CSC 160
February 27, 2011

The Pi Project

Intent and Purpose

The goal of the Pi Project is to write code for five well-established methods of approximating the value of pi and analyze the value of each approach. The five methods include Archimedes' geometric formula, infinite series devised by Leibniz, Wallis, and Newton that converge to approximately pi, and the Monte Carlo Dart Simulation. Archimedes' formula arrives at the value of pi by computing the perimeter of an inscribed n-gon inside a circle of radius 1. Leibniz's series is characterized by the equation $\frac{\pi}{4} = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$. Wallis's formula is: $\frac{\pi}{2} = \prod_{n=1}^{\infty} \left[\frac{(2n)^2}{(2n-1)(2n+1)} \right]$ Newton's formula is: $\pi = \frac{3}{4}\sqrt{3} + 24 \int_0^{1/4} \sqrt{x - x^2}\, dx$. The Monte Carlo Dart Simulation uses the ratio between the top right quarter of a square and the quarter of an inscribed circle. After simulating thousands of random dart throws, the ratio between the hits inside the circle to the total number of throws approaches pi/4.

I then analyze the results from executing my code to investigate several important aspects of each method. Aspects include: how long it takes for each to converge to pi, based on the number of sides, iterations, or darts; the time it takes for the program to execute and return results for each method based on the number of iterations; the accuracy of each approximation after a certain amount of time or iterations. I also look for the "fastest" function. Evaluation of each method, with respect to such characteristics, will allow me to make conjectures about the usefulness and value of each method.

Method

In order to analyze the data, I first had to write the code necessary to employ each method. For the Archimedes formula, I used a function that received the number of sides, N, as the input. The function then calculated the measure of the interior angle (360/N degrees), and took the sine of the interior angle (sin(360/N)). Because the circle has a radius of 1, the length of one half of a side of the inscribed polygon is sin(360/N). Multiplying this value by 2 gives us one side length. Multiplying this length by N gives us the perimeter of the N-gon. Using the knowledge that the circumference of a circle is equivalent to pi*diameter, the diameter is 2*r (r=1, therefore diameter=2) and assuming the perimeter of the N-gon is approximately equal to the circumference, we can calculate pi to be approximately equal to the perimeter of the N-gon divided by 2. Higher values of N (in this case, number of sides) lead to a closer approximation and smaller error due to the increasing similarity between the circle and the n-gon.

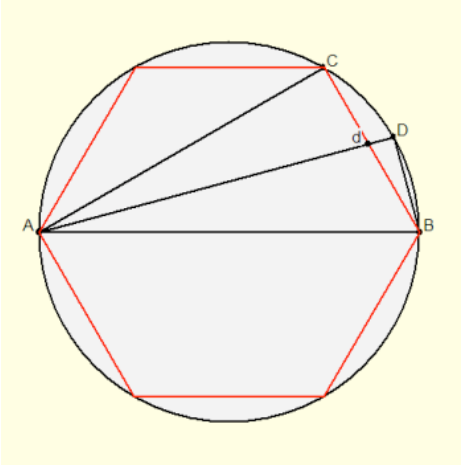

Figure 1: The inscribed n-gon becomes closer and closer to representing the circle as n increases and the chords formed by equally-spaced points along the circle become shorter.

The Leibniz series is coded using a for-loop and summation of terms. Each term can be defined by the expression (4* (-1)^k)/(2k+1). My code evaluates each term for an integer value of k from 1 to N and sums the terms together. The greater the number of iterations (N) the greater the accuracy of the function, and the longer it takes to calculate. The Leibniz alternating series appears as such:

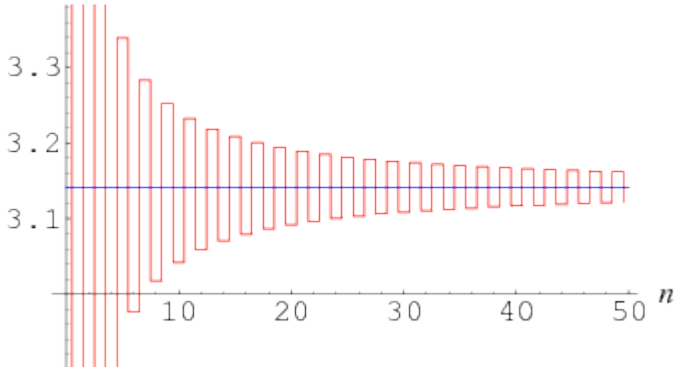

Figure 2: Graphical representation of the Leibniz series. According to Wikipedia, Leibniz converges slowly, requiring 5,000,000,000 terms to calculate pi to 10 accurate decimals.

The Wallis series is also coded using a for-loop and summation of terms. Each term in Wallis is defined by the expression ((2+2k)^2)/((1+2k)*(3+2k)). The code evaluates each term for integer value k from 1 to N. This expression approaches the asymptote of pi/2 from below, so the product must be multiplied by 2 in the end. Again, higher values for N result in a closer approximation and more time for calculation.

Newton's formula is the most complicated. The form seen in the introduction is represented in my code by a compounding for-loop. The loop uses the values for each part of the term from the previous iteration to improve efficiency. Each term to be summed consists of a numerator and 2 denominators. The numerator is defined by the equation a=a*(2k+1), where a is initially 1. Denominator 1 is defined as b=b*(2k+2), where b is initially 1. Denominator 2 is defined as c=(c/(2k+1))*4*(2k+3), where c is initially 2. Evaluating for integer values of k from 1 to N leads to a fairly accurate approximation within just a few iterations, eliminating the need for huge values of N for accuracy.

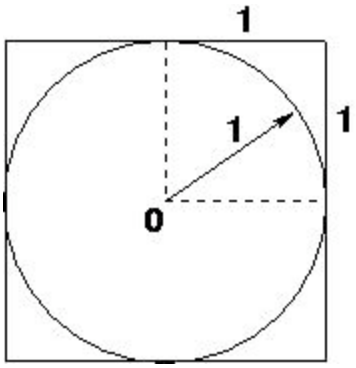

Figure 3: "Dart board" used in Monte Carlo Dart Simulation.

The Monte Carlo Darts Simulation is an interesting approach to approximating the value of pi. The Simulation uses a square of side length 2 and an inscribed circle (of radius 1) as the dart board. Looking at the top right quarter of the square and assigning this as the first quadrant with the origin at the center of the circle, the Simulation uses "random dart throws" to assess the ratio between the quarter circle and the quarter square. If the dart lands within the circle (that is, the coordinates (x,y) of the dart's location satisfy the equation x^2+y^2<=1), 1 count is added to the "hits" category. The ratio of hits to total throws is approximately equivalent to (pi/4):1, as the area of the quarter circle is (pi*radius^2)/4=(pi*(1^2))/4=pi/4 and the area of the quarter square is 1. Multiplying this ratio by 4 provides an approximation of pi.

The functions are compiled in a main script file and are executed for N values of 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 300, 400, 1000, 5000, 10000, 50000, and 100000. This range of values allows for analysis on a very small level over the first few terms, and for analysis at a much higher level which some functions may require in order to achieve higher accuracy. Two further functions serve to display the results in color-coded graphs. For the purposes of this report, some of these graphs have been resized to show important sections that are not easily visible on the original scales.

Results

The raw data presented by the function shows that all methods do approach a fairly accurate approximation of pi as N increases, as shown in Figures 4 and 5 below.

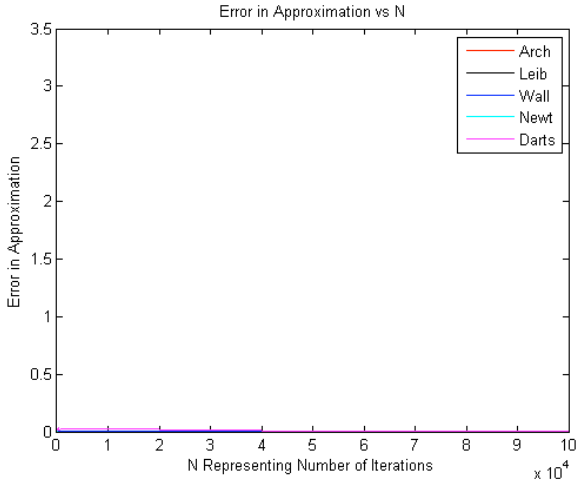

Figure 4: Full-scale graph of error in all approximations vs N

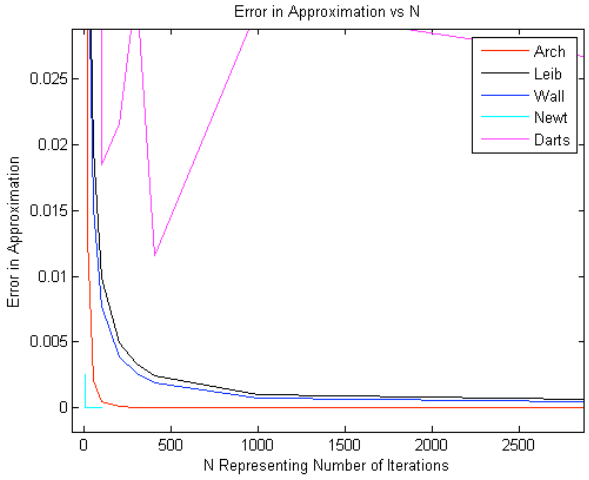

Figure 5: Small scale graph of error in approximation vs N

As shown in Figure 4, all five methods have an error approaching 0 as N approaches 100000. Figure 5 represents the errors at a smaller scale, only up to about N=3000. At this minute scale, it is clear that Newton's method converges for the smallest value of N, as the error approaches 0 most rapidly in the graphs. Archimedes is the next-quickest to converge, followed by Wallis, then Leibniz, and finally the Monte Carlo Dart Simulation.

However, being the quickest to converge in terms of iterations or sides does not always guarantee that the function is the fastest to converge when executed in MATLAB. Figure 6 (below) represents the time elapsed for the

calculation of each function for N iterations, darts, or sides.

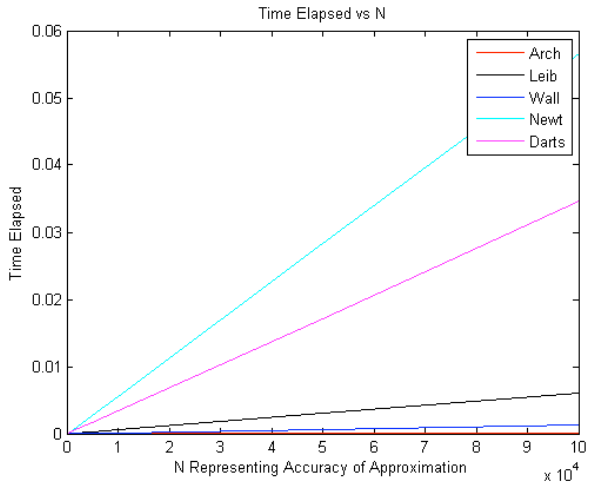

Figure 6: Full-scale graph of time elapsed for each function to execute vs N

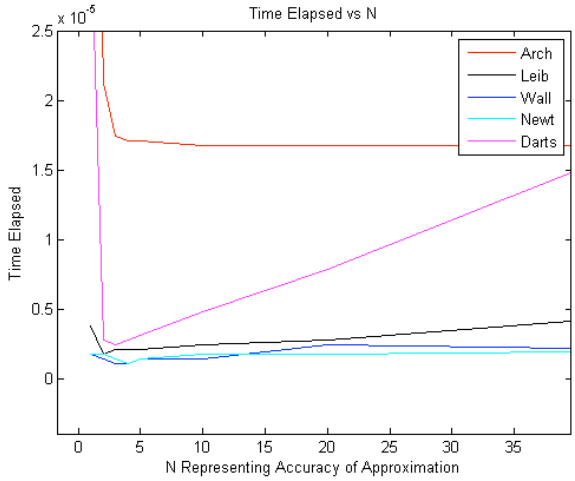

Figure 7: Smaller scale graph of time elapsed vs N

The full-scale graph clearly shows that Newton's method is the most time-consuming method at large values of N, followed by the Dart Simulation, then Leibniz's method, then Wallis's, and finally Archimedes. The smaller-scale graph, however, outlines an important aspect lost in the full-scale graph. The time elapsed for Newton's, Wallis's, and Leibniz's functions have remote differences at small values of N, differences that could amount to random processor fluctuations on the part of the computer running the program. Additionally, all three are significantly faster than Archimedes's function for low values of N. The Dart Simulation is so imprecise that even while it is faster than Archimedes at low values of N, it is impractical.

Data for Error in Pi Approximation (difference between MATLAB built-in value of pi and calculated approximation)

	Archimedes	Leibniz	Wallis	Newton	Monte Carlo
N=20	0.012903352785176	0.047592128687803	0.036315330756439	0.000000000000000	0.258407346410207
N=100000	0.000000000516772	0.000009999899927	0.000007853854235	-	0.002927346410207

Data for Time Elapsed (time for execution of function for value of N, in seconds)

	Archimedes	Leibniz	Wallis	Newton	Monte Carlo
N=20	0.000016767086688	0.000002737483541	0.000002395298098	0.000001710927213	0.000007870265180
N=100000	0.000016767086688	0.006079608758852	0.001345473160343	0.056567702074191	0.034594263876988

Archimedes

Archimedes is a rather good approximation of pi because it converges quickly and has a curious property that allows large values of N to be calculated quickly.

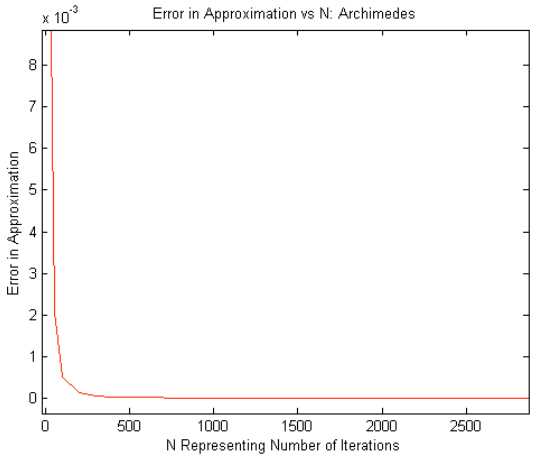

Figure 8: Error in Archimedes vs N, small scale

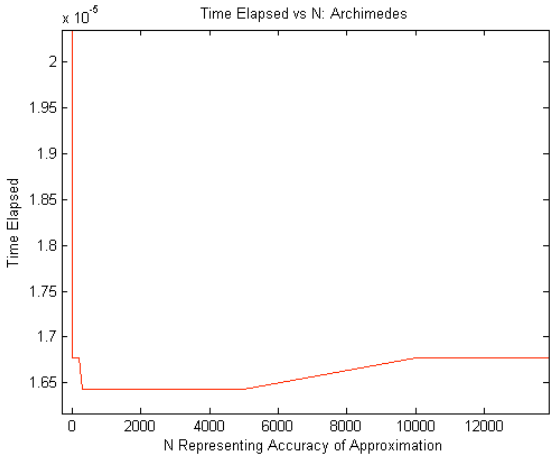

Figure 9: Time elapsed for Archimedes vs N, small scale

Archimedes presents an interesting case, as it is the only method that does not employ a for-loop in its construction. The for-loops require further time to calculate due to the numerous iterations; whereas Archimedes runs its lines of code only once, the other functions must run the loop lines N number of times, dramatically increasing the time elapsed. Instead, Archimedes's method is simply a series of algebraic and trigonometric calculations. Due to its simple design, this method requires a nearly-constant amount of time to execute, virtually regardless of the number of sides. Slight variations occur, specifically near N=0, but for large values of N where Archimedes has proven to be most accurate, this holds fairly true, as is shown in the graphs. Data collected from the program verifies that the time elapsed for N=20 is the same as the time elapsed for N=100000 (both are 0.000016767086688 seconds). This fact gives Archimedes an advantage over the other functions, as it can become increasingly accurate without increasing the time it takes to execute. Unfortunately, Archimedes's method does have a limit. At insanely high precisions, the interior angle used in the Archimedes function approaches 0, as the "interior angle" of a circle is non-existent for the smooth curve when the chords are no longer present. Without an interior angle, the sine function cannot be applied, making the Archimedes method inapplicable.

Leibniz

The Leibniz function is a rather average approximation tool. At high values of N, it takes the third longest to execute, and it has the second highest error.

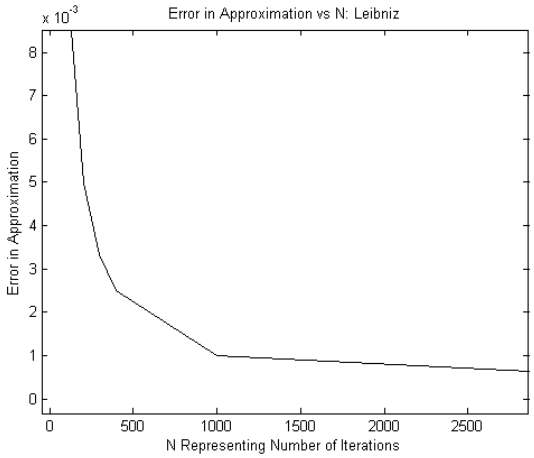

Figure 10: Error in Leibniz, small scale

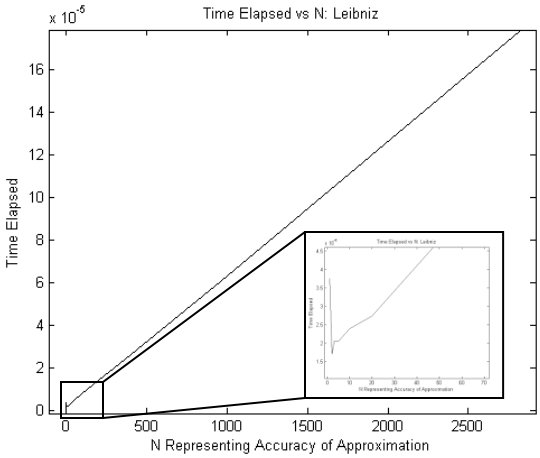

Figure 11: Time Elapsed vs N, Leibniz, small scale

The Leibniz function converges at a fairly slow rate (compared to better functions such as Archimedes or Newton), and it takes a fairly long time to do so. As Figure 11 shows, after the initial few terms, the time elapsed becomes fairly linear and increases as N increases. Leibniz is a rather unremarkable function, as there are better options for both quickness to convergence and speed of execution.

Wallis

The Wallis function is similar to Leibniz in its rather unremarkable nature. At high values of N, it takes the second longest to execute, and it has the third highest error.

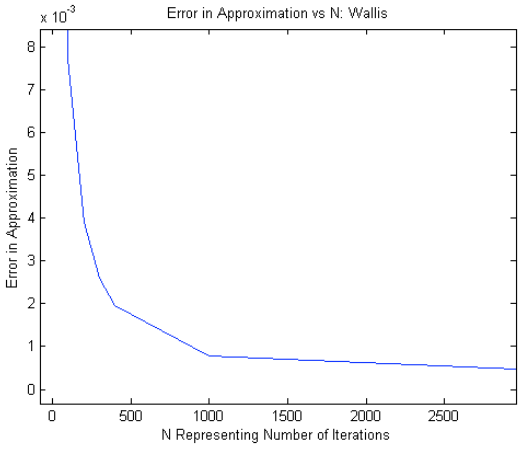

Figure 12: Error in Wallis, small scale

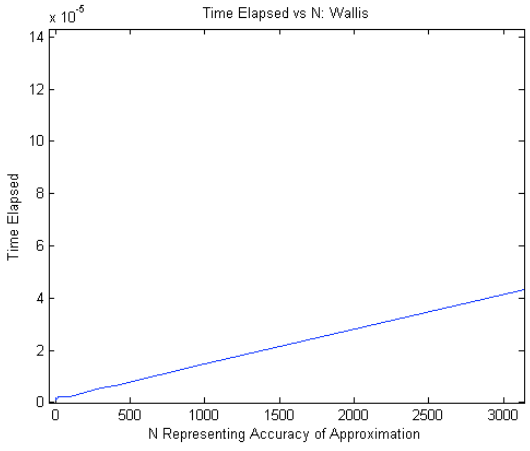

Figure 13: Time Elapsed vs N, Wallis, small scale

The Wallis approximation is better than the Leibniz method, as it both converges and executes faster (as shown by Figures 12 and 13, at approximately the same scale as Figures 10 and 11). Wallis still, however, is neither as fast nor as accurate as Archimedes or Newton, although it may be applicable

at such high values of N where Archimedes cannot be used. Newton may still be more accurate, however, as the Newton function evaluates to an indeterminably small error at N=20.

Newton

The Newton function appears to be one of the better methods tested, as it has the greatest precision, and it executes the fastest at the value of N for which is has the greatest precision (that is, N=20).

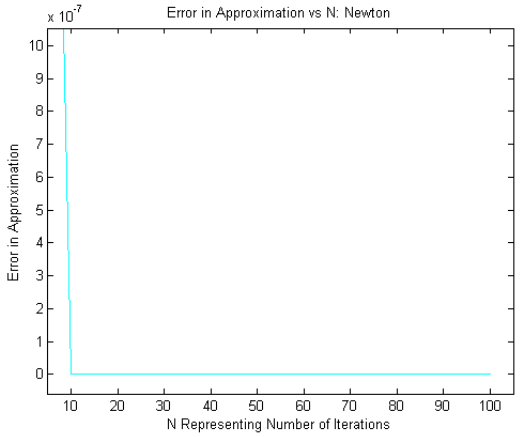

Figure 14: Error in Newton, small scale

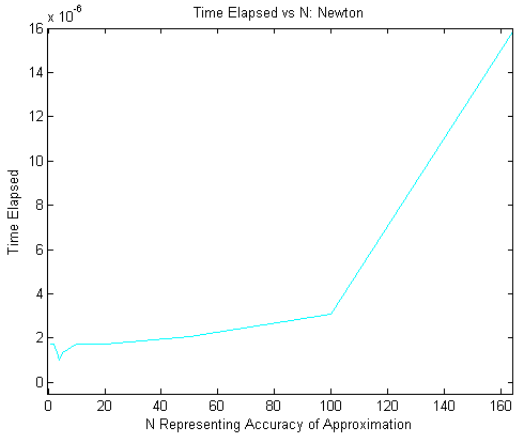

Figure 15: Time Elapsed vs N, Newton, small scale

The Newton method is very peculiar in that it reaches a precision of .000000000000000 in only 20 iterations (N=20). This is the quickest convergence of any function by far, and it also takes a very short amount of time (quicker than Archimedes at highest precision) to execute. Thus, although Newton takes the longest to execute at high values of N, this is irrelevant, as the greatest precision is achieved at N=20, where Newton is very fast in its execution.

Monte Carlo Dart Simulation

The Monte Carlo Dart Simulation is an intriguing idea, but it is completely impractical for approximating pi, as it is the second slowest at high values of N and is wildly inaccurate throughout.

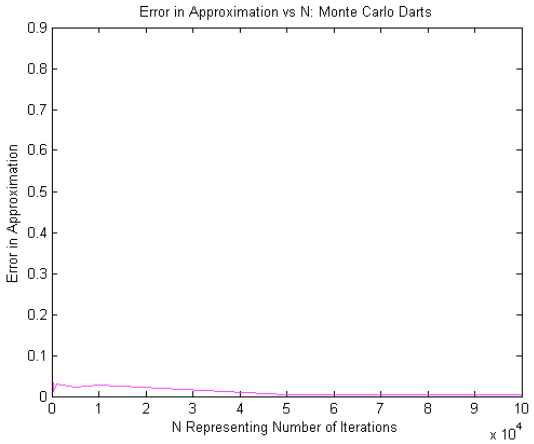

Figure 16: Error in Monte Carlo, full scale

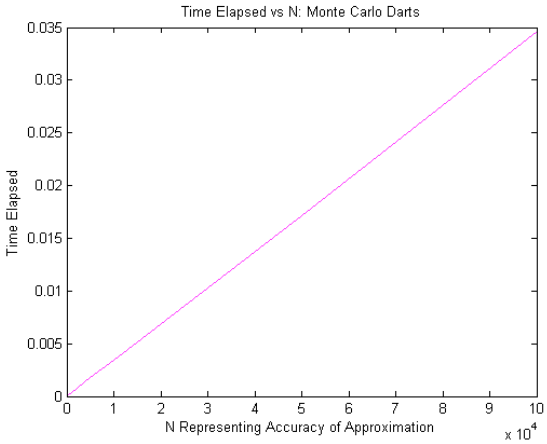

Figure 17: Time Elapsed vs N, Monte Carlo, full scale

Even seen at full scale, the graphs of the Monte Carlo Dart Simulation data show that the approximation takes a very high value of N to even begin to approach any precision similar to the other functions, and the time elapsed is greater than any other function, save Newton's method. The Monte Carlo Dart Simulation should not be used to accurately or quickly approximate pi.

Conclusion

Newton's method has an error that MATLAB calculates as 0.000000000000000 (using the long format) after only 20 iterations. The greatest precision achieved by any other method is an error of 0.000000000516772. This precision requires a 100000-sided polygon using Archimedes method in order to be calculated. For comparison, the greatest precision achieved (both after 100000 iterations) by Leibniz's method is 0.000009999899927, and the greatest precision for Wallis's method is 0.000007853854235. Analysis of Monte Carlo Dart Simulation reveals that it is impractical, as its precision at N=100000 is only 0.002927346410207.

Newton's method takes 0.000001710927213 seconds to calculate at N=20, where its precision is highest. The Archimedes method takes 0.000016767086688 seconds to execute at its highest precision (N=100000), and indeed for most values of N. Leibniz and Wallis take 0.006079608758852 and 0.001345473160343 seconds, respectively, to calculate at their highest precisions. The Monte Carlo Dart Simulation takes 0.034594263876988 seconds to calculate at highest precision, although seeing as how this precision is barely as accurate as the Newton function after only 1 iteration, it is fairly useless.

The data shows that the Newton function evaluates the fastest and provides the most precise approximation of pi; therefore, the Newton method is the most practical method available. Although the Archimedes method has a wonderful ability to evaluate at a constant speed regardless of the size of N, it reaches a limit at incredibly high precision, and it is also not as precise as Newton, even at high values of N. Leibniz and Wallis provide fine approximations of pi, but they cannot compete with Newton or Archimedes when it comes to speed or precision. However, Wallis may be useful if you find Newton and Archimedes to fail at insanely high values of N. The Monte Carlo Dart Simulation, as stated before, is interesting but ultimately useless compared to the other functions.

References

Figure 1: http://delphiforfun.org/Programs/Math_Topics/Archimedes_PI.htm

Figure 2, Newton Formula: http://mathworld.wolfram.com/PiFormulas.html

Wallis Formula: http://mathworld.wolfram.com/WallisFormula.html

Leibniz Formula: http://en.wikipedia.org/wiki/Leibniz_formula_for_pi

Figure 3: http://www.cs.rochester.edu/u/brown/160/assts/Pi/Pi.html

All other figures and data created using MATLAB and code written by Sam Butler