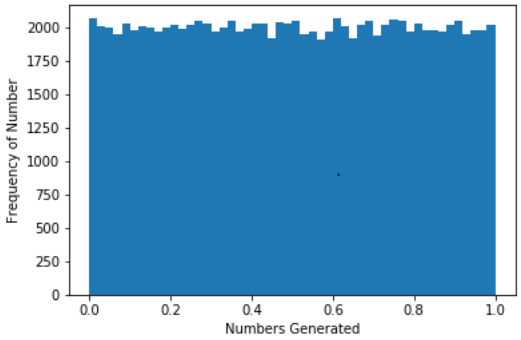
The First part of the lab is done using Python in Jupyter Notebook, and the second part by c++.

PART 1

Note :- Number of bins used to plot figures = 50.

1. See the Details in Jupyter Notebook.

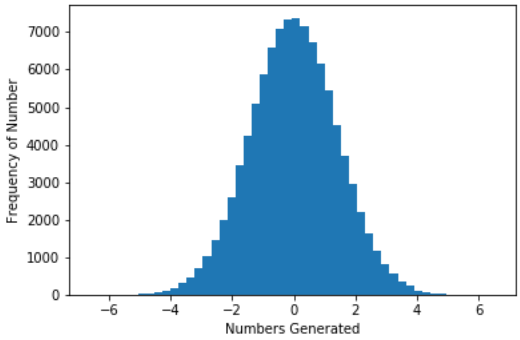
2.



A set of 100000 Random Number is Generated for this exercise which has mean of 0.495 and a standard deviation of 0.2888. The Maximum and Minimum number of this set is 0.000001 and 0.999998, with 25%,50% and 75% quantiles at 0.24957,0.498678 and 0.750421 (For reference please check the notebook Part 1, Section 2).

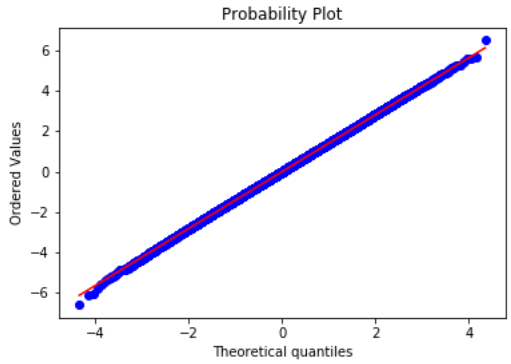
3. See the Details in Jupyter Notebook.

4.



For this exercise we were supposed to transform the random generated data to a normal distribution with mean 0 and Standard Deviation 1.414. The analysis of normal distribution gave the mean of -0.003114 ~ 0, and a Standard Deviation 1.4179, which are nearly equivalent to theoretical arguments.

5.



A Normal distributed data is supposed to follow the line Y=X, and from the graph I can state that data generated after inverse mapping is indeed normally distributed.

6.

For this problem the K-S test has the following null hypothesis: data should have a normal distribution. The P-value obtained for the data is 0.821 which strangely suggest that this hypothesis is right.

Part 2.

The code for this file is in main.cpp, main.cpp has four function 1) get_number, to generate random number. 2) Print_Num, to print the random number .3) chi_square, To calculate the chi square value and print it to console window. 4). Write_file, to write the number generated and observe frequency in an interval in a file (O_i). For all the calculation I have used Number of Subinterval: M = 10. Function get_Number and chi_square is parallelized using OpenMP.

1. See Main.Cpp

2. See Main.Cpp

3.

N, Number of Random Numbers	Chi_Squared (Using C++)	Chi_Squared (Using Python) and P-Value	Number of Threads Used
1000	6.78	6.78, 0.66	1
10000	5.286	5.28,0.808	1
100000	11.19	11.197,0.26	1

The files used for the calculation for N=1000 is: Number_1000 (1000 random number) and Observe_frequency_1000 (O_i for N=1000). Same for N=10000 and 100000. The Python code is in jupyter notebook for 3rd column.

4.

Number of Threads	Total Time (Random Number Generation + Counting Observation) in Sec		
	N = 1000	N = 10000	N = 1000000
1	0.0080666	0.0016095	0.122104
2	0.0006344	0.0012628	0.0749345
4	0.0009025	0.0012035	0.0512944
8	0.0013934	0.0018327	0.0560516

The Value of Chi Square from table 1 suggest that there is room for improvement for random number generation. The method used in this exercise can sometime produce random number which won't be ideal to use for some other important calculation. I have drawn this inference because changing N change chi square to a significant level, and I was expecting that as I increase N, the data will be distributed better. However, I am observing the quiet opposite of that.

As for parallelization, I observe that if N is small it doesn't affect the computational cost much, but for the large case i.e N = 1000000, I observe a computational cost reduction by more than half as I increased the threads from 1 to 4.

Instruction for compilation:

PYTHON: Simple run of jupyter notebook would do.

C++ : I used the command

g++ -fopenmp main.cpp => to compile

.a/.out => to see the result

export OMP_NUM_THREADS=1 => (to set number of threads, Change 1 to 2,4,8 for execution with different no of threads)

Note:- Please put the zip folder in your current working directory of jupyter notebook. Also I have commented out the call to write_file function from main program because file is already available.