How Random is "Random"?

Sunil K. Dhar and Varun Oberoi

Department of Mathematical Sciences and
Center for Applied Mathematics and Statistics
New Jersey Institute of Technology, Newark, NJ 07102

CAMS Report 0203-20, Spring 2003

Center for Applied Mathematics and Statistics

NJIT

How Random is "Random"?

Sunil K. Dhar and Varun Oberoi

Center for Applied Mathematics and Statistics
Department of Mathematical Sciences
New Jersey Institute of Technology

Abstract: Three random number generators are analyzed for data sets of 500 numbers. Their performances are evaluated through several tests. All generators create uniform numbers. The various results are then interpreted to find out which random number generator creates numbers that are in close agreement with the random theory. Using these results and interpretations some ideas are explored to improve on the randomness of numbers. A new test is proposed to check for randomness.

1. Introduction: Random number generators have always been in viewpoint of researchers as they use random numbers for several purposes. They are used for creating normal deviates, games and more recently cryptology and Internet securities.
Several applications now have in-built random number generators with a capacity of creating several thousand numbers.

Out of these, Microsoft Excel (1) and the random number generator provided with the g++ compiler for C++ (2) were used for the present work. A third generator (3) that was evaluated used a very common method of creating random numbers:

$$N_{i+1} = [Ni\,(2^n + 1) + C],\ \mathrm{mod}\ (2^{35}),$$

Where n is a positive integer between 2 and 34 and C is an odd integer. This method was proposed by Rotenberg (1960) and has been widely used ever since. Richard Kronmal has provided a more specific form of this generator in 1964, which he claims is better than most generators of the time. He mentions that, for n >10 shows, small serial correlations. He uses n = 7 even though it gives high serial correlation, but then Kronmal (1964) uses two variations of this formula and hopes to counterattack the high autocorrelation problem. This method was used by a function proposed in 'Numerical Recipes for C', which has been used in creating the data set for the present experiment. Data generated from here will also be referred to as 'formula data'. (4) Modified versions of Excel data were also used to explore ideas.

Several tests and methods were used to check for randomness of these data sets. Autocorrelation was done and q-q plots were made to check for closeness to the random theory and to look for outliers. The Difference-Sign test, Turning Point test and the Rank test were also done for each sample. In addition, a chi-squared goodness of fit test and a Kolmogorov goodness of fit test were carried out on each set of numbers.

2. Evaluating randomness

2.1. Graphically evaluating randomness

2.1.1. Q-Q Plots

Cumulative distribution function of Uniform (0, 1) is the identity function. Quantile-quantile plots or q-q plots compare the percentile fractions of each of these respectively. Figures 1 (a), (b), (c), 2 (a), (b), (c), 3 (a), (b), (c) show the close proximity between the ordered simulated data and the expected data. In addition to the q-q plots, the difference between the observed quantile and the expected quantile was graphed to see what kind of a trend line could emerge. The results were interesting as there were slight hints of quadratic trends in all three of the data sets. The random number generator used in Numerical Recipes in C has less emphatic trend lines. Please see Figure 1(c), 2(c) and 3(c). The quadratic trend further supports the bulk in the middle that we see in Figures 1(b), 2(b) and 3(b).

Microsoft Excel: - The q-q plot of the data set created by Microsoft Excel is close but not perfect. Some discrepancies are apparent between the quantiles and the ordered data, as seen in Figures 1 (a), (b), (c).

Figure 1(a)

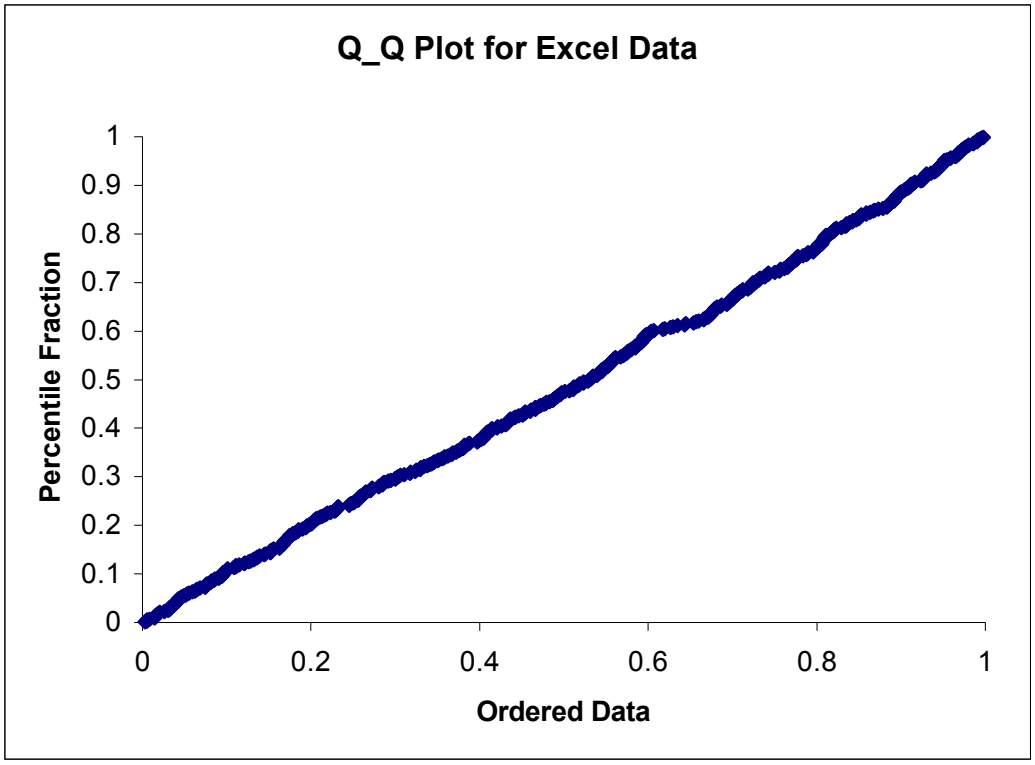


Figure 1(b)

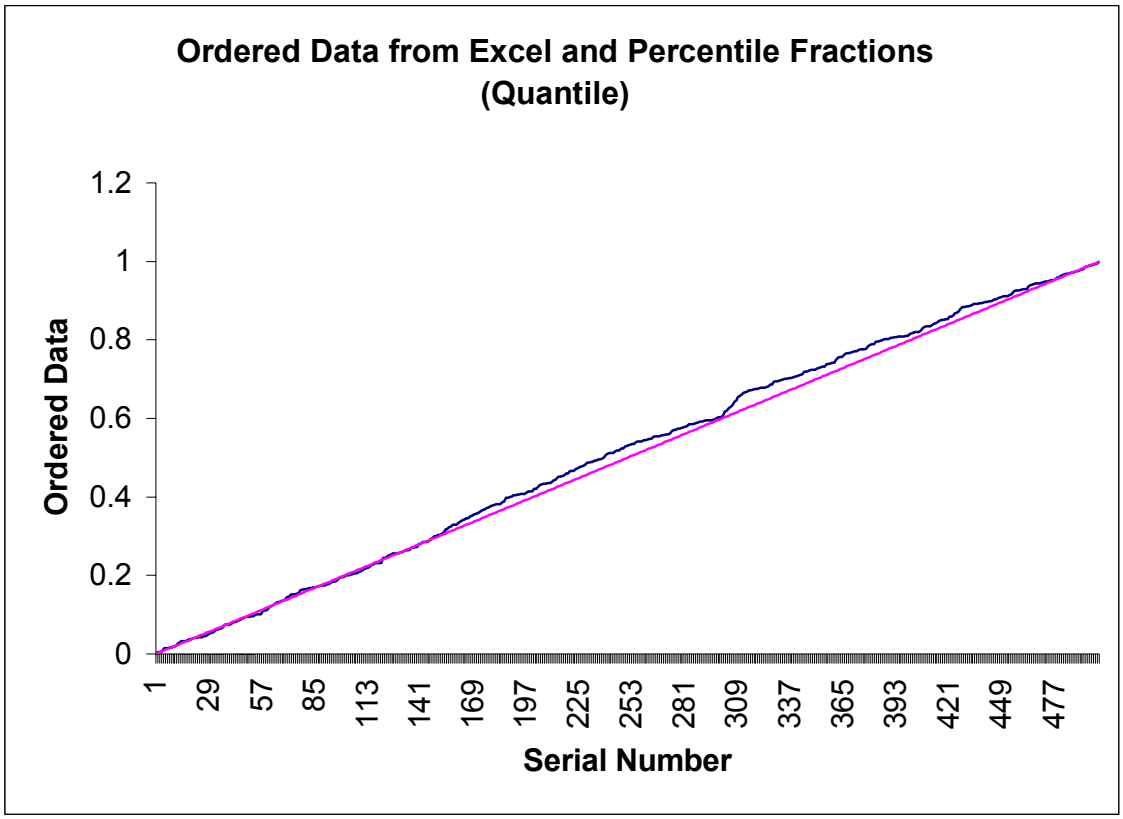
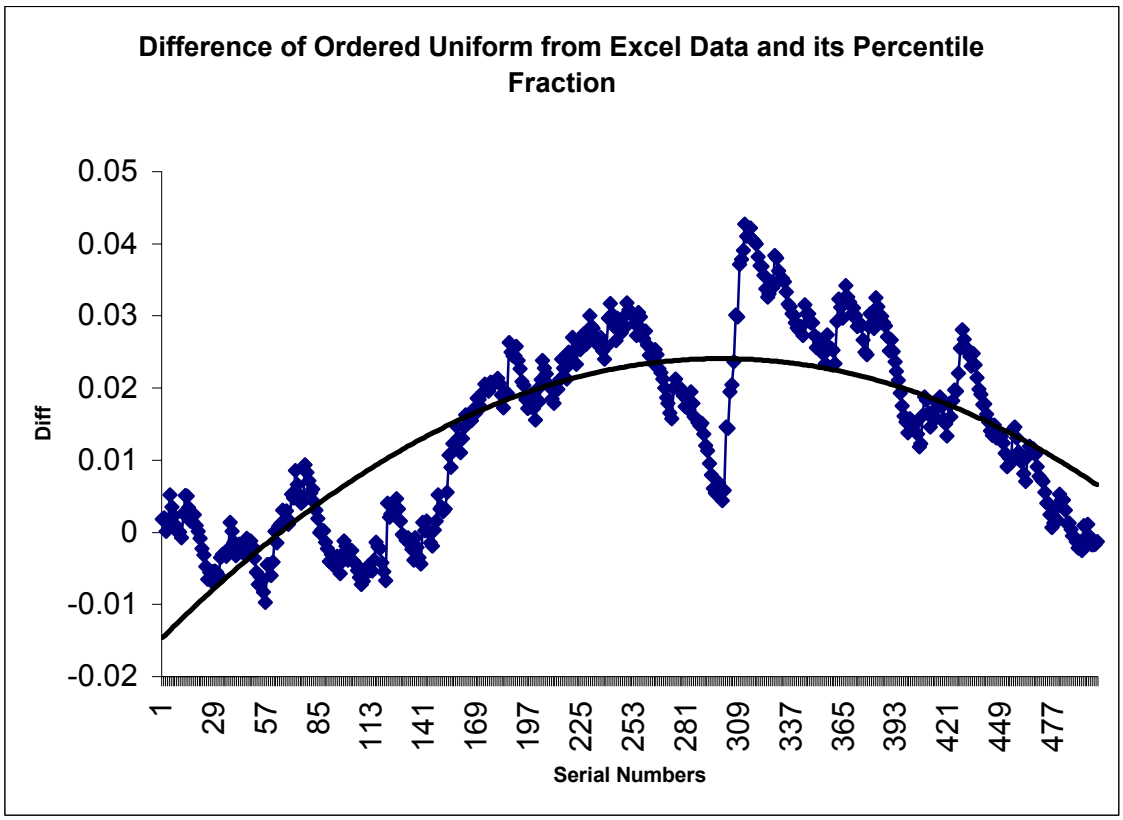

Ordered Data from Excel and Percentile Fractions
(Quantile)
Ordered Data
1.2
1
0.8
0.6
0.4
0.2
0
1
29
57
85
113
141
169
197
225
253
281
309
337
365
393
421
449
477
Serial Number

Figure 1(c)



Numerical recipes for C: -The q-q plot of this data set looks even worse than the one produced by Excel. There is more distance between the ordered data and the percentile fractions in the middle than there was in Excel. Refer to Figures 2 (a), (b), (c).

Figure 2(a)

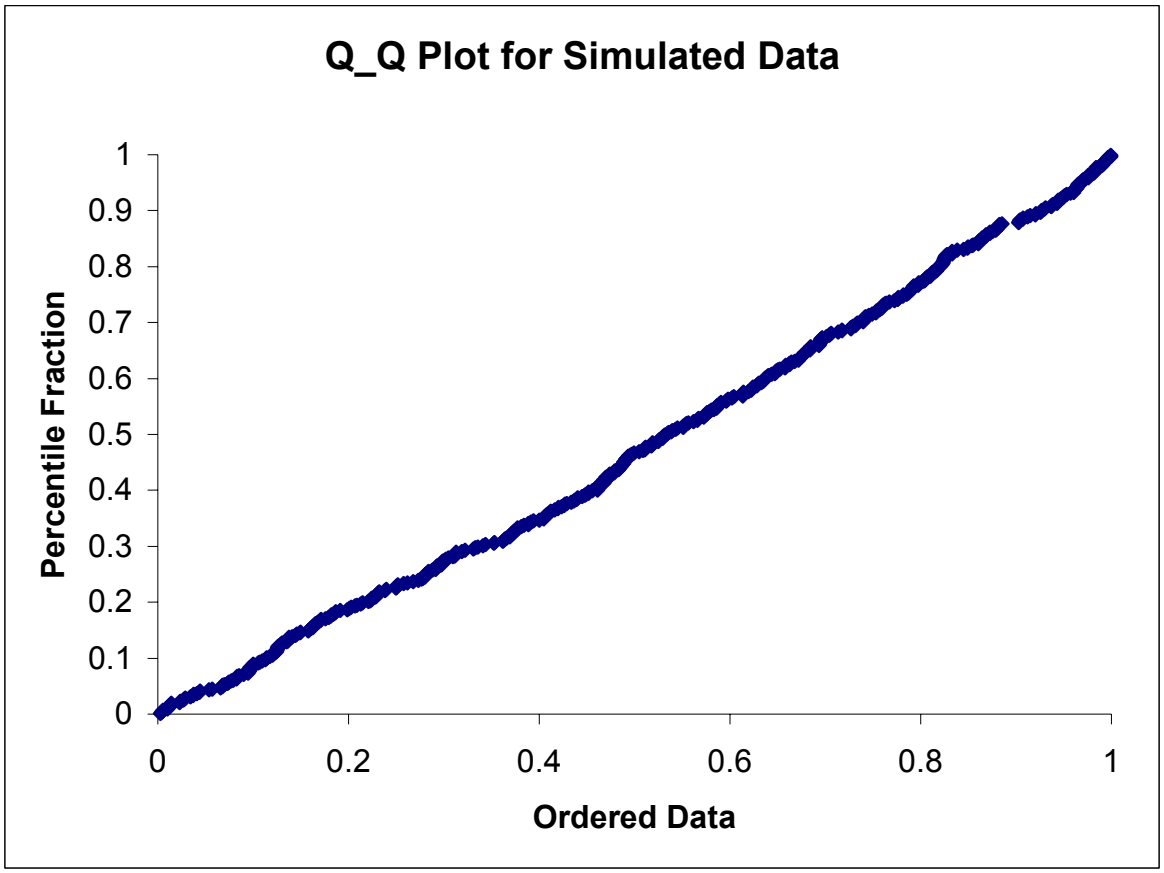

Q_Q Plot for Simulated Data
Percentile Fraction
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Ordered Data

Figure 2(b)

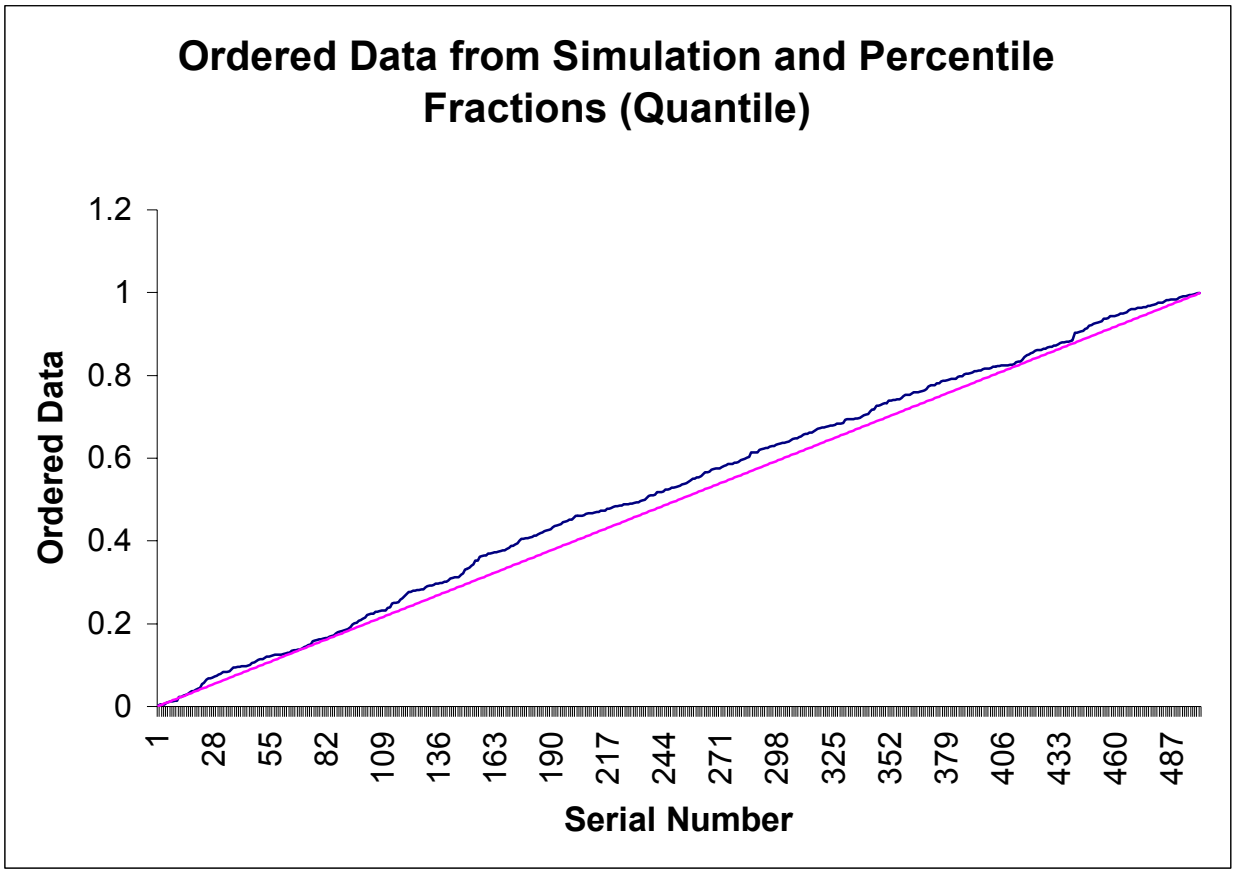
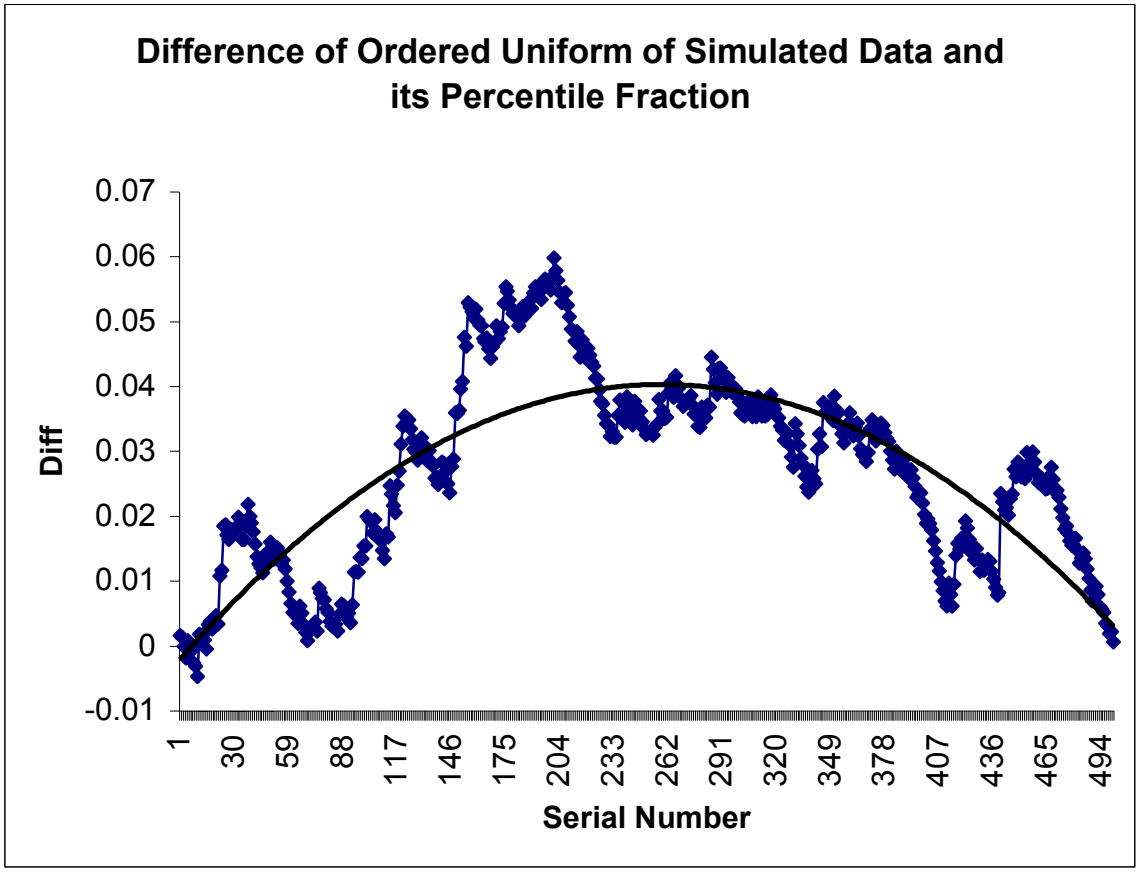

Ordered Data from Simulation and Percentile Fractions (Quantile)
Ordered Data
1.2
1
0.8
0.6
0.4
0.2
0
1
28
55
82
109
136
163
190
217
244
271
298
325
352
379
406
433
460
487
Serial Number

Figure 2(c)


Difference of Ordered Uniform of Simulated Data and its Percentile Fraction
Diff
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
-0.01
1
30
59
88
117
146
175
204
233
262
291
320
349
378
407
436
465
494
Serial Number

C++: - The q-q plot of the C++ data is similar to the other graphs. It too has a tendency to expand in the middle and narrow down near the ends. However, this graph deviates more in the middle between ordered data from C++ and percentile fraction, than those in the other methods [Figure 3 (b), (c)].

Figure 3(a)

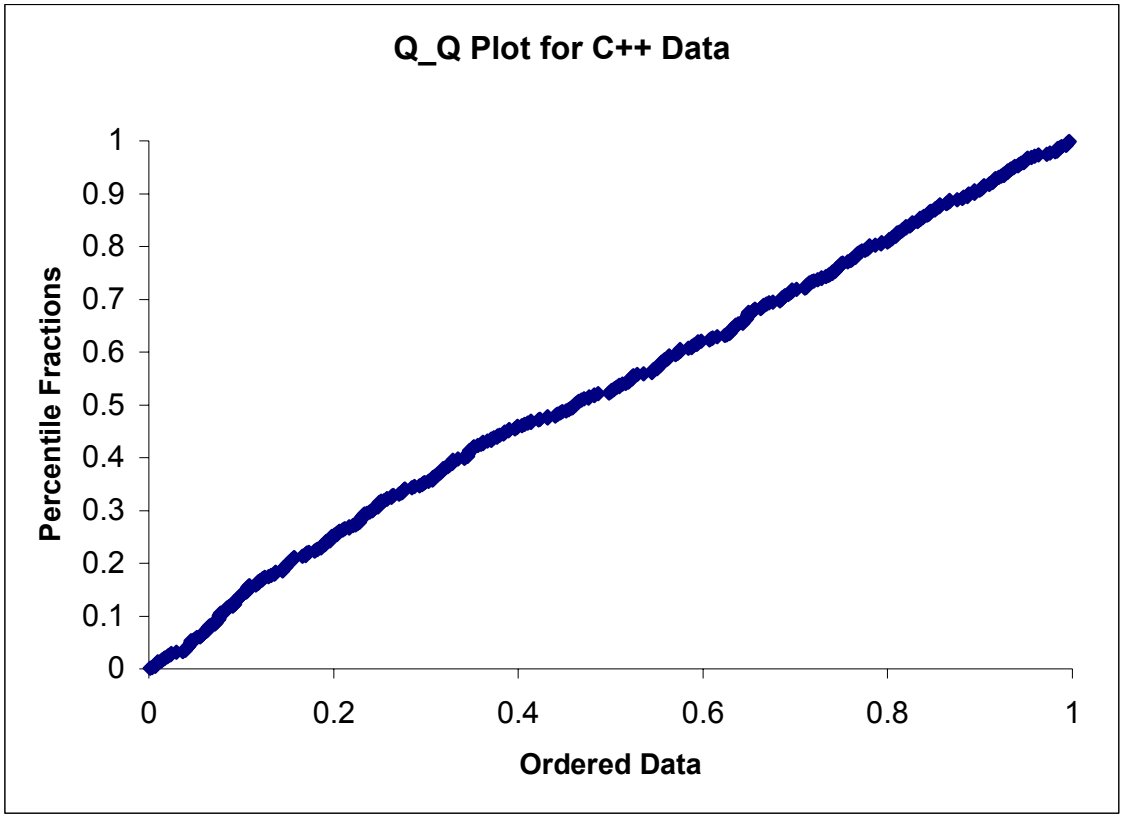


Figure 3(b)

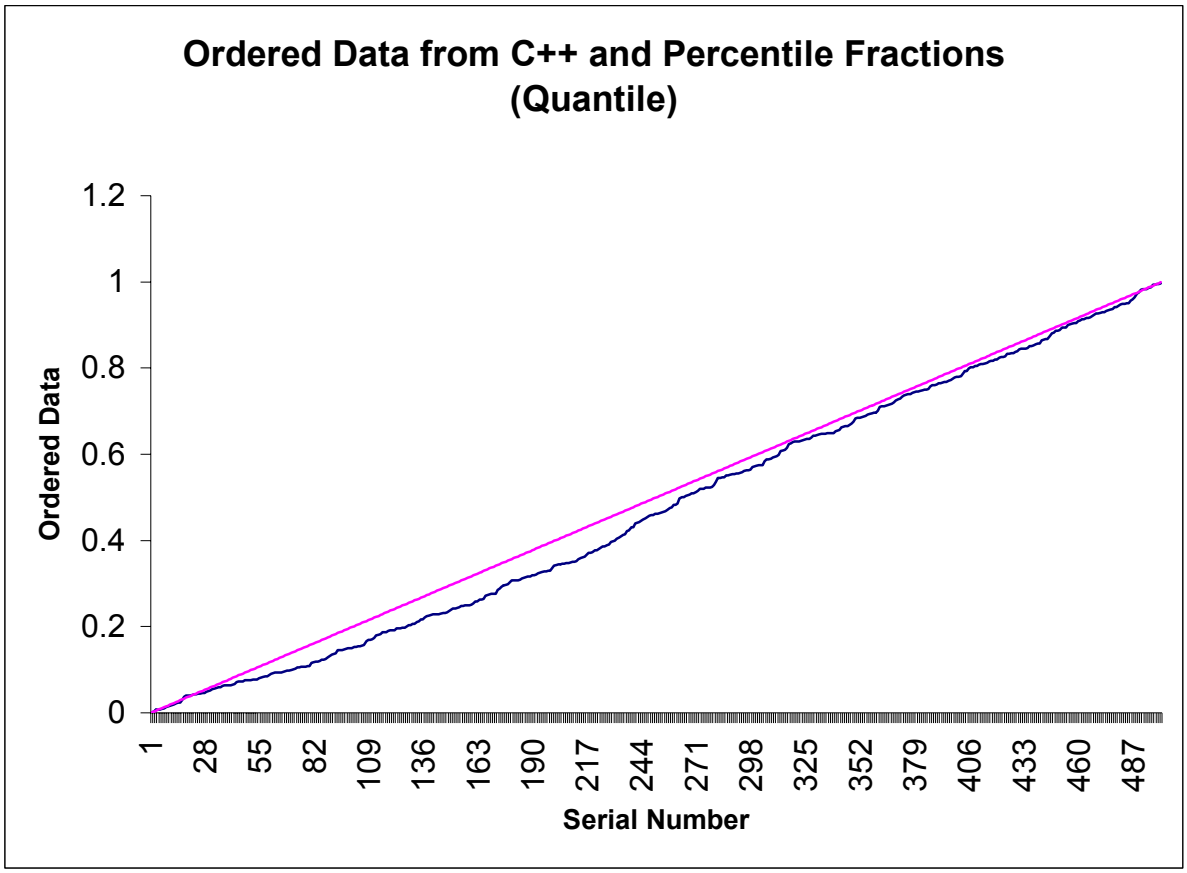
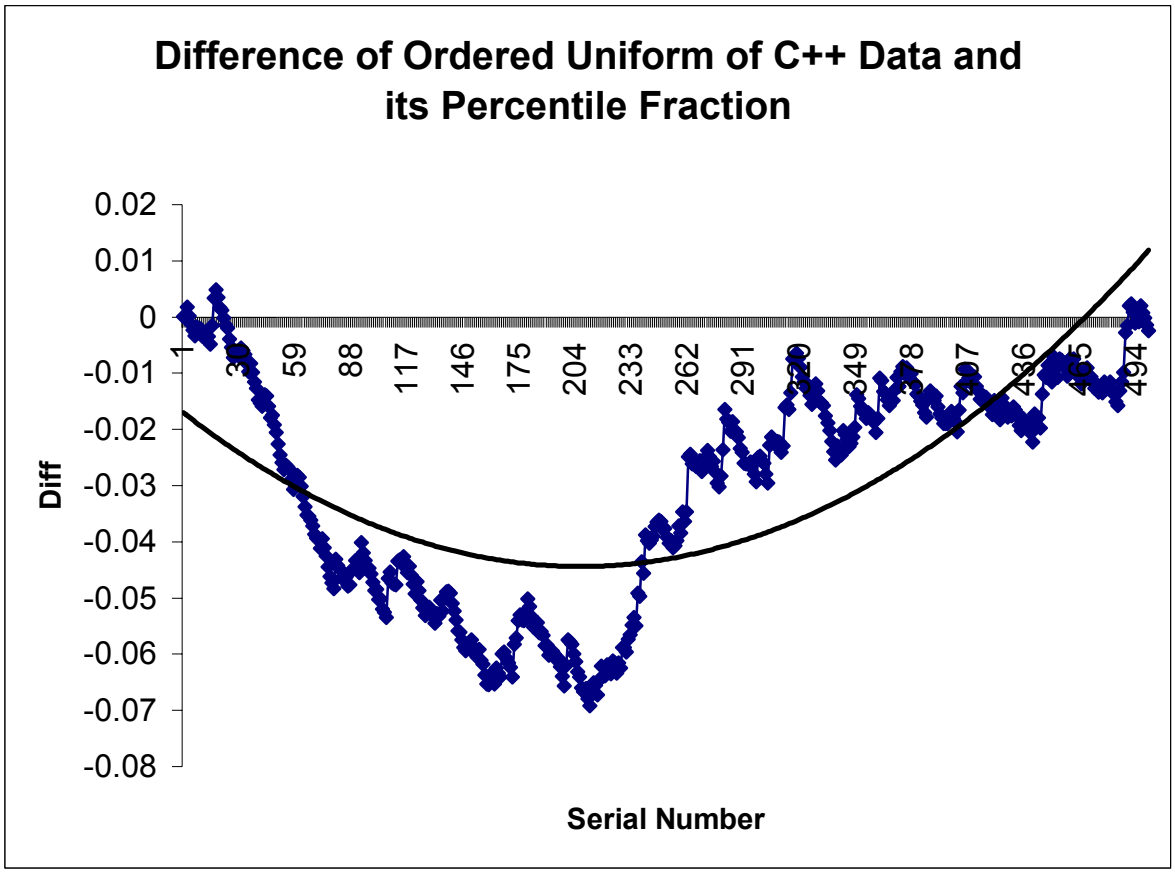

Ordered Data from C++ and Percentile Fractions
(Quantile)
Ordered Data
1.2
1
0.8
0.6
0.4
0.2
0
1
28
55
82
109
136
163
190
217
244
271
298
325
352
379
406
433
460
487
Serial Number

Figure 3(c)


Difference of Ordered Uniform of C++ Data and its Percentile Fraction
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
-0.07
-0.08
Diff
1
30
59
88
117
146
175
204
233
262
291
320
349
378
407
436
465
494
Serial Number

2.2. Tests used in Time Series Analysis to Check Randomness

Using time series analysis the data was tested for more kinds of trends in it. Three tests were performed to this extent. The Turning Point test, the Difference Sign test and the Rank test were used to check the random nature of the data set. However, none of the tests detected significant departures from randomness.

2.2.1. Autocorrelation:

Data sets that appear to be random might actually have some sort of a linear or cyclic trend. To check for this an autocorrelation test was carried out for 40 lags, which is the standard for most time series calculations. Note that $\hat{\rho}(h)$ is distributed as N (0, 1/n). For a 95% confidence, the cut off will be ± $1.96/\sqrt{n}$. You expect 0.05h = 0.05(40) = 2 to not follow these limits. Please see for, e.g., Brockwell and Davis (1996).

Microsoft Excel: - The data set generated through Microsoft Excel was inconsistent on 4 lags out of the 40 lags. Ho is the hypothesis that the data sample has no significantly large serial correlation. The 4th, 14th, 35th and 40th lags were shown to be out of bounds. For a 95% confidence, Ho is rejected as only 2 of the 40 are allowed to be out of bounds. So there is ample proof to suggest that some cyclic or quadratic trend exists in this data.

C++: - The autocorrelation for C++ looks pretty good again in having only 2 lags out of bounds, which is permissible for a 95% confidence limit. The 14th and the 37th lags are the ones, which are showing negative in this test. However, there is not enough proof to show any trend in the data.

Numerical recipes for C: - This generator has been claimed to be better than most random number generators available today. This test infers that the data set created by it is consistent with a 95% confidence interval. Only two lags the 10th and the 36th are outside the allowed bounds. So, the data does not show any trend.

The following three tests were also used to check the randomness of the simulated data. Sections 2.2.2 to Section 2.2.4 describe these tests from Brockwell and Davis (1996) for the sake of completeness of this report.

2.2.2. Turning Point Test:

If $y_1, y_{2, \ldots,} y_n$ is a sequence of observations, there is a turning point at time i, $1 < i < n$ if $y_{i-1} < y_i$ and $y_i > y_{i+1}$ or if $y_{i-1} > y_i$ and $y_i < y_{i+1}$. If T is the number of turning points of an independent data sequence of length n, then, since the probability of a turning point at time i is 2/3; $\mu_t = E(T) = 2(n-2)/3$;

$\sigma_t = (16n - 29)/90$; T ~ N (μ_t , σ_t^2). If $\frac{|T - \mu_t|}{\sigma_t} > \phi_{1-\alpha/2}$ is true, we reject H_o (hypothesis that there is a trend in the data) at α .

2.2.3. Difference –Sign Test:

Count the number, S, values of i for $y_i > y_{i-1}$

$\mu_s = E(S) = \frac{1}{2}(n-1)$; $\sigma_s = \frac{(n+1)}{2}$; S ~ N (μ_s , σ_s^2). If $\frac{|S - \mu_s|}{\sigma_s} > \phi_{1-\alpha/2}$, then the claim that there is trend in the data is rejected.

***2.2.4.* Rank Test:** The rank test specifically detects linear trend in data.
P is the number of pairs (i, j) such that $y_j > y_i$ and j > i, where i =1, 2, 3…. (n-1).
Each event has probability ½; $\mu_p = \frac{1}{4}n(n-1)$; $\sigma_p^2 = \frac{n(n-1)(2n+5)}{72}$; P ~ N (μ_p , $\sigma_p{}^2$).

If $\frac{|P-\mu_p|}{\sigma_p} > \phi_{1-\alpha/2}$ is true, we reject the claim that there is no linear trend in the data.

2.3. Goodness of Fit Tests

***2.3.1.* Kolmogorov-Smirnov Goodness-of-Fit Test**: The Kolmogorov-Smirnov test is used to decide if a sample comes from a population with a specific distribution.
The Kolmogorov-Smirnov test is defined by: H_0: The data follow a specified distribution
Test Statistic: The Kolmogorov-Smirnov test statistic is defined
as $D = \max_{1<i<n} |F(Y_{(i)}) - \frac{i}{N}|$. Where F is the theoretical cumulative distribution of the distribution being tested, which must be continuous and must be fully specified.
The hypothesis regarding the distributional form is rejected if the test statistic, D, is greater than the critical value obtained from a table. There are several variations of these tables in the literature that use somewhat different scaling for the Kolmogorov-Smirnov test statistic and critical regions. These alternative formulations should be equivalent, but it is necessary to ensure that the test statistic is calculated in a way that is consistent with how the critical values were tabulated. The statistical package S-plus was used to get these results.

EXCEL: Ks = 0.0437, p-value = 0.2945
Alternative hypothesis: True c.d.f. is not the uniform distribution with the specified parameters

NUMERICAL RECIPES IN C: Ks = 0.0608, p-value = 0.0496
Alternative hypothesis: True c.d.f. is not the uniform distribution with the specified parameters

C++: Ks = 0.0702, p-value = 0.0145
Alternative hypothesis: True c.d.f. is not the uniform distribution with the specified parameters

The Kolmogorov-Smirnov goodness of fit test rejected all samples except for the Excel data to be random samples from U (0, 1). If the p-value is too small we reject H_o (the data is a random sample). We see for Excel that the p-value is fairly large at 0.2945 and the others are 0.0496 and 0.0145.

***2.3.2.* Chi-Squared Goodness-of-Fit Test:** The chi-square test is used to test if a sample of data came from a population with a specific distribution. The chi-square test is defined for the hypothesis: H_0: The data follow a specified distribution.

Test Statistic: For the chi-square goodness-of-fit computation, the data are divided into *k* bins and the test statistic is defined as

$$\chi^2 = \sum_{i=1}^{k} (O_i - E_i)^2 / E_i,$$

Where O_i is the observed frequency for bin *i* and E_i is the expected frequency for bin *i*. The expected frequency is calculated as the length of the interval for evaluating data from U (0, 1). Therefore, the hypothesis that the data are from a population with the specified distribution is rejected if $\chi^2 > \chi^2_{(\alpha, k-1),}$ where $\chi^2_{(\alpha, k-1)}$ is the chi-square critical point with *k* - 1 degrees of freedom and significance level of α . This was taken from Conover (1998).

The χ^2 test values for the three main methods of random data from uniform (0, 1) are as follows: - EXCEL: 10.48. NUMERICAL RECIPES IN C: 10.28 and C++: 16.16.

The critical value for this test with 10 classes at 5% significance level is 16.92. So this test approves all of the above samples to be random samples from U (0, 1).

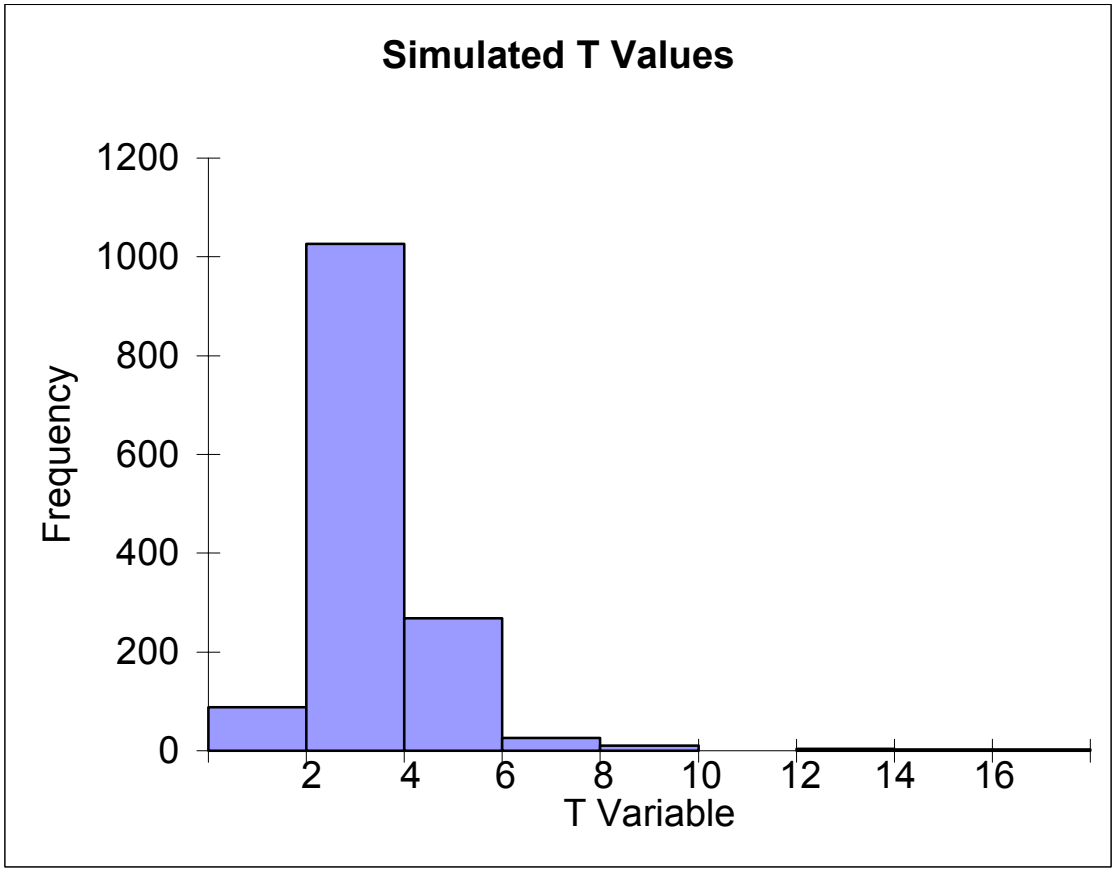
3. Kolmogorov-Smirnov Type Test for Goodness-of-Fit

This test is described by

$$T = \sup_{0 \le x \le 1} \frac{\left| \sqrt{n} \left\{ \dfrac{\sum_{i=1}^{n} (X_i \le x)}{n} - x \right\} \right|}{\sqrt{x(1-x)}}$$

Reject H_o: the data is a random sample from Uniform (0, 1) if T > C. It is theoretically shown that T has a limiting distribution of |Z|, where Z is the standard normal. This means that T should have a half-bell shaped graph as Z has a bell shaped curve symmetric around 0. This fact is being evaluated using simulation. To simulate this situation, a program in C++ was written to create 5000 sets of random numbers. Each set had 500 random numbers in it. The above transformation was performed on each of the data sets and then the data set was sorted out. The largest number of each data set was collected in an array. The graph created (Figure 4) in this case was not very dissimilar to what we expected it to be. The random number generator not being perfect and the sample size of n = 500 not being large enough could be some of the reasons for shift to the right of the histogram curve as compared to what it should be when compared to the theoretical curve. See. For the simulation program please see Section 5.2.

Figure 4



4. Summary:

Compare and contrast through Table 1 shows that none of the random number generators are really perfect. There is some discrepancy in each one of them.

Table 1. *Following is a summary of all the tests performed on all the generators.*

	Auto correlation	Q_Q plot	Chi Squared	Kolmogorov- Smirnov	Difference-Sign	Turning-Point	Rank
C++	Pass	Fail	Pass (16.16)	Fail (p-value = 0.0145)	Pass	Pass	Pass
Numerical Recipes in C	Pass	Fail	Pass (10.28)	Fail (p-value = 0.0496)	Pass	Pass	Pass
Excel	Fail	Fail	Pass (10.48)	Pass (p-value = 0.2945)	Pass	Pass	Pass
Sub Intervals	Fail	Excellent	Excellent (4.48)	Terminated because of poor autocorrelation results			

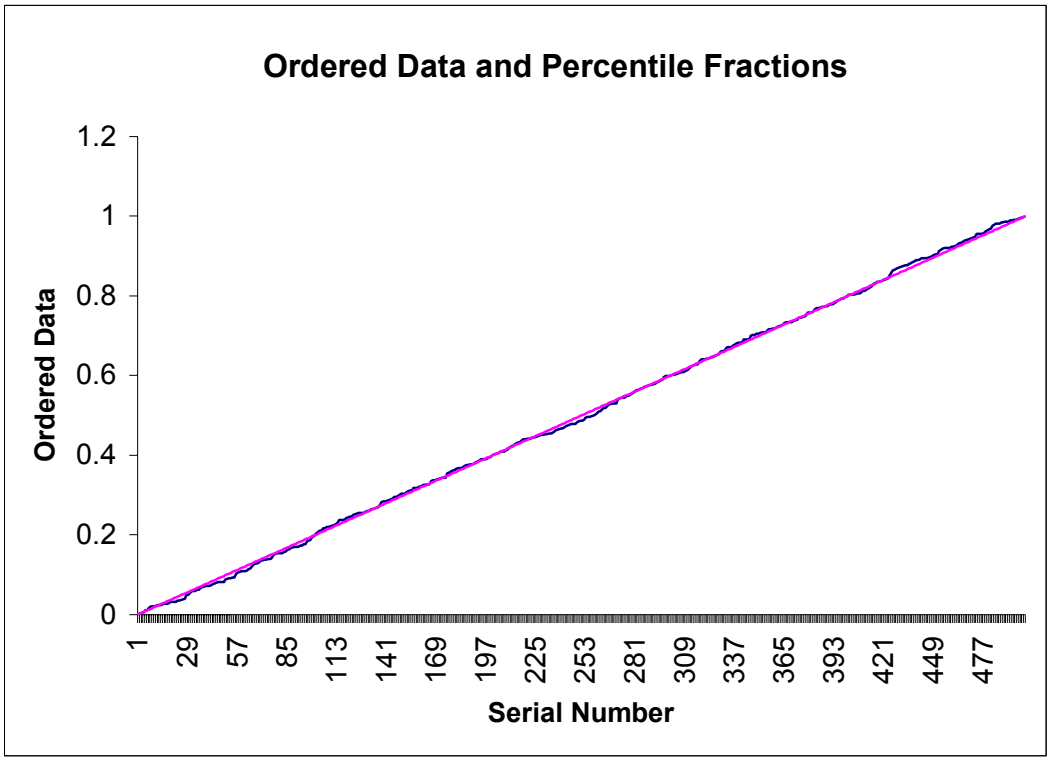
5. Appendix:

5.1. Modified Excel Methods

The methods described here try to improve on the random numbers created by Microsoft Excel.

An analysis of several q-q plots created shows that the actual data line has a tendency to stick to the end points and deviate away from the centre. It is apparent from this that one could rescale the data points to lie between -0.2 to 1.2 and then truncate the selection to points between 0 and 1. However, this did not improve the "randomness" of the data set. Since the q-q plot line tends to stick to the end points and deviate from the centre, we wanted to explore the effect on data set of having more end points. Random numbers were therefore created in sub intervals. Instead of generating 500 numbers between 0 and 1, we created 100 between 0 and 0.2, 100 between 0.2 and 0.4, and so on, and also choose these five sub-intervals randomly. We observed that smaller interval widths improved the random generation in terms of the Q-Q plot as is seen from Figures 4 and 1(b). However, the autocorrelation results are worse for all the random numbers created by this method, than any other method.

Figure 5



5.2. Simulation Program

```
#include <cstdlib>
#include <ctime>
#include <fstream>
#include<time.h>

using namespace std;

double g(double v)
{
        return v/500;
}

void delay(int duration)
{
  clock_t target;
  int rep;

  for (rep=0; rep < duration; rep++)
  {
   target=clock()+1;
   while ( clock() < target );
  }
}

void main()
{
        double a[500],h[500],temp,result[5000];
        int i,j,k,out;
        char dummy;

for(out=0;out<5000;out++)
{

srand((unsigned)time(NULL));


//ofstream fout("data.dat");


for(i=0; i < 500; i++)
{
// Generate a random number
// Mod it by 9999 - This gives a number between 0-9998 inclusive
```

```
// Add 1 to it to make the range between 1-9999 inclusive
// Divide by 10000 to shift the decimal place 4 to the left
// (make it from 0.0001 - 0.9999 inclusive)
// Casting the entire equation back to a double at the same time

double Num = static_cast<double> ((rand() % 9999) + 1) / 10000;

a[i]=Num;
}

        for(i=0;i<500;i++)
                for(j=i;j>0;j--)
                        if(a[j+1]<a[j])
                        {
                                temp=a[j+1];
                                a[j+1]=a[j];
                                a[j]=temp;
                        }
        for(k=0;k<500;k++)
                h[k]=(g(k+1)-a[k])*sqrt(500/a[k]*(1-a[k]));

        for(i=0;i<500;i++)
                for(j=i;j>0;j--)
                        if(h[j+1]<h[j])
                        {
                                temp=h[j+1];
                                h[j+1]=h[j];
                                h[j]=temp;
                        }

result[out]=h[499];
cout<<result[out]<<endl;
delay(25);
}
ofstream outf("result.txt");
for(i=0;i<5000;i++)

        outf<<result[i]<<endl;
outf.close();
}
```

REFERENCES

1. Kronmal, R. Evaluation of a Pseudorandom Normal Number Generator. J. ACM 11, 3 (1964), 357-363.
2. Rotenburg, A. A new pseudorandom number generator. J. ACM 7, 1 (1960), 75-77.
3. Goldreich, O., Goldwasser, S., Micali, S. How to Construct Random Functions. J. ACM 33, 4 (1986), 792-907.
4. Boyar, J. Inferring sequences produced by Pseudorandom Number Generators. J. ACM 36, 1 (1960), 129-141.
5. Practical Nonparametric Statistics by W. J. Conover, 3rd Edition, December 14, 1998, ISBN: 0471160687
6. Introduction to Time Series and Forecasting by Peter Brockwell and Richard Davis, 2nd Edition, 1996, ISBN: 0387947191.
7. Probability and Statistics for Engineering and the Sciences by Jay L. Devore, 5th Edition, December 1999, ISBN: 0534372813.
8. Object-Oriented Programming in Microsoft C++ by Robert Lafore, 4th Edition, 1992, ISBN: 1878739085.
9. C++, How to Program by Deitel & Deitel, 4th Edition, 2002, ISBN: 0130384747.
10. http://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm.