Using π digits to Generate Random Numbers: A Visual and Statistical Analysis

Ilya Rogers, Greg Harrell, and Jin Wang
Department of Mathematics and Computer Science
Valdosta State University, Valdosta GA 3198, USA

Abstract - *Monte Carlo simulation is an important method with widely applications in real-world problem modeling, solving, and analysis. Random numbers are key part of this method. A good random number generator should have the following qualities: randomness, speed, simplicity, and large period. In this research, we study using pi database to generate random numbers. Our study shows that this method is efficient and simple with large period. The pi database is a free resource on the internet with 12.1 trillion digits. The special structure of the pi random number generator made it simple and fast with almost no cost. Is pi a good random number generator? The most important thing is the randomness. Based on our experiment outputs, the 2D and 3D plots indicate that the randomness of pi is pretty good comparing with the existing popular LCG pseudo random number generates in computer simulation community. Finally we use this pi random number generator to simulate the true pi value. Our result shows that the pi approximation is very accurate.*

Keywords: Monte Carlo Simulation; Random Number Generator.

1 Introduction

1.1 History of π:

π is an irrational number that is extremely helpful in calculating area of a circle. With the ability of calculating area of a circle gives us unparalleled ability to apply the idea in numerous applications. Such applications include: engineering, measuring sound waves, simulation, GPS, and pretty much anything that has a "curved" surface. π, more specifically, is a ratio of circles' circumference and diameter

$$\pi = C/d$$

which allows us to closely estimate circumference and area of a given circle with radius r:

$$C = 2 * \pi * r$$
$$A = \pi r^2$$

History of π is extremely rich and diverse and yet still π hold many mysteries that have not yet have been discovered. It is not known who has originally come up with concept of π but the earliest record of a civilization trying to find the ratio is about 4000 years old belonging to Babylonians and Egyptians. It is speculated that a rope was being used to measure the circumference and the diameter after which they have estimated that π is slightly larger than 3, more specifically approximately 3.125. [1]. Next appearance of π is in a Egyptian Papyrus dated back 1650BCE. The papyrus outlines a list of problems for students to solve one of which required a student to figure out an area of a circle inside of a square [2]. This problem calculated π to be about 3.1605 or 3 and 13/81. The approximate value of π as we know it today was calculated by Archimedes by taking 2 hexagons and doubling the sides 16 times. The final result came to about π = 3.1415926535[1]. Fast forwarding to more recent events, the creation of computers and the ability to calculate more decimal digits of π the current record holder as of December 2013 is 12 trillion digits held by Alexander J. Yee & Shigeru Kondo[3].

1.2 How to calculate π?

π, being complex number it is, can be fairly easy but costly to calculate. The problem lies in how precise of decimal places you want it to be. There are multiple ways of calculating π. It is possible to compute π using Numerical methods such as 22/7 or drawing hexagons and multiplying their sides; more sides equal more precise value of π. Another way to compute π is to use computers and algorithms to automate the process. Last but not least option is by using a random number generator to simulate π. Geometrical way of calculating π is by inscribing and circumscribing n number of polygons and the calculating their perimeter and areas. Archimedes used this technique to estimate π being roughly 3.1416. More modern way of calculating π is using Gregory's formula:

$$\int_0^x \frac{dt}{1+t^2} = \arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

Evaluating for x = 1 we get:

$$\frac{\pi}{4} = 1 + \frac{1}{3} - \frac{1}{5} + \frac{1}{7} - \cdots$$

This method was used by Abraham Sharp to calculate π to 72nd decimal place [2]. With computer age the possibilities of computing decimal places for π has significantly increased. Instead of spending years of computing π to several thousandth place early computer could do it in matter of hours. One of the computer algorithms using ENIAC in 1950 to calculate π to the 2037 digits used the following algorithm [5]:

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} (-1)^n * \left[\frac{100(0.2)^{2n+3} - \left(\frac{1}{239}\right)^{2n+1}}{2n+1} \right]$$

It has taken the machine 70 hours to finish the computation. That record was beaten in 1955 by using the same formula but with a better machine. As the computers evolved at exponential rate (Moore's Law) the possibility of calculating π to higher number of decimal places has grown along with it. Lastly, it is possible to calculate π using simulation; the method is called "Monte Carlo π". The Monte Carlo method calculates π/4. We begin by drawing a 1 by 1 square on a coordinate plane, and then we inscribe a circle inside of the square. Next, use LCG (Linear Congruential Generator) RNG to randomly generate X and Y value plotting them in the first quadrant. Using a computer algorithm we check if $X^2 + Y^2 \leq 1$ meaning that the point is inside/on the line the quarter-circle and we increment our "success" or k counter which is represented by red points in picture above. After the simulation, depending on number of samples we calculate our p̂ where n is number of trials and k is number of successes:

$$\hat{p} = k/n$$

With enough rounds we will start to see that p̂ will begin to look like true π value decimal place by decimal place. Due to the nature of simulation and equation of standard error:

$$ste = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

We would have to run the simulation 100 times more in order to gain one decimal place accuracy every time. After a while it is obvious that calculating decimal places of π using simulation can get extremely costly in terms of time and resources.

1.3 Trillion digits π value

The record as of December 2013 in calculating decimal points of π is 12.1 trillion digits achieved by Alexander J. Yee & Shigeru Kondo. Yee and Kondo have built a computer [4]. Note the amount of RAM memory and HDD space. Calculating π to approx. 12 trillionth digit is no easy task and requires tremendous resources. The resources required go up as the number of digits increases, especially HDD space requirements. Yee and Kondo used Chudnovsky algorithm displayed below:

$$\frac{1}{\pi} = 12 \sum_{k=0}^{\infty} \frac{(-1)^k (6k)! \, (54514013k + 13591409)}{(3k)! \, (k!)^3 * 640320^{\frac{3k+3}{2}}}$$

After the algorithm completes one simply takes the inverse of the result giving them the value of π to the n number of decimal places. Implementing this algorithm in a computer along with some I/O operations to write data it has taken Yee and Kondo 94 days to calculate 12.1 trillion digits of π… then they ran out of HDD space. It is clear that any computer can compute π to extremely high number of decimal places, however, hardware plays major role in terms of time and storage.

2 π RNG Testing and Analysis

2.1 Generating Random number using π values

Talking about generating π to an astounding number of decimal places is great, however, to keep on the to πc we must shift our attention to actual random number generators (RNG). There are numerous random number generators on the market today. Some are quite good (LCG) and some are notoriously bad UNIVAC which as only 5 numbers in the cycle. The optimal RNG produces truly random number and does not have a cycle. Due to the realities of the real world and limitations of computer hardware producing truly random numbers is extremely difficult. Instead algorithm based RNGs were developed. The problem with algorithm RNG is that we can predict next random number if we have the seed and the iteration number, and that those usually have a cycle. The larger the cycle the better RNG is considered due to the fact that there are more numbers to pick from. Speaking in terms of π, there is no said cycle proven to date in π. Theoretically we can calculate π infinitely but due to hardware limitations we only have 12.1 trillion digits. Even still no strong patterns were found in that impressive number. If we continue calculating π past the 12 trillion it is going to be nearly impossible to predict which values will come next. A good RNG is measured on following criteria:

- Uniform distribution
- Memory requirement
- Speed
- Reconfigurable
- Portable
- And implementation easiness

I am going to run some tests and grade the π generator on above mentioned criteria along with some other things. The objective of this paper is to determine whether π decimal digits can be used as random numbers. To achieve my objective I am going to compare my π RNG against a Linear Congruential Generator (LCG) which uses a seed and an algorithm to generate a random number. My hypothesis is that π can be used as a cycle free RNG with similar success as the LCG.

2.1.1 Test 1: 3D uniform distribution of π vs. LCG RNG visual comparison

All of the Java code to create graph plots can be found on my GitHub repository. [8]. The Test method for π RNG is as follows:

- Using y-cruncher ver. 0.5.5. [3] I am going to generate 1 billion decimal places of π and save them into a text document. Y-cruncher saves database file as text file by default.

- Use code to read in the π database file
- Initialize beginning pointer at the beginning of the π value (Number 3)
- Use slice size of 5 digits and n size of 5000, 10^4, and 10^5 resetting RNG back to the beginning of π (Init. pointer = 0) beginning every new number of n.
- Output generated RNs into another output file.
- Using data in the output file I am going to plot the resulting number of samples in 3D scatterplot to try to identify patterns
- As I am generating the random number data for a specific slice I am keeping track of the pointer resetting it only when I am generating numbers for a different slice size.
- Plot the x, y, and z values using Java code and displaying them in a 3D graphs.

Test method for LCG RNG

- Perform same exact procedures as for π RNG.

The algorithm for π is outlined below:

```
• Initialize all input and output streams and any relevant variables.
• Initialize number array to set number and string array to slice number
• Skip to the initialization decimal place of $\pi$ database file.
• Loop for number of sets generating first row of random numbers and storing them in a number array of size set
    o Loop for slice number
        ▪ Read 1 Byte of the file converting it from ASCII to a readable character
        ▪ If character = '.'
            • Throw away the character and read next Byte
        ▪ Add the character to string array building a number/character string
    o Convert each containing string to a number then divide by $10^{slice}$
    o Place resulting number in number array[i]
    o Write the number into the file on the same row
• Skip down to next row of the output file
• Begin main while loop running until number of trials-1 or end of input file
    o Loop for number of sets-1 times
        ▪ Copy contents of number array[i+1] to number array[i]. That leaves last spot open for newly generated RN
    o Loop for slice number
        ▪ Read 1 Byte of the file converting it from ASCII to a readable character
        ▪ If character = '.'
            • Throw away the character and read next Byte
        ▪ Add the character to string array building a number/character string
    o Add the resulted string to the last spot of the number array
    o Loop for the number array length
        ▪ Convert each containing string to a number then divide by $10^{slice}$
        ▪ Write resulting number to output file
    o Skip down to next row of the output file
    o Decrement/increment while control variable
• Flush and close all output/input streams
```

The algorithm for LCG U16807 RNG:

```
• Initialize all input and output streams and all relevant variables
• Initialize m, c, and a to $2^{31} - 1$, 0, and 16807 respectively; w0 is seed variable for the RNG.
• Initialize number array to set size
• Loop for number of sets generating first row of random numbers (Explained in while loop) and storing them in a number array of size set
    o Calculate a temp variable using equation: $\text{temp} = (a * w0 + c) \bmod m$
    o Copy temp variable to w0 variable
    o Write the number into the file on the same row
• Skip down to next row of the output file
• Begin main while loop running until number of trials-1
    o Loop for number of sets-1 times
        ▪ Copy contents of number array[i+1] to number array[i]. That leaves last spot open for newly generated RN
    o Calculate a temp variable using equation: $\text{temp} = (a * w0 + c) \bmod m$
    o Copy temp variable to w0 variable
    o Fill in last spot of the number array with $w0/m$ value
    o Loop for the number array length
        ▪ Write contents of the number array to file as 1 row
    o Skip down to next row of the output file
    o decrement/increment while control variable
• Flush and close all output/input streams
```

We generate 3D graphs where $U_i = x$, $U_{i+1} = y$, and $U_{i+2} = z$. First I am going to display the distribution of slice =

5 and sample size = 5000, 10^4, and 10^5 for π RNG in figures 3-5 respectively. Graphs generated by the U16870 Generator are displayed in figures 6-8. Do note each set of graphs are displayed from different angles of view.

Figure 1. U(0,1) Slice = 5 and Sample size = 5000

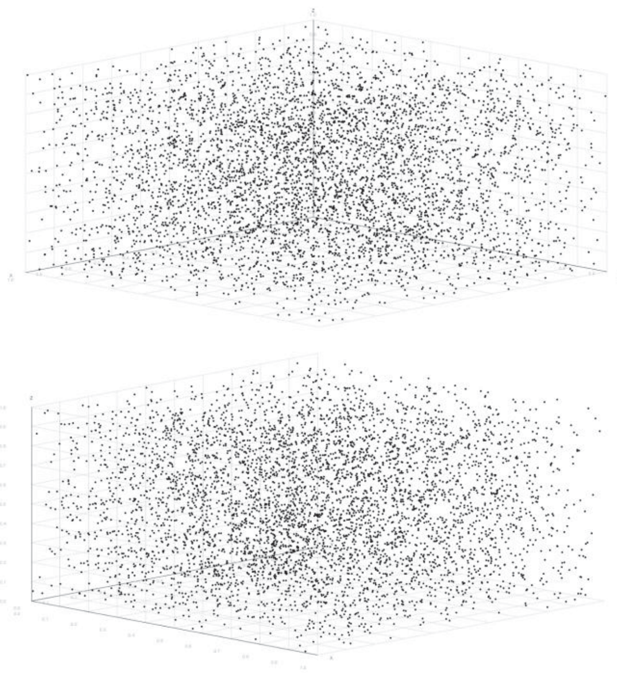


Figure 2. U(0,1) Slice = 5, Sample size = 10,000

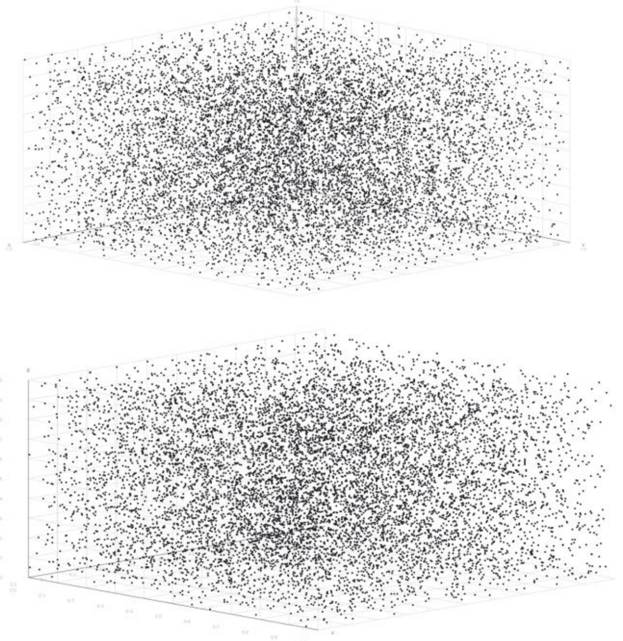


Figure 3. U(0,1) Slice = 5, Sample Size = 100,000

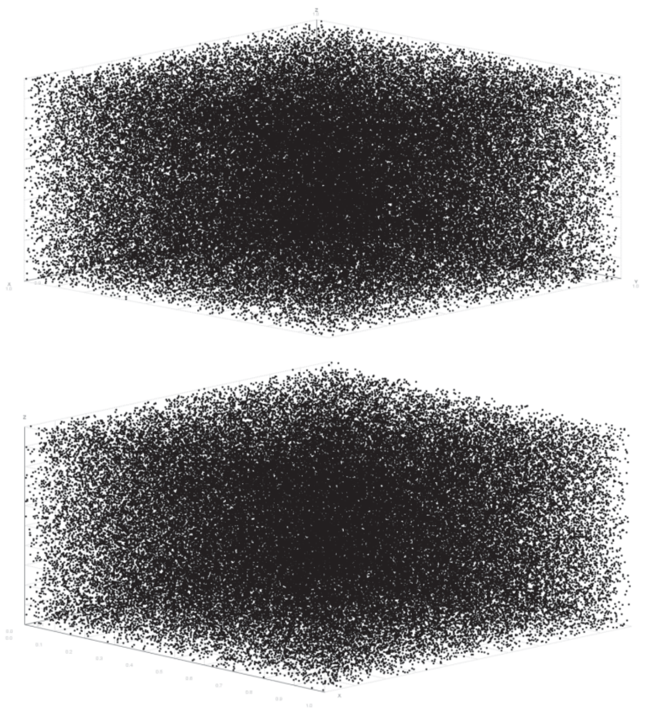


Figure 4. U(0,1) w0 = 1, Sample size = 5000

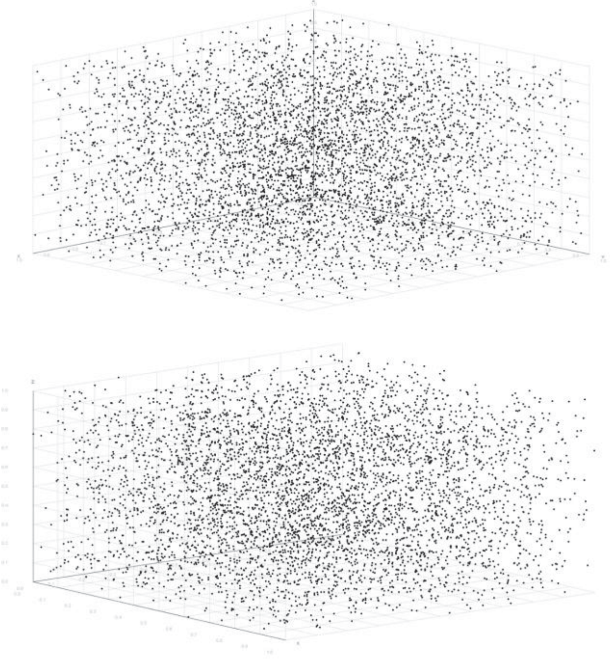
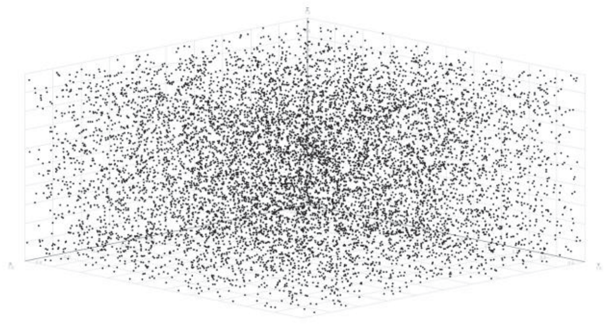


Figure 5. U(0,1) w0 = 1, size = 10,000



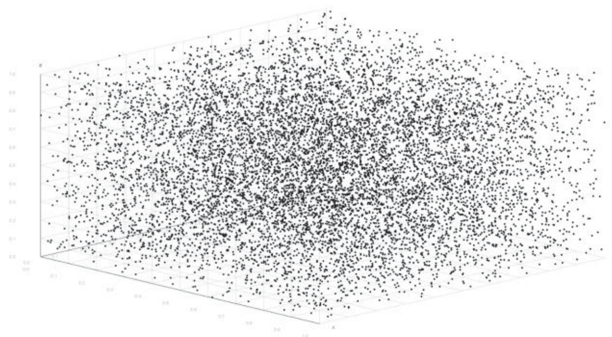
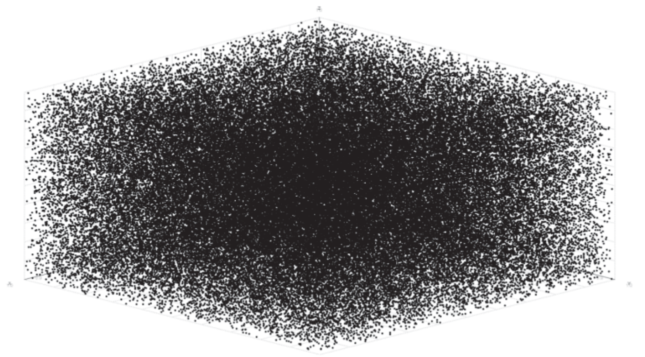
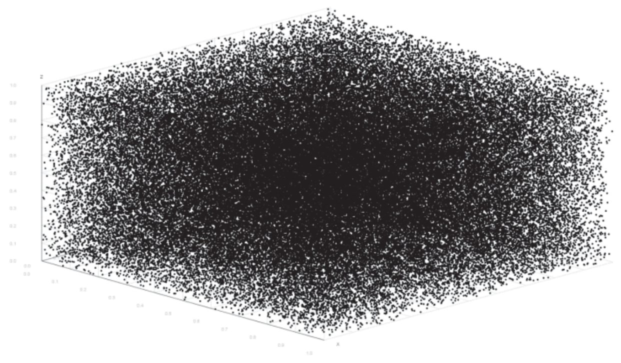


Figure 6. U(0,1) w0 = 1, Sample size = 100,000





Test 1 Summary:

Comparing the uniform distribution graphs of the π and U16807 generators there is a minimal difference. The distribution is uniform across all of the tests. If done for higher number of iterations a solid rectangle would appear indicating that the distribution is all the way across U(0,1). There are no observable patterns indicating any cycles or "preferred" numbers. Final assessment is that π RNG is identical to U16807 in terms of uniform distribution and does generate good random numbers.

2.1.2 Generating π Monte Carlo method using U16807 and π RNG and statistical analysis

Second test I am going to perform is Monte Carlo π. I am going to use both RNGs to emulate a real world problem. I am going to use π and LCG RNGs to generate π. I am going to use a circle inscribed inside 1 by 1 square method to approximate π. Each RNG will run for $10^4, 10^6$, and 10^8 iterations then will be compared in terms of p̂ vs. true π value using swing digit method to approximate cut-off decimal place. Swing digit method works by removing unnecessary decimal places for generated p̂ value. Suppose:

$$\text{Generated } \hat{p} = 3.112343452$$
$$Ste = 0.02346434$$

Look at ste value from left to right and count all the zeros without break. If a non-zero digit is encountered stop and that is how many fist digits of p̂ we will keep. Idea is that the first non-zero digit is where the actual uncertainty error is so the p̂ will fluctuate that that decimal place which is not what we want.

$$\text{After swing digit: } \hat{p} = 3.11$$

Method (Same for both generators):

- Each generator will be run for 10^4, 10^6, and 10^8 iterations.
    - Due to the fact that I have to run 100 times more got get extra decimal point
- LCG will start at $w0 = 1$ and π RNG will start at pointer location 0 for each iteration test.
- π RNG will have a slice of 5 for each RN
- All data will be imported from corresponding RNG output files
    - Must generate 2 times number of samples due to reading X and Y value per iteration.
        - 1bn π number is just enough to do large test
    - Single set
- Each RN sample set will be run through π simulator
- Success counter will be incremented only if condition is $x^2 + y^2 \leq 1$
    - $A = \pi * r^2$
- $\hat{p}$ standard error will be calculated at the end of the run
    - $\hat{p} = k/n$
    - $ste = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$

- Results will be recorded for comparison where $\hat{p}$ will be cut off using swing digit method.

Algorithm:

The algorithm is largely the same for both π and U16807. Due to both programs grabbing values from pre-generated list of random numbers

- Pass in the file containing a list of pre-generated random numbers
    - For π it's the list generated with π RNG and with U16807 is the list generated with U16807 RNG
- Initialize file reader
- Take input on how many iterations to perform
- Initialize success variables
- Start while loop running for entered number of iterations
    - X = first random number read from the list
    - Y = second random number read from the list
    - If $X^2 + Y^2 \leq 1$
        - Increment success variable
    - Increment/decrement loop control variable
- Calculate and store p̂
    - Success/iterations
- Calculate and store true p
    - Use system provided variable for π. Java = Math.PI Due to the fact that we are only generating to 2-4 decimal places there is no need to have a large decimal point of π.
- Calculate and store standard $\hat{p}$
    - $\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$
- Return a string/print all above mentioned variables.

Results:

Table 1. Results for sample size 10^8, π slice = 5, and $w0$ = 1.

RNG	Raw $\hat{p}$	True p	ste	Swing digit $\hat{p}$	Time (ms)
π	3.14151212	3.141592653589793	1.6422393491489316E-4	3.1415	117152.0
U16807	3.14143824	3.141592653589793	1.642290700292035E-4	3.1414	205539.0

Table 2. Results for sample size 10^6, π slice = 5, and $w0$ = 1.

RNG	Raw $\hat{p}$	True p	ste	Swing digit $\hat{p}$	Time (ms)
π	3.142004	3.141592653589793	0.0016418973366151735	3.142	3846.0
U16807	3.142096	3.141592653589793	0.0016418333431819441	3.142	11836.0

Table 3. Results for sample size 10^4, π slice = 5, and $w0$ = 1.

RNG	Raw $\hat{p}$	True p	ste	Swing digit $\hat{p}$	Time (ms)
π	3.1528	3.141592653589793	0.01634335387856483	3.15	1787.0
U16807	3.1496	3.141592653589793	0.016365878650411655	3.14	1622.0

Test 2 Summary:

For all tests U16807 and π RNG performed calculations in the similar manner have produced rather close results. Overall, both RNGs calculated π to the same STE thus calculating same "accurate". Do take note that the smaller the sample size that I used the less accurate swing digit p̂ became indicating that in order for us to get 1 extra decimal place accuracy we have to run the simulation 100 times more than previous trial. Both generators have performed at the same success rate and efficiency. One thing to mention is time. Due to U16807 producing larger decimal place numbers it has taken slightly longer to process opposed to π RNG where it was calculated to 5 decimal places. Overall result is that π and U16807 RNG performed the same.

2.2 Technical Issues

π generator was a unique generator to implement. It has required me to extend my Java knowledge to new levels. First major issue was generating the actual π number. Due to calculation intensity it has taken me substantial amount of resources to calculate π to 1 billion decimal places. Not to say I have a bad computer but it was extremely surprising to see that that calculation has taken up almost all of my RAM memory 6/8GB which resulted in my computer nearly halting for the duration of the calculation. After the calculation finished in roughly 5 min I was surprised to find a file size of 1GB+ in my directory. For commercial implementation π will have to be calculated to much greater decimal places in comparison resulting in numerous terabytes or even petabytes of space being taken. Transferring 1 GB file between directories on 7200RPM HDD was tedious in terms that it would take the computer some minutes to copy the file somewhere else. Next issue I have encountered was actually reading such big file. Instead of Java Scanner class I was forced to use file input streams due to Java running out of heap memory. Using input streams has its advantages however, now my code can read files of theoretically unlimited size. Last major issue was again, the file size except in this scenario it was my output files. I, again, had to use buffered output streams to properly write output files. In some instances generating my RN sets from 1GB file yielded 2-4GB files which could pose much greater issue in commercial sense. Writing to those file have also given me some issues specifically by buffered output streams. For buffered output or input one must flush the stream before exiting the stream, otherwise, you will end up with incomplete set of RNs in your output file. Overall computer hardware plays an immense role in success of π RNG. The relation of computer hardware, specifically RAM, CPU, and HDD to π RNG is that the better the hardware the better π RNG you will have.

2.3 Is π a good random number generator?

In summary, I have performed three tests, each has put π RNG against one of the more popular U16807 RNG. π RNG has proven to be competitive in visual test, iteration (π value calculation) test, and probability calculation test. In terms of uniform distribution both generators perform the same. Memory requirement U16807 has the advantage due to when we generate RNs using Chudnovsky formula we use quite a bit of memory and other resources. To generate 1 billion digits of π it has taken my computer over 8GB of memory; the higher value of decimal numbers I wanted that memory requirement gone up. In terms of speed π generator loses to U16807 in the same manner as mentioned in memory requirement. The more RNs I want the heavier calculations have become. π wins the reconfiguration criteria over U16807. π I can specify start point and slice size giving me different random numbers each time where U 16807 I can only specify w0 which will only place me as some point of the cycle giving me the same RNs if I run it long enough. U16807 is more portable than π generator. π generator requires huge database size, 12.1 trillion ≥ 20TB, in order to have a decent pick of random number whereas U16807 is limited by computers word size and is not backed by database. π generator wins in ease of implementation. If I have a large database all I need to do is read it where in U16807 I have to implement a function for it run properly.

3 Summary

π RNG is an unconventional random number generator however it offers unprecedented speed and accuracy of commercially created RNGs; assuming database is not an issue. π generator certainly has great potential however, there are few issues that can keep it from being as “convenient” as U16807. As already mentioned, π RNG requires a very large database to read the random numbers from in order for it to work well and indefinitely. Due to π having no proven strong patterns in its number sequence to date π does give us luxury of having a good cycle free generator. Overall, provided that the π digit database is large enough or resources for calculating π as you generate RNs is not a factor π can be considered an excellent random number generator.

4 References

[1] http://www.math.rutgers.edu/~cherlin/History/Papers2000/wilson.html

[2] http://www.math.tamu.edu/~dallen/masters/alg_numtheory/ π.pdf

[3] http://www.numberworld.org/misc_runs/ π-10t/details.html

[4] http://mathforum.org/library/drmath/view/57045.html

[5] http://www.jstor.org/stable/pdfplus/1403789.pdf?&acceptTC=true&jpdConfirm=true

[6] http://mathfaculty.fullerton.edu/mathews/n2003/montecarlo πmod.html

[7] https://code.google.com/p/jmathplot/ -- JMathPlot A Π For 3D Java plotting

[8] https://github.com/ikrogers/Operations-Research- π-RNG-Java-Source