatrix} x_0^4 & x_0^3 & x_0^2 & x_0 & 1 \\ x_1^4 & x_1^3 & x_1^2 & x_1 & 1 \\ x_2^4 & x_2^3 & x_2^2 & x_2 & 1 \\ x_3^4 & x_3^3 & x_3^2 & x_3 & 1 \\ x_4^4 & x_4^3 & x_4^2 & x_4 & 1 \end{pmatrix} \begin{pmatrix} a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

Here, the x_i, y_i values match with those listed on the table. We can then solve for a by setting $a = X^{-1}Y$.
Next, we perform a quadratic regression in order to fit a curve on the data set. To do this, we set $Y_i = b_0 + b_1x_i + b_2x_i^2 + \epsilon_i$. Or rewrite as $\sum_{i=0}^4 \epsilon_i^2 = \sum_{i=0}^4 (Y_i - (b_0 + b_1x_i + b_2x_i^2))^2$. To solve for the b values, we take the partial derivatives of ϵ^2 with respect to each b_i, set them equal to 0. With three equations and three unknown constants to solve for, we can express this as the following matrix:

$$\begin{pmatrix} 5 & \sum_{i=0}^4 x_i & \sum_{i=0}^4 x_i^2 \\ \sum_{i=0}^4 x_i & \sum_{i=0}^4 x_i^2 & \sum_{i=0}^4 x_i^3 \\ \sum_{i=0}^4 x_i^2 & \sum_{i=0}^4 x_i^3 & \sum_{i=0}^4 x_i^4 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^4 y_i \\ \sum_{i=0}^4 y_ix_i \\ \sum_{i=0}^4 y_ix_i^2 \end{pmatrix}$$

As with the interpolation polynomial matrix, we invert this above matrix and multiply it by the y vector to get the regression coefficients.
The matrix inversion and multiplication computations were performed via Python's numpy package. For the interpolation curve, we obtain the function $f_1(x) = 24.125x^4 - 240.917x^3 + 736.375x^2 - 736.583x + 7184$. For the quadratic curve, we obtain $f_2(x) = -22.286x^2 + 81.314x + 6958$.
Finally, we plot these lines, together with the initial data points. This is done through Python's MatPlotlib package and obtain the following graph.

[1]https://finance.yahoo.com/quote/%5Eixic/history/



We can see from this graph that the interpolation curve is smooth and goes through the points as expected, although the quadratic curve does not pass through any points. A higher ordered regression, such as 3rd or 4th order with more maxima/minima points may help lead to a better fit.
The code used for this problem can be seen in the attached file, 'HW2-3.py', and runs quite fast, especially due to the very small number of data points.