Efficient Retrieval of Irrational Numbers

LAHS-3

Supercomputing Challenge final report

Student: Elijah Pelofske

Teacher: Adam Drew

Mentors: Gordon McDonough, Adam Drew

Abstract

This project focuses on developing python code that efficiently stores segments of non-terminating numbers that makes retrieval of specified ranges or specific digits of that given number more efficient. The code takes repeating segment lengths of the given number and converts them to integers, and then finds the square root of that number, and stores all of them in a list. Code was developed for this project which also combines selected digits of given non-terminating numbers in a repeating pattern. In order to find the requested range of the digits of the stored number, the selected item in the list is raised to the power of two. This presents an alternative way to store the non-terminating digits of any transcendental or irrational number, in any given range. In addition, non-terminating numbers offer a method to retrieve random numbers.

Two pieces of code are presented in this paper, the first, which can be found in appendix A, does in fact effectively store digits of mathematical constants. The second, which can be found in appendix B, does not accurately store all digits of the mathematical constants.

Purpose

The purpose of this project is to create python code that efficiently stores the non-terminating digits of irrational or transcendental numbers.

Methods

When finding the digits of a non-terminating number, there are two ways to do that. The first is brute force calculation. The second is simply storing all of those digits. Each of these methods demonstrates the trade-off between computing power and memory. This project offers a medium between these two options. Storing all of the digits of ⅓ would be completely useless, and so this project only stores useful unique numbers such as pi. The storage aspect is a list of floating point numbers. The computation side of this project is simply raising the floating point number to the power of two. Through trials with the code in appendix A up to 10,000 digits of each of the numbers used, it was found that 15 digit length segments of the number was the longest possible length without giving errors in the calculations. Rounding was also implemented for all of the calculations, which significantly increases the digit length of the number that can be stored.

Results

The python code developed for this project outputs a series of floating point numbers that corresponds to the given strings of non-terminating numbers in the code. The output from appendix A shows the condensed version of the first 90 digits of 5 important mathematical constants. The code in appendix A uses the first 100 digits of those 5 mathematical constants, but the condensed version only uses the first 90 because 90 is divisible by 15. The code breaks the given string into 15 length segments. An artifact of the function in appendix A is that zero's at the beginning of the number, that is a segment of the given numbers digits, are not shown because that string is converted to float. This means that if used in the real world, once the calculation was

performed, the length of the resulting number would be subtracted from 15. That number is how many zeros would be appended to the front of that number.

The code found in appendix B is very similar to the code in appendix A. Appendix B is simply underneath the code for appendix a. The difference is that the code in appendix B interlaces the digits of two given mathematical constants together so that their digits are repeating in a constant pattern. For example, in appendix B, the first 100 digits of pi and e are combined into a repeating pattern. Segments of that number formed are then applied to the same function used in appendix A. To make sure that there is an equal number of the digits from both numbers, this program uses only 14 digits instead of 15 like the code in appendix A. It is certainly possible to extend the number of concatenations done, and that could then adjust the number of digits that could be equally stored. The important part for this program is that all of the digits used should be of equal length otherwise indices might get out of range. Order can also be accordingly adjusted for interlacing all of these digits. A problem found with these two pieces of code is that when the program concatenation begins with a zero, that zero is disregarded, and so these two pieces of code do not represent accurate digits of the given numbers, specifically deleted zero's.

Conclusions and Applications

In addition to making creating a program that condenses ranges of non-terminating numbers, this project offers a way to generate and store random numbers. Numbers that never end are a way to generate random numbers, but the problem is that for them to be truly random, you have to use massive amounts of memory to store their digits. The advantage is that irrational and transcendental numbers always have the same digit order and never end. In terms of efficiently storing digits of these numbers, the program as seen in appendix A is not as efficient as simply storing the original digits in a file. The advantage offered is that you can get ranges of numbers very quickly by a simply calculation. The pieces of code in appendices A and B only use the first 100 digits of 5 important mathematical constants. However, these

programs can handle any non-terminating number to any realistic digit, assuming time is not a problem.

The method presented in the code appendix B does not work for the purpose of storing accurate digits because of how concatenation works with zero's. However, the code in appendix A fulfills the objective of this project. Future uses in random number generation would an excellent next step.

Acknowledgments

I would like to thank my mentor, Gordon McDonough, and my teacher, Adam Drew.

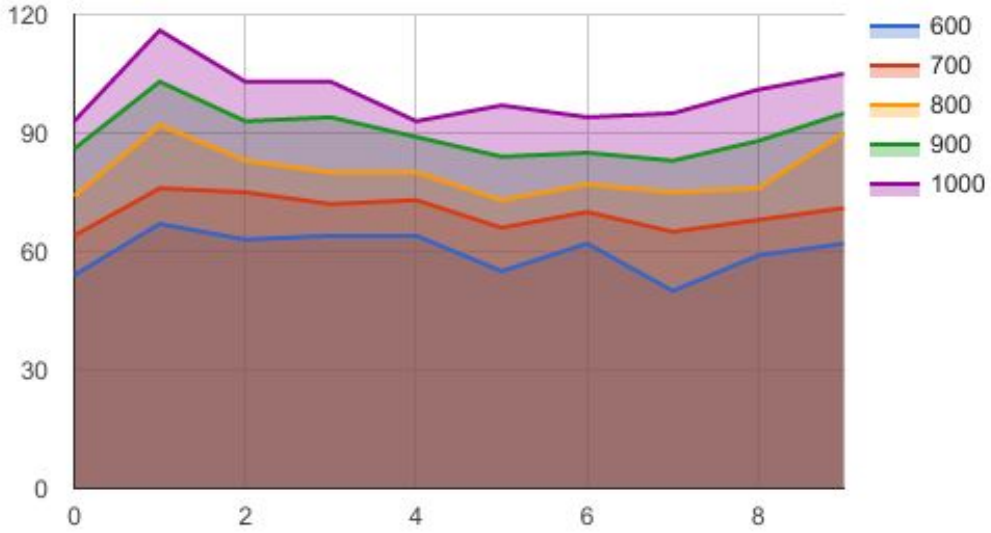
References

Using pi Digits to Generate Random Numbers: A visual and Statistical Analysis. Ilya Rogers, Greg Harrell, and Jin Wang. http://worldcomp-proceedings.com/proc/p2015/CSC2676.pdf.

Graphs

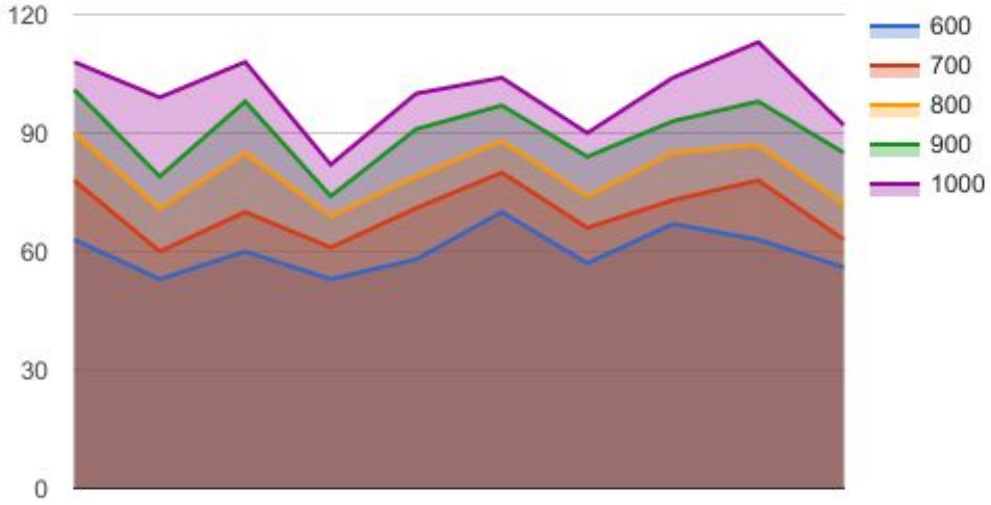
These two graphs show a very small example of trends in both the digits of pi and the square root of two. Given that these numbers do not terminate, different trends exist in different ranges of these digits. Analysis of trends like these would be important if using the never ending digits of irrational or transcendental numbers as random number generators.

Pi



The x-axis is numbered 0-9, the y-axis is a digit count. Each of the 5 lines shown are the first given digits of pi. For example, the blue line shows the digit count of the numbers 0-9 in the first 600 digits of pi.

Square root of 2



The x-axis is numbered 0-9, the y-axis is a digit count. Each of the 5 lines shown are the first given digits of the square root of 2. For example, the blue line shows the digit count of the numbers 0-9 in the first 600 digits of the square root of 2.