Emile Goulard
uID: u1244855
CS3200 - Assignment 4 - Report

Question 1

To find the Absolute and Relative Errors, simply open “A4_Question1.m” in MATLAB. Follow the instructions provided in the README file or Header Comment.

The methods I used to find the Absolute Error of the Stirling Approximation for N Factorial is to start out setting up a 1D array to store the absolute error values and the actual solution, n!, through iteration from n=1-10. Additionally, e=exp(1) for representing Euler’s Number. In a loop from 1 to n, I use both formulas to calculate the absolute error by using:

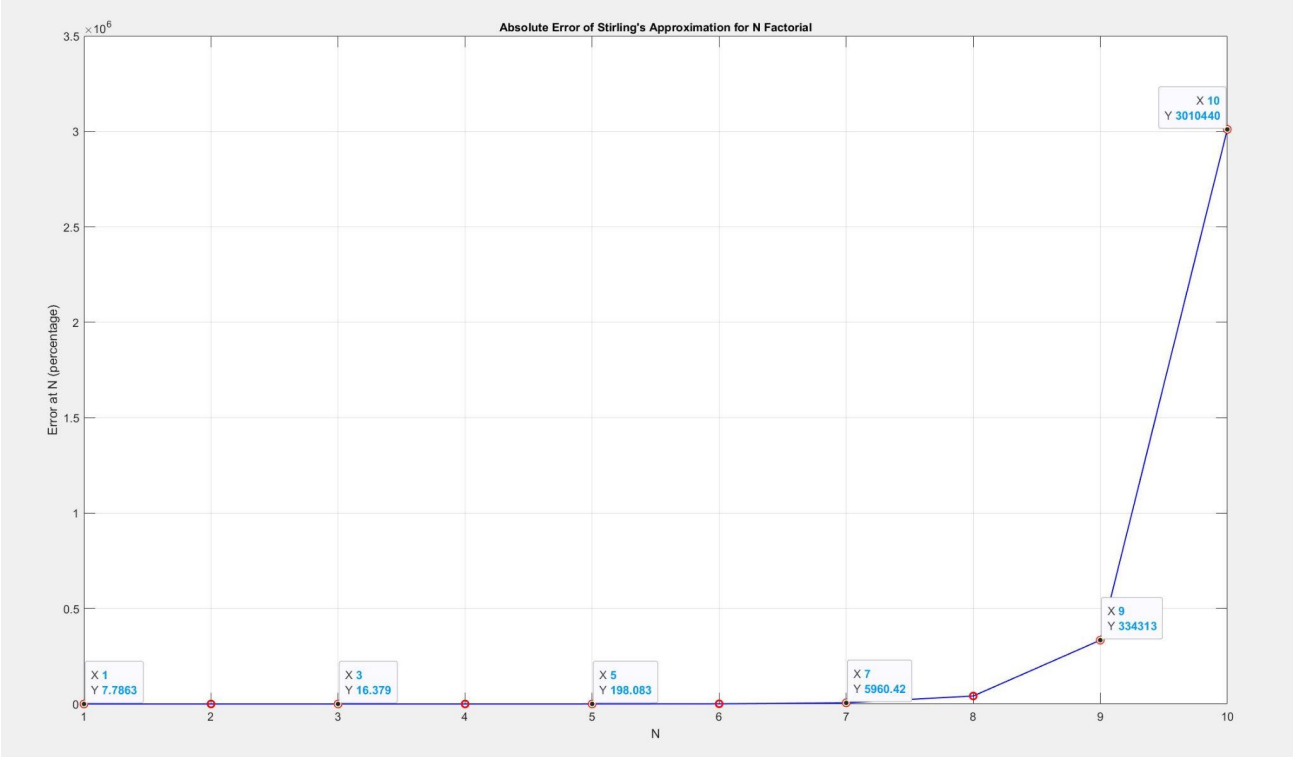
```
abs(actualSolution(i) - stirlingApprox) * 100
```

The methods I used to find the Relative Error of the Stirling Approximation for N Factorial is similar to how I set up the Absolute Error plot. The only difference is in the loop from 1 to n, the formula to calculate the relative error is being used:

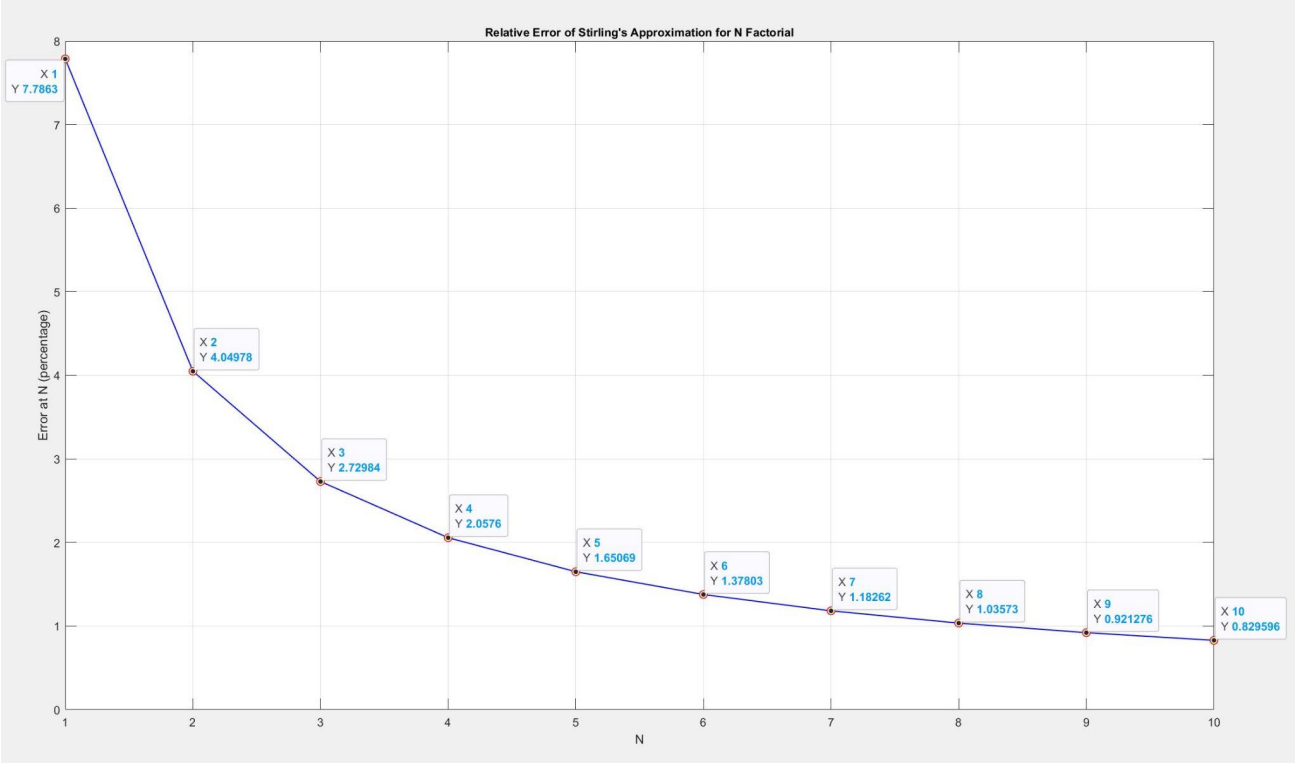
```
abs((actualSolution(i) - stirlingApprox)/actualSolution(i)) * 100
```

These methods will plot the Absolute and Relative Error Percentage of N Factorial as shown here:

Absolute Error Plot



Relative Error Plot



As N increases, the Absolute and Relative Errors become affected differently. The Absolute Error increases sharply after 9 iterations, showing that it's final error is 3010440. When subtracting from the real value of 10!, the difference is 618,360. It may seem pretty sizeable of a difference, but the Relative Error Plot shows how accurate it actually is to the real solution. The Relative Error steadily decreases and becomes more accurate to Factorial N as N increases. It shows the final error is 0.829596 which is much smaller than its initial iteration at 1. The two plots showcase how accurate the Stirling Approximation is to the real solution of Factorial N as N increases because of this proof. From this, I can only infer that with larger size N iterations, the Stirling Approximation will be more accurate because of the pattern of the Relative Error.

Question 2

To find the approximation of π/4, simply run “A4_Question2.m” and enter the accurate number of decimal places into the parameter. Additionally, there will be a Rate Of Convergence of π/4. Follow the instructions provided in the README file or Header Comment.

The methods used to find the approximation of π/4 starts with setting up variables that will be used during iteration. Firstly, the actual solution is π/4 and I used a variable called “decimals” to store 10^(numberOfDecimalPlaces). This will help determine the most accurate approximation of π/4 because we will be looπng from 0 to “decimals” to sum of the approximation using Leibniz’s Series Formula as shown here:

```
approx = approx + (-1)^(-i)/(2*i+1);
```

From there, I printed out the values for Leibniz Series approximation, the arctan(1) expansion term, and the actual value (π/4) into the command window using a ‘long’ type format. For this experiment, the number of decimal places used is 8, as it produces the most accurate value similar to arctan(1), which I will discuss soon:

Figure 1 - 8 significant decimal places

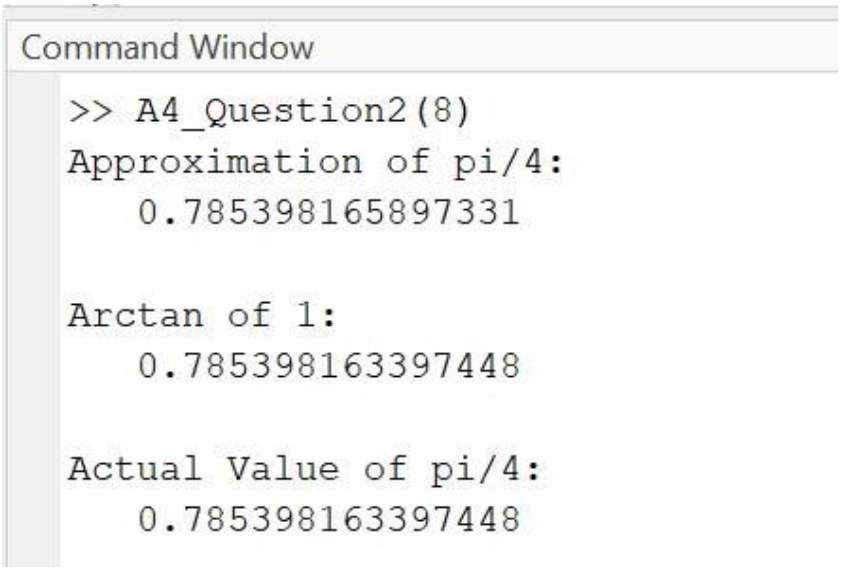


Figure 2 - 1 significant decimal place

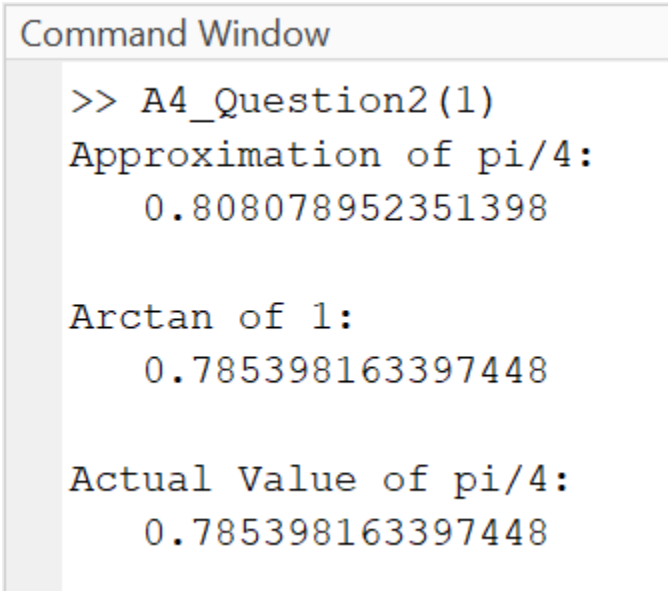
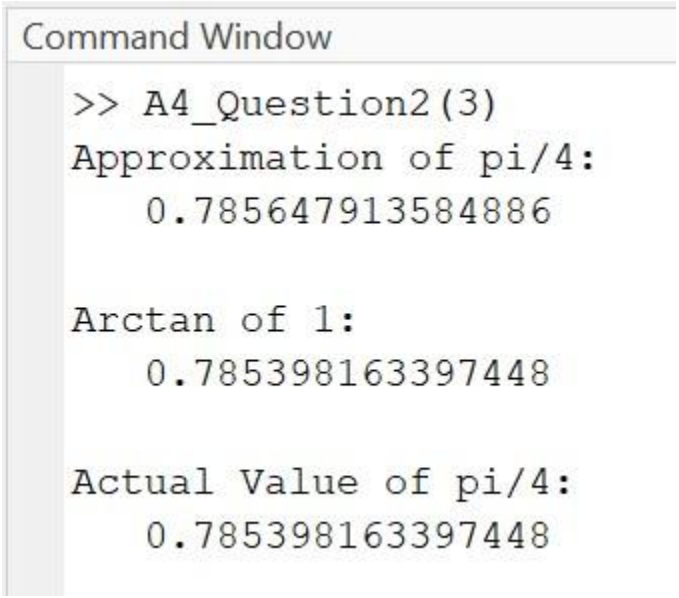


Figure 3 - 3 significant decimal places



Lastly, to find the rate of convergence for π/4, I used the Shanks Transformation formula as it was the easiest to implement into MATLAB code. To do this, I used the formula to find the i^th term:

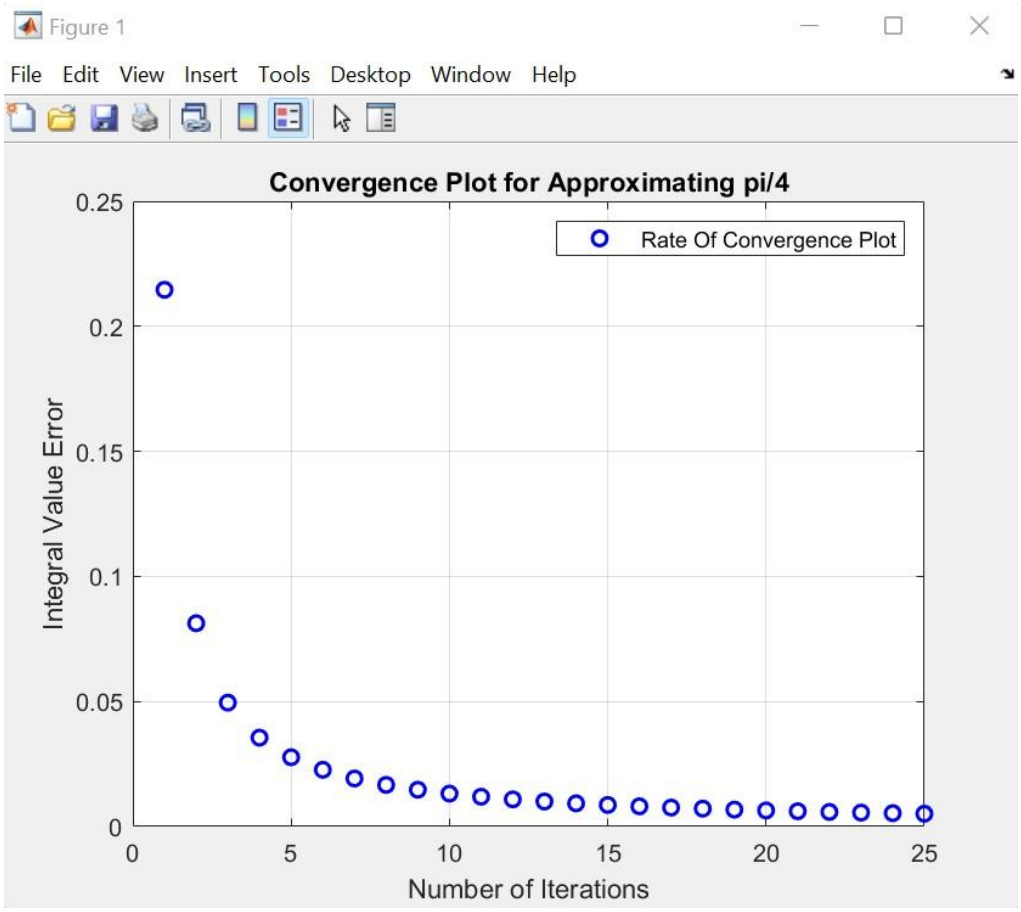
```
approx = @(i) (-1)^(i)/(2*i+1);
```

From there, I had variables that stores the current totalSum of the approximation for some iterative ‘k’ value. I then store the Rate Of Convergence array at that iteration by subtracting totalSum from the actual solution (π/4) such that we find the error percentage:

```
rateOfConvergence(k) = totalSum(k) - actual;
```

I then plot the odd terms of the Rate Of Convergence as shown here:

Figure 4



As you can see from Figure 1, **the number of significant decimal places related to the number of arctan expansion terms is 8.** This is because it is closest by the nearest billionth of a decimal to arctan(1) since the value is 0.7853981**5879331** (it slightly overestimates, but is still more accurate than 7 significant decimal places). Computationally, this means we have to iterate 10^8 times (or 100,000,000 times) which is extremely slow and computationally exhaustive. Any higher terms will be much longer to compute and any smaller terms would be a little more inaccurate because of this data. This slow convergence is supported by the data in Figure 4 which shows the sublinear convergence of approximating π/4. It is sublinear because as the number of iterations grows larger, the formula converges to 0 in a sublinear pattern. It is worth noting that initial terms are very inaccurate on the plot as supported by Figures 2 and 3.

Question 3

To find the approximation of π using the Chudnovsky Brother's implementation, simply run "A4_Question3.m" and insert an 'numberOfDecimalPlaces'' parameter which will represent the n^th iteration of the approximation (keep these values small i.e. between 1-10).

The methods used to find the approximation of π starts with setting up variables to use during iteration. The "iteration" variable stores the n^th iteration by storing 10^n which will determine the length of our loop. I also set an array called "π_approx" that will show these approximated values at N. Of course, I also store the initial variables before iteration, as told in the assignment prompt. To begin, I loop from 2 to iterations+1 to account for all values of the approximated π value and apply the Chudnovsky Brother's iteration as told in the assignment:

```
for i=2:iterations+1
        x = (1/2) * (sqrt(x) + 1/sqrt(x));
        π_approx(i) = π_approx(i-1) * ((x + 1)/(y + 1));
        y = (y * sqrt(x) + (1 / sqrt(x))) / y + 1;
    end
```

I then plotted the Absolute Error by subtracting the actual value of π from the π approximation:

```
abs(actual - π_approx(2:end))
```

Figure 1 - 'numOfDecimalPlaces' = 1

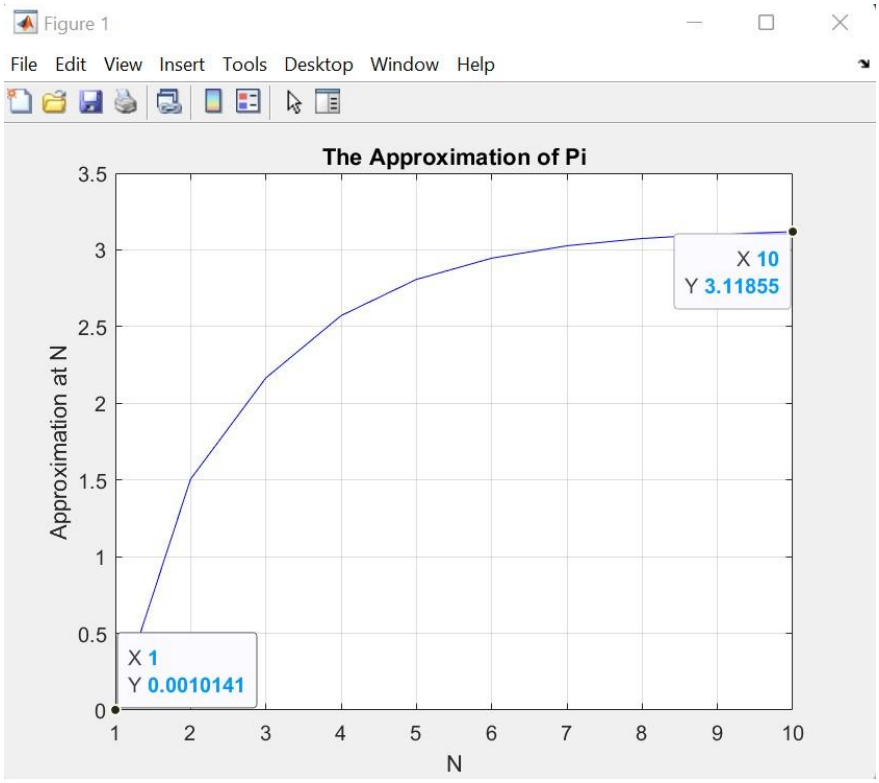
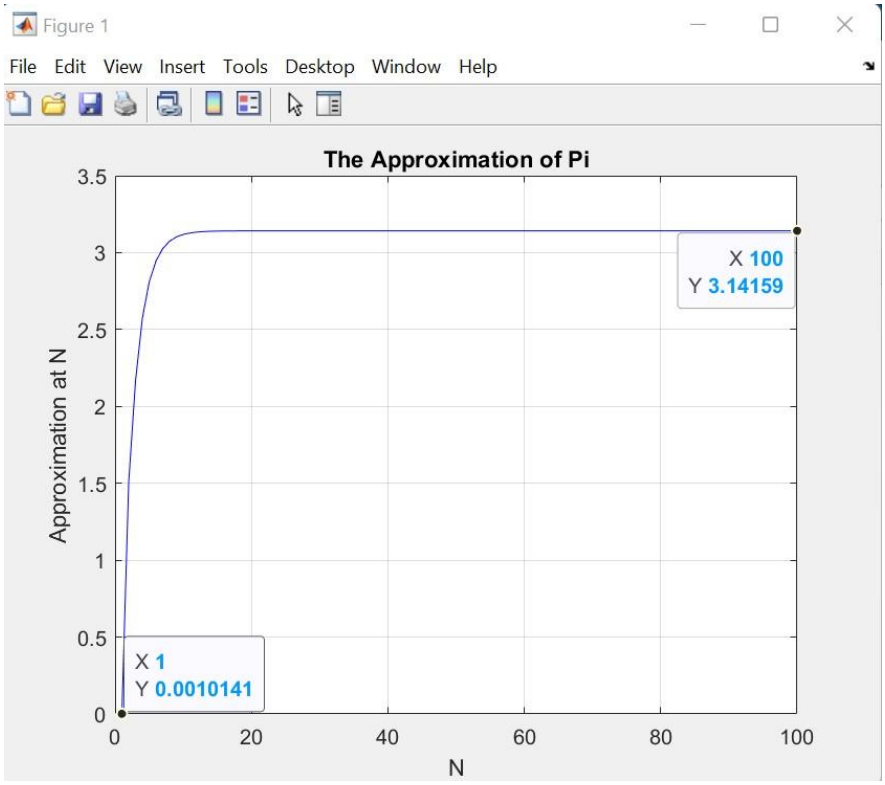


Figure 2 - ‘numOfDecimalPlaces’ = 2



Lastly, to prove that the Absolute Error Percentage is less than the specified threshold, 10^-(2^(n+1)), I used an if-conditioal statement that prints out the Absolute Error Percentage is the statement is true (which it is).

Here is the result:

Figure 3 - ‘numOfDecimalPlaces’ = 1

```
>> A4_Question3(1)
The Absolute Error of this formula (percentage):
     2.726209087833018e-05
```

Figure 4 - ‘numOfDecimalPlaces’ = 2

Command Window

```
>> A4_Question3(2)
The Absolute Error of this formula (percentage):
     2.726209087833018e-09
```

As you can see from the following Figures, **the π approximate does converge to the actual π value.** I want to emphasize that in Figures 1 and 2, the convergence to π is drastically faster at 10^2 iterations for nth terms, this proves that the convergence of the Chudnovsky Brother's Formula is quite fast. In Figure 2, it starts to converge to the actual value at roughly 15 iterations (just past the 1st term). This is also supported by the fact that the Absolute Error Percentage in Figures 3 and 4 are different in the significant decimal place making the 2nd term iteration much more accurate (as shown in Figure 2).

Overall, this means that approximating π using this implementation is much faster and more accurate than the other approximations thus far.

Sources

1. https://en.wikipedia.org/wiki/Leibniz_formula_for_%CF%80
2. https://en.wikipedia.org/wiki/Shanks_transformation
3. https://www.mathworks.com/matlabcentral/answers/549723-why-euler-s-number-e-2-71828-is-not-a-built-in-constant-in-matlab
4. https://my.eng.utah.edu/~cs3200/Errors.pdf