COE 347 Homework 2

William Zhang

February 2024

Question 1

The implementations of all ODE methods and scripts for analyzing can be found in my GitHub Repository: `https://github.com/williamzhang0306/COE347`. The solutions produced by all methods were plotted together with the exact solution (MATLAB's ode45 method with e-10 tolerance values).

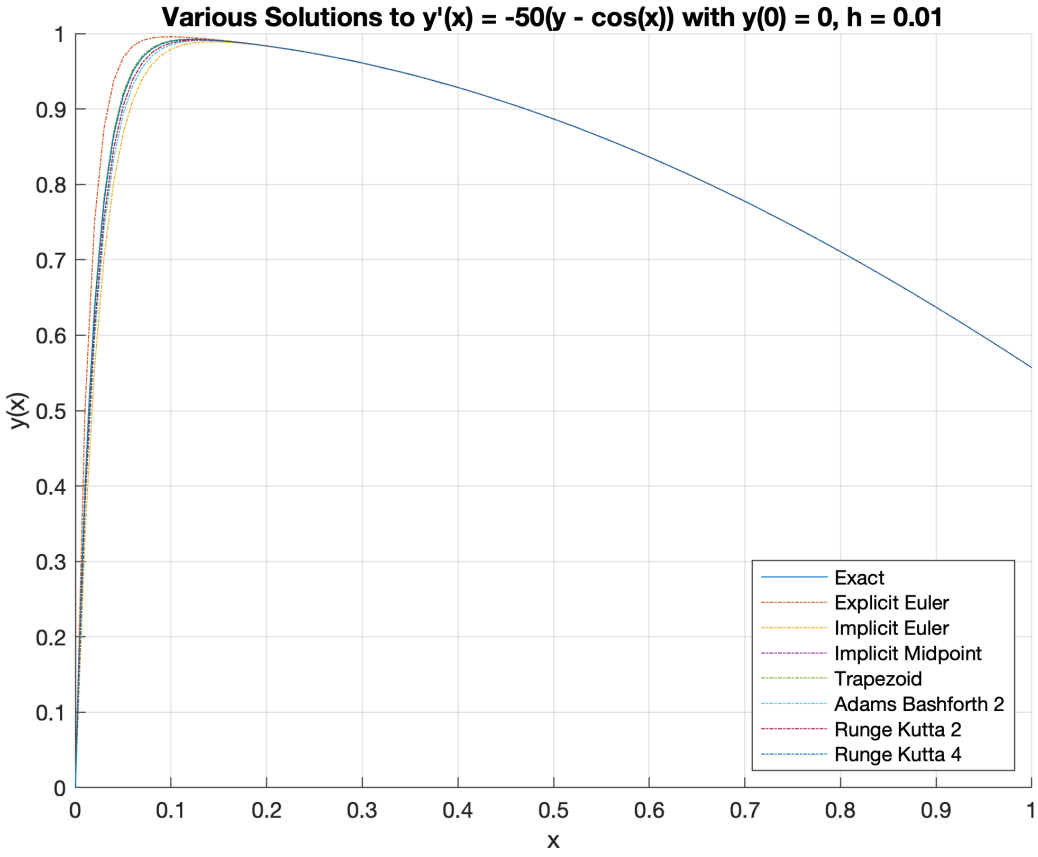

Figure 1: All Numerical Methods Visualized

For each method, the solutions produced for various step sizes were plotted against each other with the exact solution. All plots suggest that each method results in solutions that converge towards the exact solution.

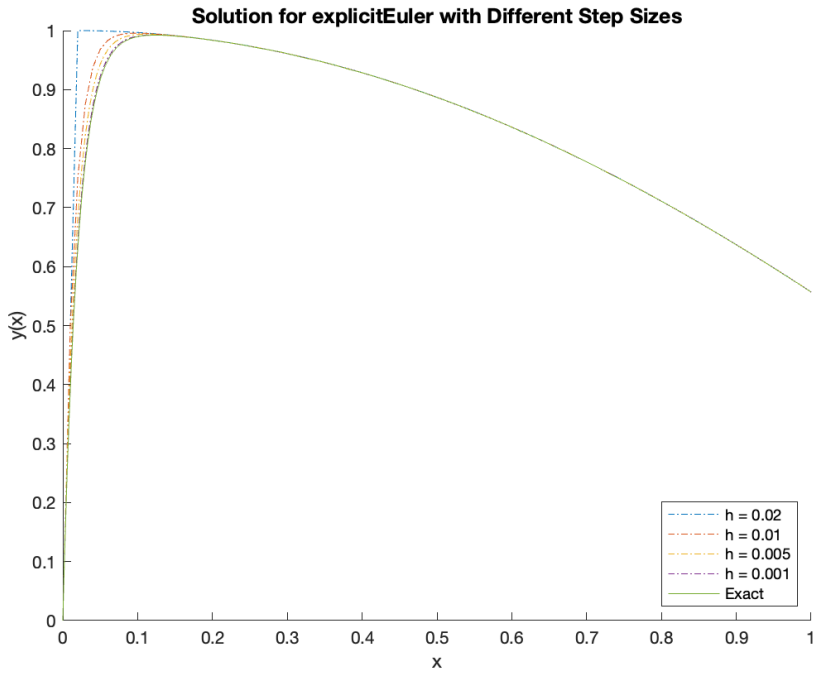

Figure 2: Explicit Euler Converging Towards the Exact Solution

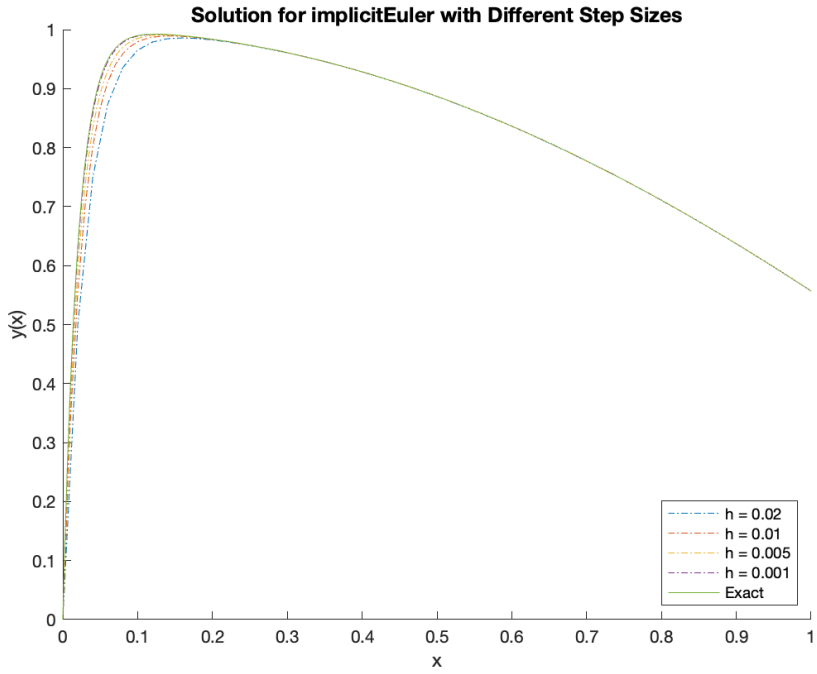

Figure 3: Explicit Euler Converging Towards the Exact Solution

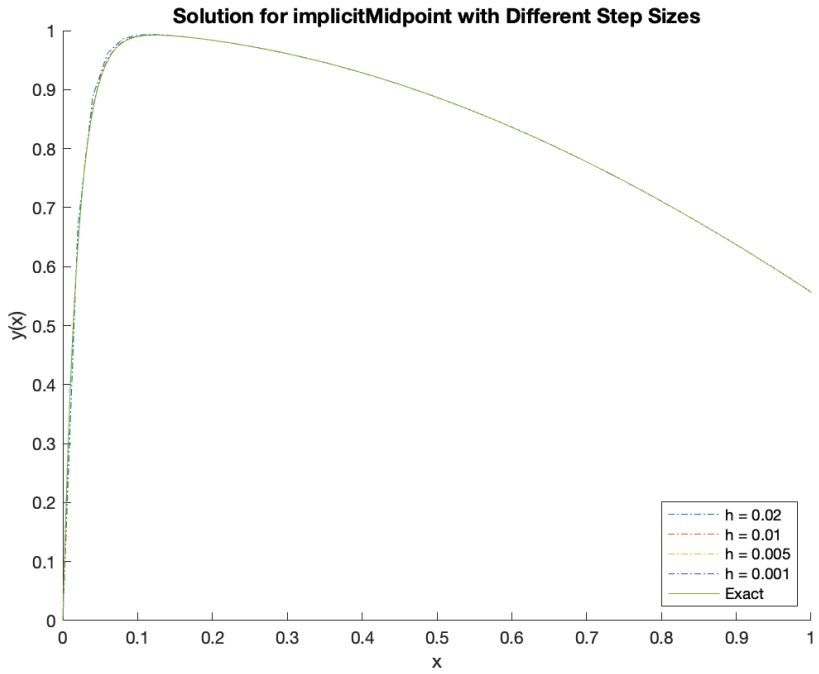

Figure 4: Explicit Euler Converging Towards the Exact Solution

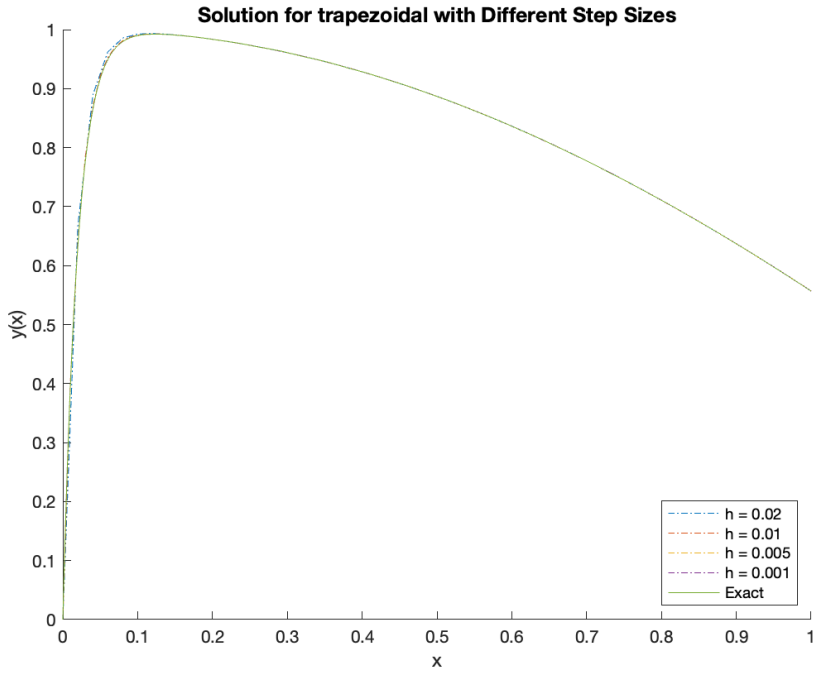

Figure 5: Explicit Euler Converging Towards the Exact Solution

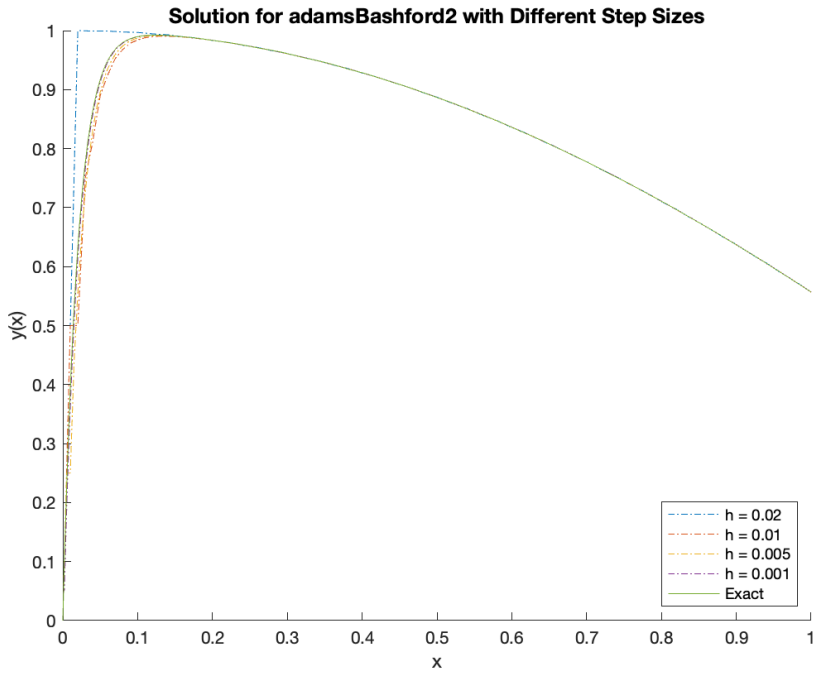

Figure 6: Explicit Euler Converging Towards the Exact Solution

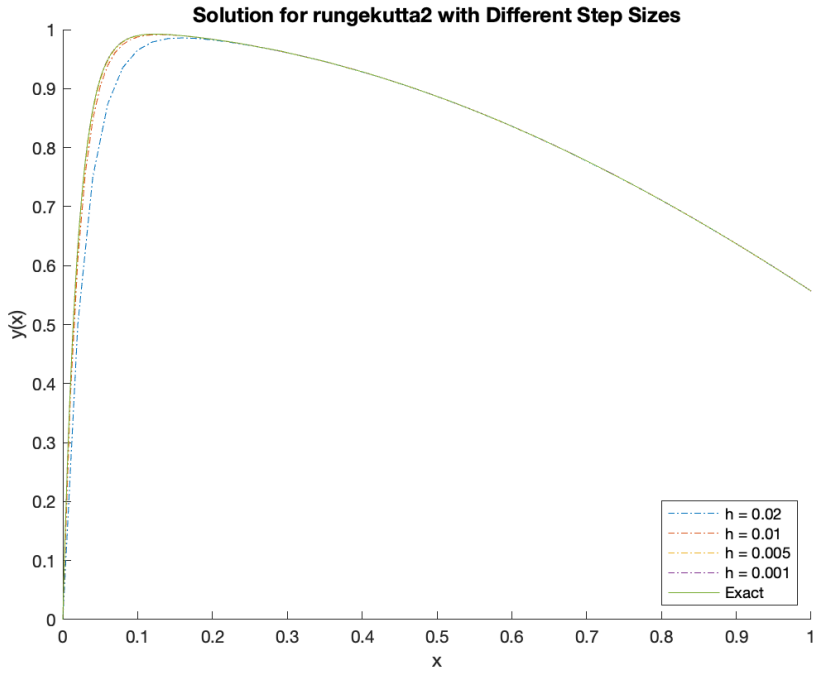

Figure 7: Explicit Euler Converging Towards the Exact Solution

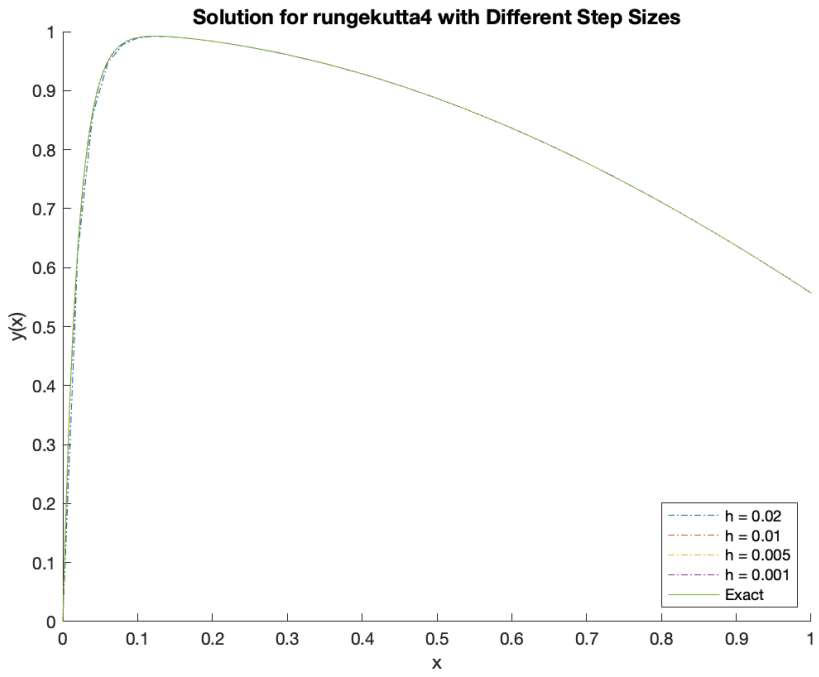

Figure 8: Explicit Euler Converging Towards the Exact Solution

For each method, the global error (i.e. the difference between the method's solution and the exact solution for y at x = 1) was calculated for a range of values for h. The global errors $e(h)$ were then plotted as a function of $\frac{1}{h}$ on a loglog plot.

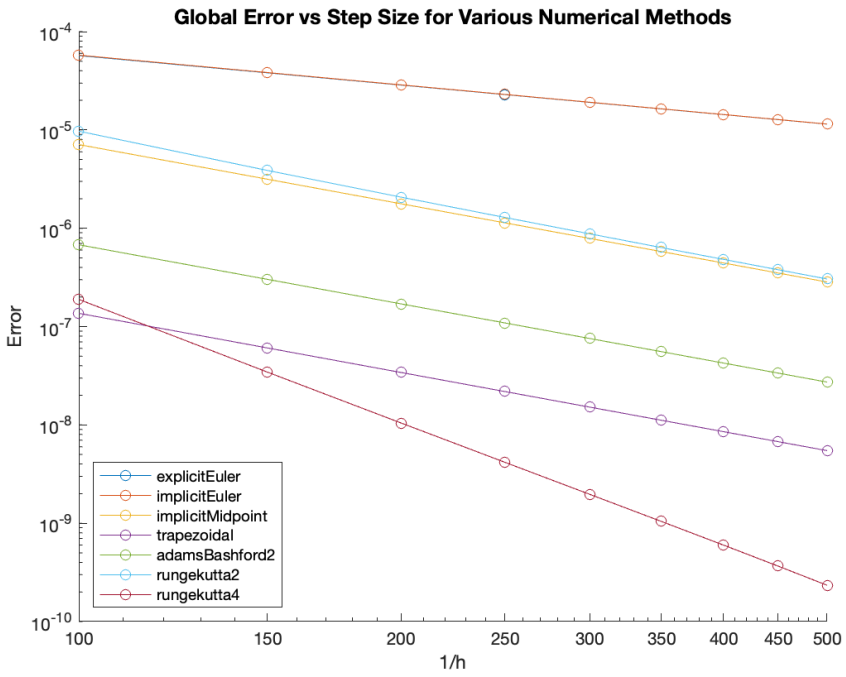

Figure 9: Explicit Euler Converging Towards the Exact Solution

At the step size $h = \frac{1}{500}$ the following global errors were calculated:

Method Name	Global Error
Explicit Euler	0.1146e-4
Implicit Euler	0.1148e-4
Midpoint	0.0284e-5
Trapezoidal	0.0055e-6
Adams Bashforth 2	0.0273e-6
Runge Kutta 2	0.0305e-5
Runge Kutta 4	0.0002e-6

Table 1: Global Error at x = 1 for step size h = $\frac{1}{500}$

Using the error vs step size data, the function $E = Ch^{\alpha}$ was fitted. This was accomplished by taking the log of the equation, $ln(E) = ln(Ch^{\alpha} = \alpha * ln(h) + ln(C)$ and using MATLAB's 'polyfit' function to solve for the coefficients α and $ln(C)$. The calculated values of α are reported below alongside the expected values based on each method's theoretical global error order.

Method Name	Computed α	Expected
Explicit Euler (First Order)	0.99788	1
Implicit Euler (First Order)	1.0021	1
Midpoint (Second Order)	2	2
Trapezoidal(Second Order)	1.9984	2
Adams Bashforth 2 (Second Order)	1.9984	2
Runge Kutta 2 (Second Order)	2.1339	2
Runge Kutta 4 (Fourth Order)	4.1428	4

Table 2: Error Order Analysis for Various Numerical Methods

A work metric was developed which is defined as the number of times the function $f(x, y)$ is evaluated when the method steps from $x = 0$ to $x = 1$. For each numerical method, the method was evaluated for a range of different time steps. Then the final global error and work metric were recorded. The error vs work data was then plotted on a loglog plot.

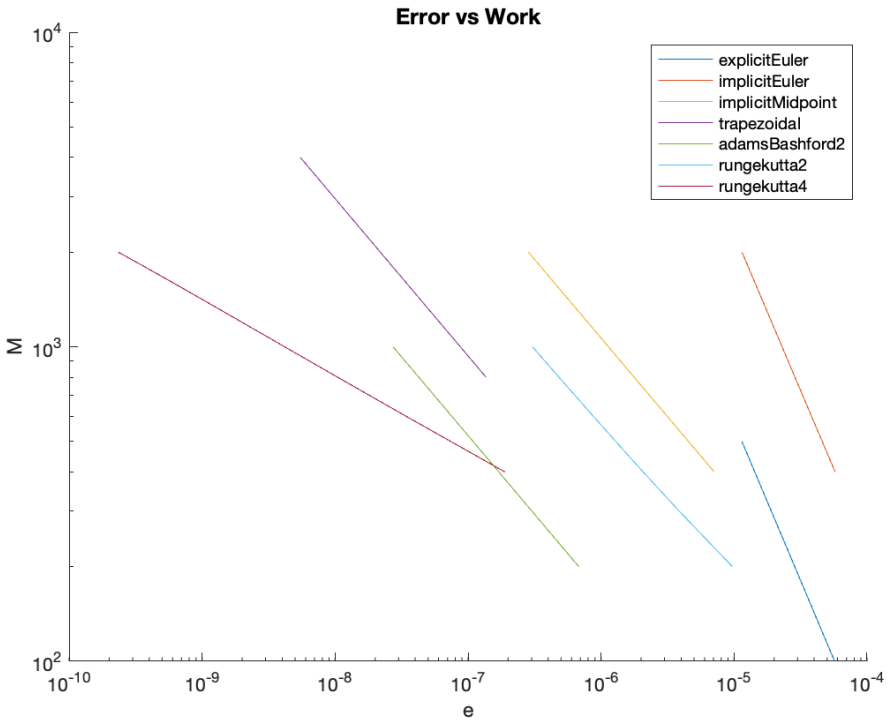

Figure 10: Error vs Work of Various Numerical Methods

One way to interpret this graph is that a point further left is a solution with higher accuracy, and a point further down is a solution found with a small amount of work. So the most efficient solution ODE solvers produce solutions that can be

found in the bottom left area.

For example, implicit Euler is an inefficient solution compared to explicit Euler. To produce the same level of accuracy, the implicit Euler method generally has to work harder.

Similarly, the slope of the Runge Kutta 4 method is shallower than any of the other lower-order methods. This implies that the rk4 method gets a much better return on its work. It's able to reach much lower error for much less additional work compared to all the other methods.

Question 2

Consider the tridiagonal matrix T_N

$$T_N = \begin{bmatrix} -2 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -2 \end{bmatrix}$$

Using the MATLAB eig() function, the following eigenvalues for the 10 by 10 tridiagonal matrix T_{10} were computed:

$$\lambda_1 = -0.0810$$
$$\lambda_2 = -0.3175$$
$$\lambda_3 = -0.6903$$
$$\lambda_4 = -1.1692$$
$$\lambda_5 = -1.7154$$
$$\lambda_6 = -2.2846$$
$$\lambda_7 = -2.8308$$
$$\lambda_8 = -3.3097$$
$$\lambda_9 = -3.6825$$
$$\lambda_{10} = -3.9190$$

For $N = 10$ t was confirmed that:

$$\lambda_i = -2(1 - \cos(\pi i/(N+1)), \quad i = 1, \ldots, N \tag{1}$$

i	λ_i	Equation
1	-0.0810	$-2(1-\cos(\pi/11))$
2	-0.3175	$-2(1-\cos(2\pi/11))$
3	-0.6903	$-2(1-\cos(3\pi/11))$
4	-1.1692	$-2(1-\cos(4\pi/11))$
5	-1.7154	$-2(1-\cos(5\pi/11))$
6	-2.2846	$-2(1-\cos(6\pi/11))$
7	-2.8308	$-2(1-\cos(7\pi/11))$
8	-3.3097	$-2(1-\cos(8\pi/11))$
9	-3.6825	$-2(1-\cos(9\pi/11))$
10	-3.9190	$-2(1-\cos(10\pi/11))$

Table 3: Eigenvalues calculated using the given equation for $i = 1:10$

Next for $N = 1, \ldots, 20$, the maximum absolute value of T_N was computed and plotted.

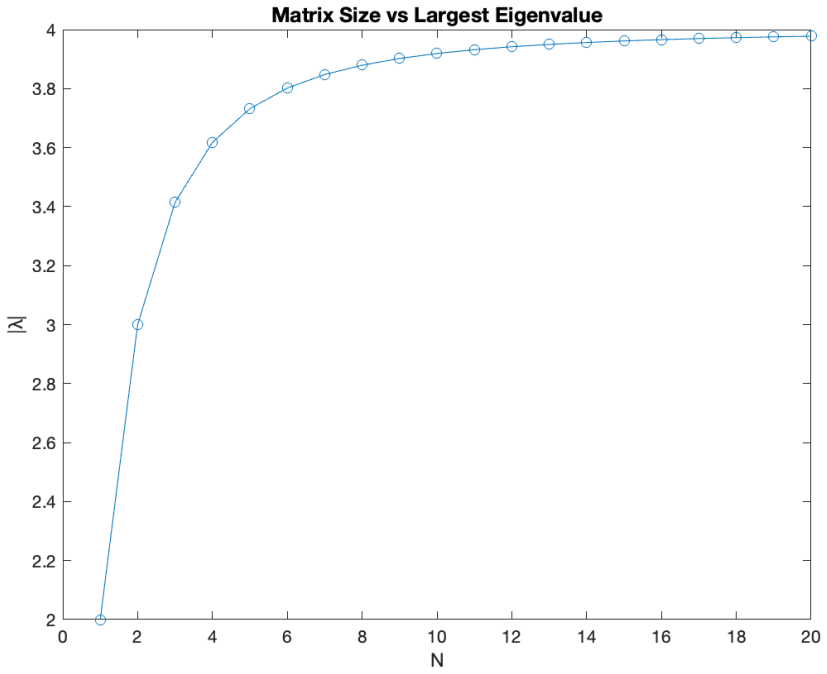

Figure 11: Enter Caption

Visually, as N increases, $|\lambda|$ asymptotically approaches 4. This makes sense as $\cos$ ranges from -1 to 1 and when $\cos(\pi i/(N+1)) = -1$, Equation 1 is maximized:

$$|\lambda| = |-2(1--1)| = 4$$

Further, $\cos(\pi i/(N+1))$ approaches -1 as $\pi i/(N+1)$ approaches π. If we let $i = N$ then we have

$$\lim_{N\to\infty} \frac{\pi N}{N+1} = \pi \tag{2}$$

And it follows,

$$\lim_{N\to\infty} \lambda_N = \lim_{N\to\infty} -2(1-\cos(\pi N/(N+1)) = 4 \tag{3}$$

This explains the behavior of the plot. That as the N increases the term within the cosine can get closer to π. As a result, the entire function approaches 4.