TRAVELER'S GUIDE TO SPECTRAL DEFERRED CORRECTION

BENJAMIN ARMSTRONG

Abstract. Numerical methods for approximating a solution to ODEs and PDEs are of great importance to numerical analysis. Euler methods, while intuitive, are susceptible to large amounts of error, and lack the precision that is sometimes required of a particular problem. Deferred correction algorithms seek to enhance the precision of approximations while also improving their rate of convergence with respect to the time step used. This paper aims to explain the foundations of and evaluate the performance of one such deferred correction algorithm, Spectral Deferred Correction (SDC). Analysis of the algorithm's performance suggests that applying SDC several times greatly improves the precision of an approximation, resulting in lower error than was achieved by Euler methods alone given the same time frame. While SDC is general enough to work with both ODEs and PDEs, this paper only evaluates its performance on a few simple ODEs. Further research could help solidify these findings for a wider range of differential equations.

Contents

1. The Problem

Definition 1.1. An Initial Value Problem (IVP) consists of a set of initial conditions (i.e. the temperature along a rod or the position of a particle) along with either an ODE or a PDE describing how the system will change over time. While not all IVPs involve changes occurring over time, we will use time as it is an intuitive way of thinking about differential equations. Solving an IVP generally amounts to finding (or in our case, approximating) some unknown function. As with any function, we can describe this unknown one as mapping a set of one or more inputs to a set of one or more outputs.

Example 1.2. The first group, the measured or output variables, are the ones we are interested in tracking. For instance, in the heat equation [4] this would be temperature, or in other cases such as the Lorenz system (see (2.2)) these could be two or three Cartesian coordinates representing position of a particle.

The second group are the variables which impact our measured variable, our input variables. In some cases, such as the heat equation, positions on a rod - in addition to time - may influence our measured variable, whereas in others, such as the aforementioned Lorenz system, each time corresponds to only a single measured value so no other variables are needed.

2. Euler Methods

Euler methods, stemming directly from the definition of derivatives in 1-d, are the most natural way to advance initial conditions through time. To do this, we must have a function which sends our set of input variables to the partial derivative of one or more measured variables. In most cases, we can simply arrange our differential equation(s) to isolate any partial derivatives with respect to time.

Example 2.1. For instance, the Lorenz system, which models a particle moving through 3-d space, can be written

$$\begin{aligned} \frac{\partial x}{\partial t} &= \sigma(y-x) \\ \frac{\partial y}{\partial t} &= x(\rho - z) - y \\ \frac{\partial z}{\partial t} &= xy - \beta z \end{aligned} \tag{2.2}$$

where the constants $\sigma, \rho, \beta \in \mathbb{R}$.

Note that in this case, we could have written these as total derivatives since time is the only input variable, but partial derivatives were used for consistency. Also note that re-arranging may not be possible in second-order or higher differential equations, but we will not consider these cases here. And so each of our measured variables has a corresponding right hand side (RHS) computed from input variables, output variables, and in the case of PDEs, from partial derivatives involving input/output variables as well.

Definition 2.3. Some function ϕ is the solution to our IVP if and only if it satisfies the Picard equation

$$\phi(t) = \phi_a + \int_a^t F(s, \phi(s))ds \tag{2.4}$$

exactly, where F is the aforementioned RHS of our ODE. Because the Picard equation defines $\phi(t)$ implicitly, we cannot solve for $\phi(t)$ using analytic methods, and so we must "discretize" in time in order to estimate the integral. Euler schemes are a class of methods of estimating the $\phi(t)$ using the Picard equation by iterating forwards in time.

Definition 2.5. Explicit Euler Method. If we let $u : \mathbb{R}^m \to \mathbb{R}^n$ be our unknown function, then we know from the definition of partial derivatives that

$$F\Big((x,t), u(x,t)\Big) = \frac{\partial u}{\partial t}(x,t) \approx \frac{u(x,t+h) - u(x,t)}{h}$$

as $|h| \to 0$, where F is the RHS of our ODE, $x \in \mathbb{R}^{m-1}$, and $t, h \in \mathbb{R}$.

Rearranging this gives

$$u(x,t+h) \quad \approx \quad u(x,t) \quad + \quad h \cdot F\Big((x,t),u(x,t)\Big), \tag{2.6}$$

which allows us to approximate the value of u after a small time-step forward of h. Now take initial value $u(x,t_0) \in \mathbb{R}^n$ at some $x \in \mathbb{R}^{m-1}$. We can continue stepping forward in this way by small time-steps until we reach $u(x,t) \in \mathbb{R}^n$ for any desired $t > t_0$. Note that the same could be done for $t < t_0$, but we will assume without loss of generality that we are always stepping forward.

Definition 2.7. Implicit Euler Method. Consider defining the partial derivative from Definition 2.5 by

$$\frac{\partial u}{\partial t}(x,t) \quad \approx \quad \frac{u(x,t) - u(x,t-h)}{h}.$$

Then this will rearrange to

$$u(x,t) \quad \approx \quad u(x,t-h) \quad + \quad h \cdot F\Big((x,t),u(x,t)\Big).$$

Now substituting $t+h$ in for t, we get

$$u(x,t+h) \quad \approx \quad u(x,t) \quad + \quad h \cdot F\Big((x,t+h),u(x,t+h)\Big). \tag{2.8}$$

This time, the value at the next time-step (at $t+h$) is implicitly defined in terms of the derivative at $t+h$, instead of the derivative at t. This means that we either need to solve through algebraic manipulation – in the cases where F is linear or otherwise convenient - or use a more sophisticated method such as Newton-Raphson method, as detailed in [2].

3. Finite-Difference Methods

In the previous section, we discussed Euler methods, and showed how two such methods, explicit and implicit Euler, could be used to solve our IVP. In the case of ODEs, such as (2.2), the RHS can be computed from only the output variables at the previous time-step along with a set of constants. However, in the case of PDEs, calculating the RHS is more involved. Take, for instance, the heat equation in 1-d.

Example 3.1. We can model the temperature along some essentially 1-dimensional object (such as a thin rod) with the following equation

$$\frac{\partial u}{\partial t} \quad = \quad \alpha \frac{\partial^2 u}{\partial x^2} \tag{3.2}$$

where constant $\alpha \in \mathbb{R}$ is the thermal diffusivity of the medium, and $u : K \times I \to \mathbb{R}$ sends a point on the rod and a particular time to a temperature at that point and

time. Note that we can choose which space interval $K \subset \mathbb{R}$ and time interval $I \subset \mathbb{R}$ to work with.

In order to solve for the RHS of (3.2), we need a method for estimating partial first- and second-derivatives (in this case, space derivatives) given a set of function values (along the rod) at any particular time.

Definition 3.3. Center-Difference Method. Consider the two equations, derived from Taylor's Theorem,

$$u(x+h) \approx u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u'''(x) \tag{3.4}$$

$$u(x-h) \approx u(x) - hu'(x) + \frac{h^2}{2}u''(x) - \frac{h^3}{6}u'''(x) \tag{3.5}$$

where $u' = \frac{\partial u}{\partial x}$, $u'' = \frac{\partial^2 u}{\partial x^2}$, and so on.

Now by adding (3.4) and (3.5) together, we get an estimate for the second partial derivative of u with respect to x,

$$\begin{aligned} u(x+h) + u(x-h) &\approx 2u(x) + h^2u''(x) \\ \implies u''(x) &\approx \frac{u(x+h) + u(x-h) - 2u(x)}{h^2}. \end{aligned} \tag{3.6}$$

If we instead subtract (3.5) from (3.4), we get an estimate for the first partial derivative of u with respect to x,

$$\begin{aligned} u(x+h) - u(x-h) &\approx 2hu'(x) \\ \implies u'(x) &\approx \frac{u(x+h) - u(x-h)}{2h}. \end{aligned} \tag{3.7}$$

4. Deferred Correction Algorithms

Sections 2 and 3 outlined simple methods for approximating solutions to our IVP, but these methods alone can be prone to large amounts of error, especially in systems where values may "blow up" as one moves forward through time. Spectral deferred correction (SDC) belongs to a larger category of so-called deferred correction algorithms, which seek to iteratively improve the accuracy of an IVP approximation by estimating its errors from the system's actual behavior. We will summarize the method here. Note that much of the description that follows is directly compiled and paraphrased from [1].

Example 4.1. In deferred correction algorithms, the goal is to look at how slight inaccuracies in the derivative function at each point in time compound on each other as time moves forward, which can result in larger errors over time.

Consider an approximation $\phi : \mathbb{R}^m \to \mathbb{R}^n$ to the unknown function $u : \mathbb{R}^m \to \mathbb{R}^n$, and use F to denote the RHS, which gives $\frac{du}{dt}$. Our approximation takes the form of a set of values $\phi(t_0), \cdots, \phi(t_k) \in \mathbb{R}^n$ at certain time points $t_0, \cdots, t_k \in \mathbb{R}^m$ spread across an interval. This set of times is called the temporal discretization of our

algorithm, and can be either equispaced or follow some other quadrature scheme [5]. Now suppose that we have some error at the i^{th} time point

$$\epsilon \quad = \quad u(t_i) \quad - \quad \phi(t_i)$$

then the derivative that we initially used to step our approximation forward in time was inaccurate by

$$G \quad = \quad F\Big(t_i, \phi(t_i) + \epsilon\Big) \quad - \quad F\Big(t_i, \phi(t_i)\Big),$$

which means that $\phi(t_{i+1})$ will not be estimated correctly using an Euler method. This, in turn, will affect the derivative at t_{i+1}, in addition to any inaccuracies that were already there.

Definition 4.2. Classical Deferred Correction. For any sort of deferred correction to work, there must be some source of error, since if the error function has an initial value of zero there is nothing to compound on as described in Example 4.1. For Classical Deferred Correction algorithms, the error comes from inaccuracies in the derivative used with our Euler method of choice. In particular, we have taken the RHS derivative at each time discretization point, but this does not take into account the function's behavior directly before or after that point.

We can interpolate the points of our approximation with a polynomial, since polynomials all have known analytic derivatives. Then we can consider the compounding effect that this refined derivative would have on our approximation to get an estimate of the approximation's error.

More formally, suppose we have an approximation $\phi(t_0), \cdots, \phi(t_k)$. We know that there is exactly one interpolating polynomial of degree $(k-1)$ that exactly fits k points, called the Lagrange Interpolating Polynomial [6], which we denote $L_\phi(t)$.

Since we have interpolated ϕ, we have not changed the value of our approximation function ϕ at the time points, but that does not mean that the derivative remains unchanged. At any time point t_i,

$$F\Big(t_i, L(t_i)\Big) \quad - \quad \frac{dL_\phi}{dt}(t_i)$$

is likely to be nonzero. As suggested earlier, this is because Euler methods operate on the assumption that the function is nearly linear at small enough time-scale, but by interpolating the function we may find a different, more accurate derivative.

Now consider the error function $\delta : \mathbb{R}^m \to \mathbb{R}^n$ given by

$$\delta(t) \quad = \quad u(t) \quad - \quad L_\phi(t). \tag{4.3}$$

Since differentiation is linear, we know that

$$\frac{d}{dt}\delta(t) \quad = \quad \frac{d}{dt}u(t) \quad - \quad \frac{d}{dt}L_\phi(t)$$

which can be rewritten as

$$\delta'(t) \quad = \quad F\Big(t, u(t)\Big) \quad - \quad \frac{dL_\phi}{dt}(t).$$

In order to get rid of the $u(t)$ term, which is unknown, we can substitute using (4.3) to get

$$\delta'(t) \quad = \quad F\Big(t, L_\phi(t) + \delta(t)\Big) \quad - \quad \frac{dL_\phi}{dt}(t). \tag{4.4}$$

We were able to get rid of the unknown, but now δ' is implicitly defined! However, we now have all the elements needed to constitute a solvable IVP: an unknown function δ, its RHS derivative δ', and an initial value of $\delta(t_0) = 0$. Note that the initial error must be zero since our approximation function started out matching the value of $u(t_0)$ exactly. More concisely,

$$L_\phi(t_0) \quad = \quad \phi(t_0) \quad = \quad u(t_0).$$

So we can compute an approximation for δ at the time points $t_0, \cdots, t_k$ using the same Euler method that we used to compute an approximation for u.

Finally, we can update our previous approximation function according to the error that we estimated, such that

$$\phi_{new}(t) \quad = \quad \phi_{old}(t) \quad + \quad \delta(t).$$

Definition 4.5. Spectral Deferred Correction. Following a similar approach, the main distinction that sets Spectral Deferred Correction (SDC) apart is the source of the error. Rather than construct an interpolating polynomial to estimate error, we will instead perform numerical integration of the RHS function, now that we have an approximation at all of our time points. In particular, we seek to estimate a so-called residual function

$$\epsilon(t) \quad = \quad \left[\phi(t_0) + \int_{t_0}^{t} F(s, \phi(s))ds\right] - \phi(t) \tag{4.6}$$

This time, we must change the domain of the error function from (4.3). Let $\delta : \{t_0, \cdots, t_k\} \to \mathbb{R}^n$ given by

$$\delta(t) \quad = \quad u(t) \quad - \quad \phi(t).$$

It is fine that δ is not defined except for at our time discretization points, since those are the only points we seek to estimate the error at.

Now in order to estimate how much the errors added by ϵ will compound over time, define function $G_\phi : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$ given by

$$G_\phi(t, \delta) = F\Big(t, \phi(t) + \delta(t)\Big) - F\Big(t, \phi(t)\Big). \tag{4.7}$$

Now we can step the error forward in time with the same Euler method we used for our initial approximation, in much the same way that we did in Definition 4.2. The only difference is that this time we must add on any residual gained over that time step. This gives us slightly different explicit and implicit Euler functions:

Explicit

$$\delta(t_{i+1}) = \delta(t_i) + (t_{i+1} - t_i) \cdot G(t_i, \delta_i) + \Big(\epsilon(t_{i+1} - \epsilon(t_i)\Big) \tag{4.8}$$

Implicit

$$\delta(t_{i+1}) = \delta(t_i) + (t_{i+1} - t_i) \cdot G(t_{i+1}, \delta_{i+1}) + \Big(\epsilon(t_{i+1} - \epsilon(t_i)\Big) \tag{4.9}$$

We have estimated the error δ, so can update our approximation accordingly:

$$\phi_{new}(t) = \phi_{old}(t) + \delta(t).$$

5. Error Plots and Timings

While Sections 1 to 4 discussed the theoretical foundations for SDC, much of my work during the REU was devoted to designing and testing a computer program, written in Matlab. A link to the program can be found here[1]. The purpose of SDC is to be a quicker, more accurate alternative to using Euler methods alone, but I wanted to put that to the test by implementing the SDC algorithm myself.

We will evaluate the program using two methods: error plots and timing. First, we use two ODEs with closed-form solutions - one non-stiff and one stiff - to create error plots. We will run an Euler method over the interval $[0, 10]$ and measure by how much the numerical value at $t = 10$ differs from the exact analytic value at that point. On the horizontal axis we will vary the number of time discretization points (nt) used, and on the vertical axis we will plot the error at $t = 10$. We will also plot lines for the error after the first few times SDC was run ($j = 1, 2, 3$). For each ODE, we will create plots for both explicit and implicit Euler to compare results between the two.

Remark 5.1. Euler methods are said to have linear convergence, since we expect error to be reduced by the same factor that nt is increased by. So multiplying nt by 6 will give one sixth of the error at $t = 10$. However, if we repeatedly apply SDC, we can bring this relationship to be quadratic, cubic, quartic, etc. with respect to nt. For instance, after applying SDC once, an increase of nt by a factor of 6 corresponds to approximately $\frac{1}{36}$ of the error, and if SDC was applied two more times, this same increase of nt would result in approximately $\frac{1}{1296}$ of the error. So SDC can not only improve estimates for each value of nt, but it can also make increases of nt exponentially more effective.

[1] https://github.com/bdarmstrong/SDC

Example 5.2. Exponential ODE. First, we consider the simple ODE

$$\frac{dy}{dt} = y, \qquad y0 = C$$

produces the analytic solution

$$y = C \cdot e^t.$$

If we look at initial conditions $C = 1$ and time interval $[0, 10]$, we can measure the error at t = 10, as suggested and plot them, as seen in Figure 1 and Figure 2.

Observation 5.3. For explicit Euler, shown in Figure 1, the error without applying any SDC (when $j = 0$) was decreased by a factor of around 7 when the number of temporal points nt was multiplied by 10. So explicit Euler performed a litle worse than the factor of 10 that was expected based on Remark 5.1. Similarly, for $j = 1$, $j = 2$, and $j = 3$ the error decreased by a factor of around 60, 500, and 4500 respectively. While this under-performed from the 10^2, 10^3 and 10^4 that we would expect, the order of magnitude still matches theoretical predictions.

For Implicit Euler, shown in Figure 2, the error with no SDC decreased by a factor of 15 when nt was multiplied by 10. For $j = 1$, $j = 2$, and $j = 3$ the error decreased by a factor of around 200, 2400, and 28000 respectively. We can see that implicit Euler significantly over-performed the theoretical predictions in terms of rate of convergence, and further, this over-performance becomes more pronounced as j increases, going from 1.5 times the expected at $j = 0$ to 2.8 times the expected at $j = 3$.

At $nt = 100$, explicit Euler had half as much error at $j = 0$ and up to 10 times less error for $j = 1, 2, 3$ when compared to Implicit Euler. However, at $nt = 1000$, these errors had nearly evened out, and were about the same between both Euler methods for $j = 0, 1, 2, 3$. This indicates that although explicit Euler performs better at small values of nt, it has a slower rate of convergence as nt increases than implicit Euler, particularly after SDC has been applied more times (larger j).

Example 5.4. Trigonometric ODE. To test the impact of using implicit methods, we should use a stiff ODE with closed-form solution. We will use the following set of equations, pulled from [3]:

$$\frac{dx}{dt} = -16x + 12y + 16\cos(t) - 13\sin(t)$$

$$\frac{dy}{dt} = 12x - 9y - 11\cos(t) + 9\sin(t)$$

These, together with the initial condition $(x_0, y_0) = (1, 0)$, produce the solution

$$x(t) = \cos(t)$$

$$y(t) = \sin(t)$$

So we expect the values at t = 10 to be

$$(\cos(10), \sin(10)) \approx (-0.839, -0.544)$$

We will plot only the error in x at this point, since we would not expect error in x and y to differ by much. Results using explicit and implicit Euler are shown in Figure 3 and Figure 4.

Observation 5.5. First, we can look at explicit Euler. Whereas the exponential ODE under-performed with explicit Euler, as described in Observation 5.3, the trigonometric ODE worked about as expected. For $j = 0, 1, 2$, error decreased by factors of 10, 100, and 990 when nt was multiplied by 10. However, for $j = 3$, the convergence was significantly worse, only decreasing by a factor of 2400. This could indicate that our explicit method starts to perform worse at higher values of j.

For implicit Euler, the trigonometric ODE had error which decreased by a factor of around 10, 100, 1000, and 9950. This matches up nearly exactly with what we would expect based on Remark 5.1. Implicit Euler appears to more reliably converge at the rates expected than explicit Euler did for the trigonometric ODE, and so for higher values of j and higher values of nt, implicit Euler would be preferable.

Timings

Next, we will evaluate the cost/benefit of using SDC as opposed to simply applying an Euler method with a greater value of nt. Holding time constant, we will see which method results in a lower error, and repeat this process for $j = 0, 1, 2, 3$, as we did in the error graphs discussed above.

Observation 5.6. First, note that the plots are no longer log-log plots. The y-axis is still logarithmic, but the x-axis is now linearly spaced between 0.1 and 1. And so this time each line appears as more of an arc rather than linear, with error expectedly decreasing as the program is allowed more time to run (and in effect, allowed to use larger values of nt).

There are two processes which take most of the program's time: the overhead of creating an integration matrix and running Euler methods. Creating an integration matrix is an $O(nt^2)$ operation, so it can be very time consuming. While a 1000 by 1000 matrix may only take 1 second to create, a 10000 by 10000 matrix is expected to take 1 minute 40 seconds. This can very quickly become prohibitive - while values of nt in the millions could take less than a second at $j = 0$, the nt was restricted to less than 10000 for $j = 1, 2, 3$.

In order to avoid the long creation time for the integration matrix, I pre-loaded matrices for all multiples of 100, up to $nt = 10000$. This means that the only significant factor contributing to the program's runtime is the number of times that an Euler method is run. Each time that SDC is applied, our chosen Euler method must be run to propagate the error forward in time, as detailed in Definition 4.2 and Definition 4.5, and in addition, Euler must be run to make our initial approximation.

As seen in the two figures for the exponential ODE, Figure 5 and Figure 6, running SDC once was actually more time-consuming than it was worth, with the red $j = 1$ line having consistently greater error than the $j = 0$ line. However, the benefits of running SDC several times begin to be seen, with $j = 2$ having slightly lower error than $j = 0$ and $j = 3$ having much lower error.

The trigonometric ODE had different results, shown in Figure 7 and Figure 8. For explicit Euler, running SDC once did result in a lower error than not running it at all, with greater values of j resulting in lower errors at all time limits. For implicit Euler, however, the lines for $j = 1, 2, 3$ are blank at the lower time limits. This indicates that the approximation blew up to infinity when only allowed smaller values of nt, but when the time limit was raised, the blow-up could be avoided. When they did not blow up, larger values of j did indeed result in quicker convergence, but the possibility of a blow-up is something to be mindful of.

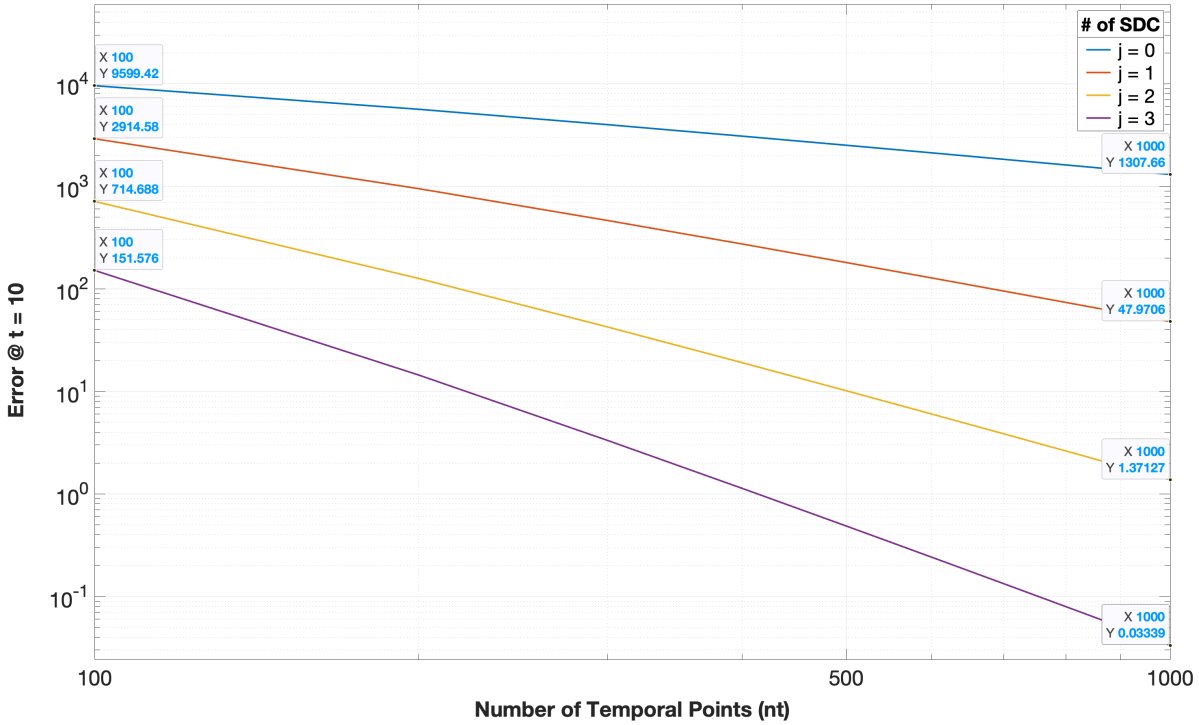

FIGURE 1. Exponential ODE, Explicit Euler

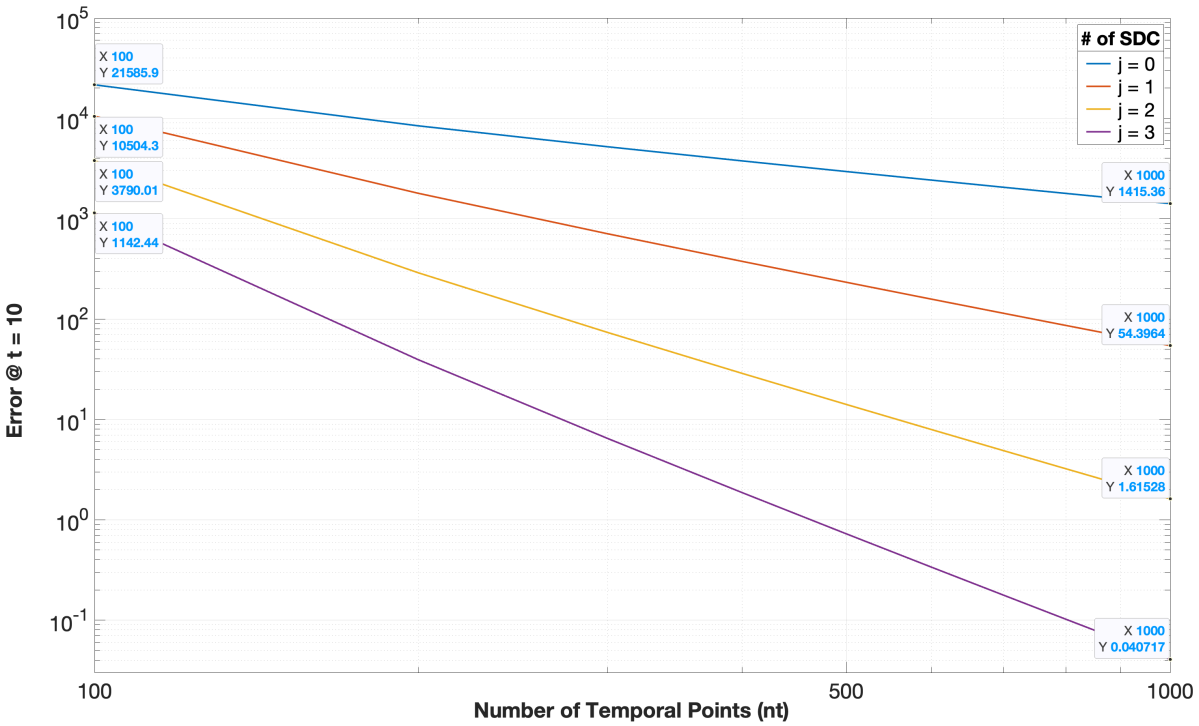

FIGURE 2. Exponential ODE, Implicit Euler

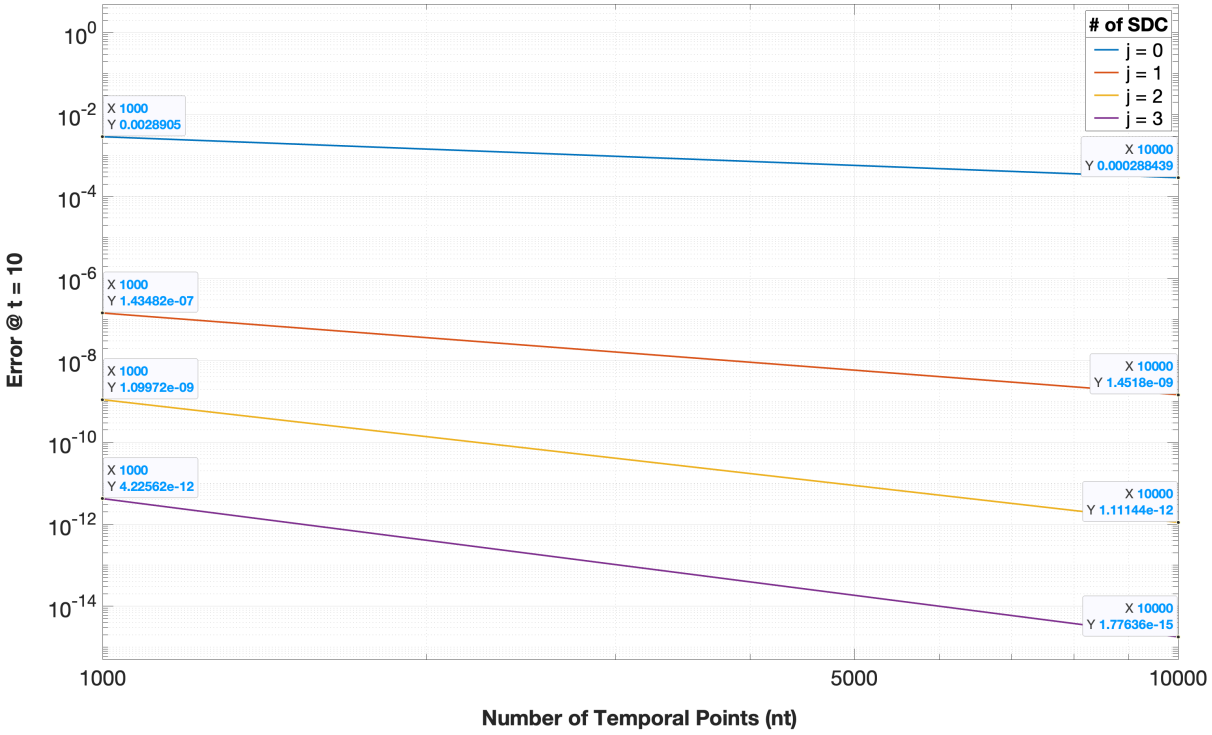

FIGURE 3. Trig ODE, Explicit Euler

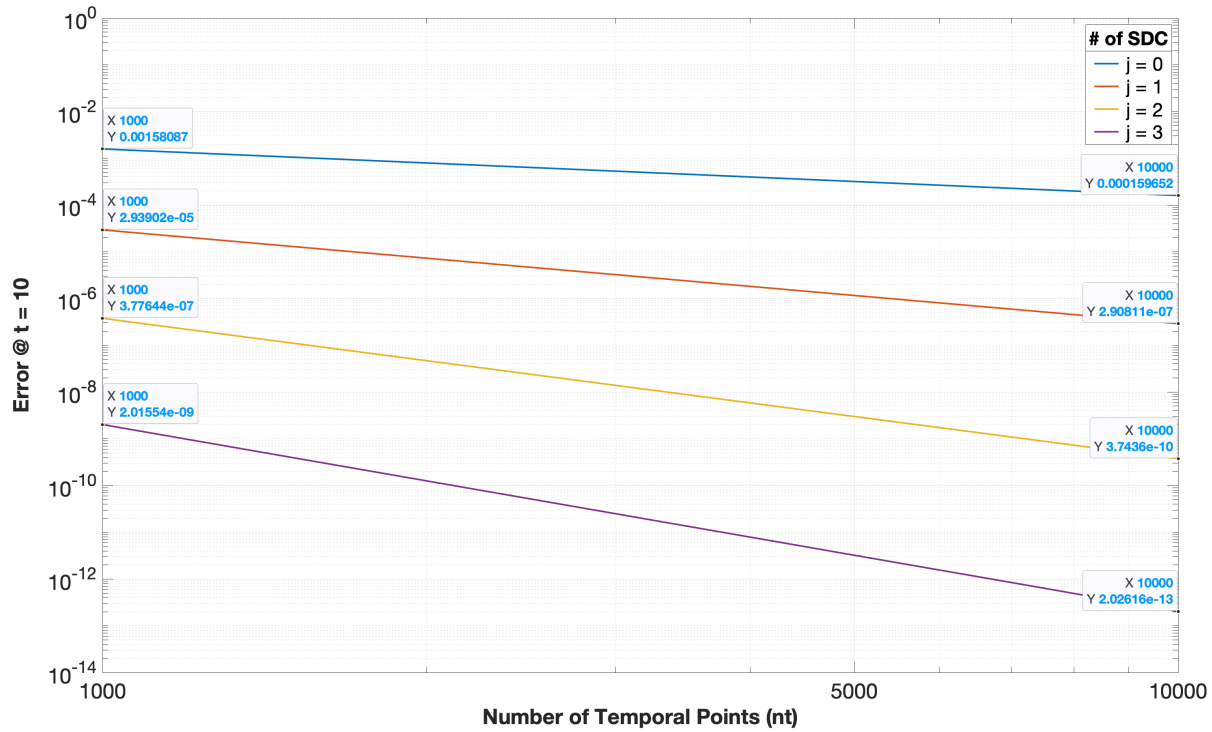

FIGURE 4. Trig ODE, Implicit Euler

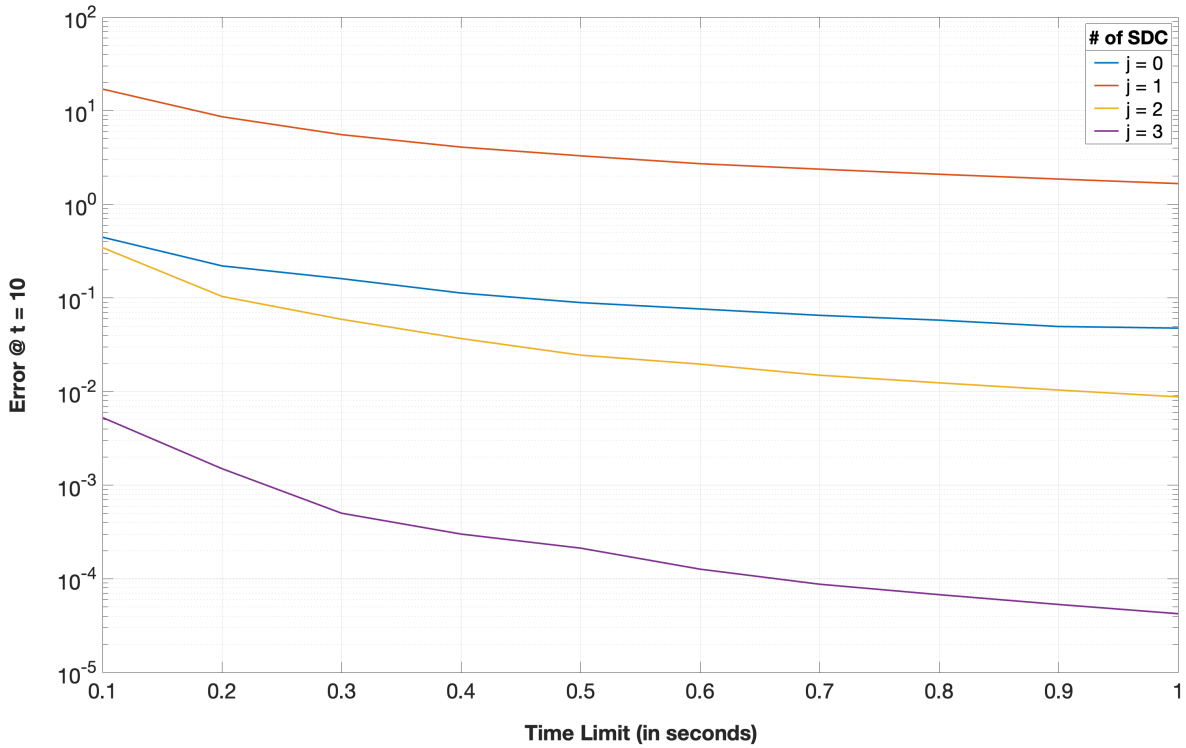

FIGURE 5. Exponential ODE, Explicit Euler

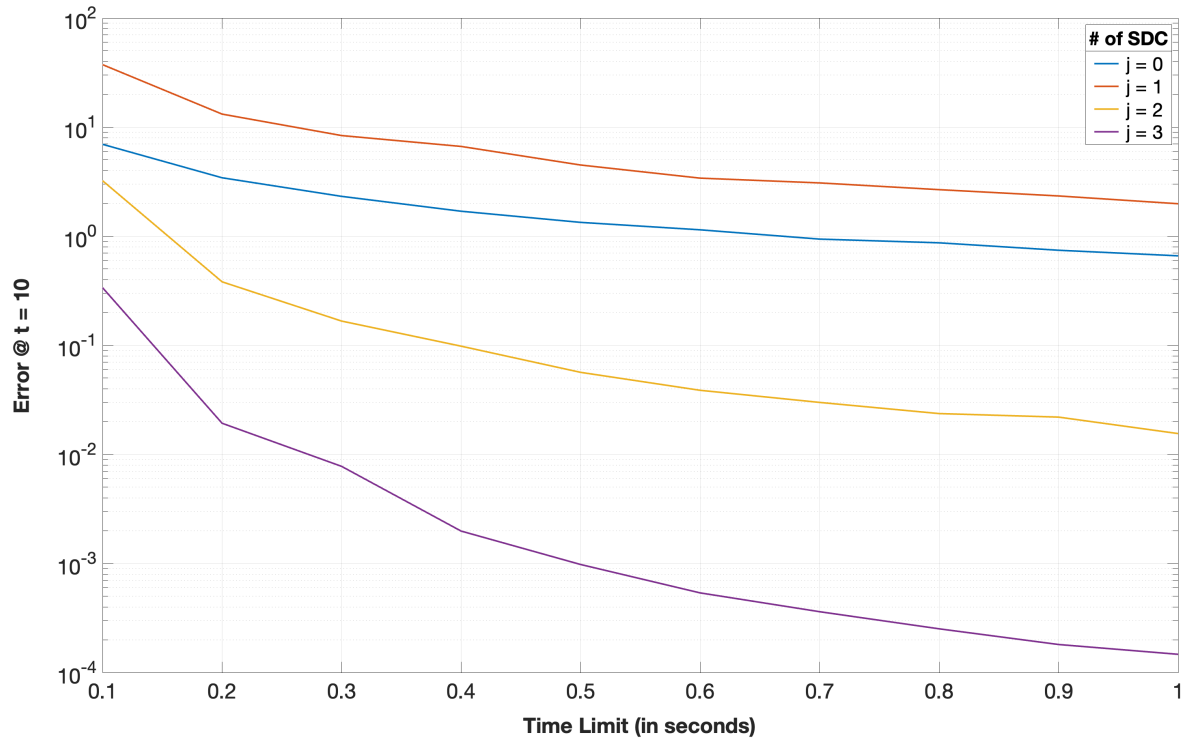

FIGURE 6. Exponential ODE, Implicit Euler

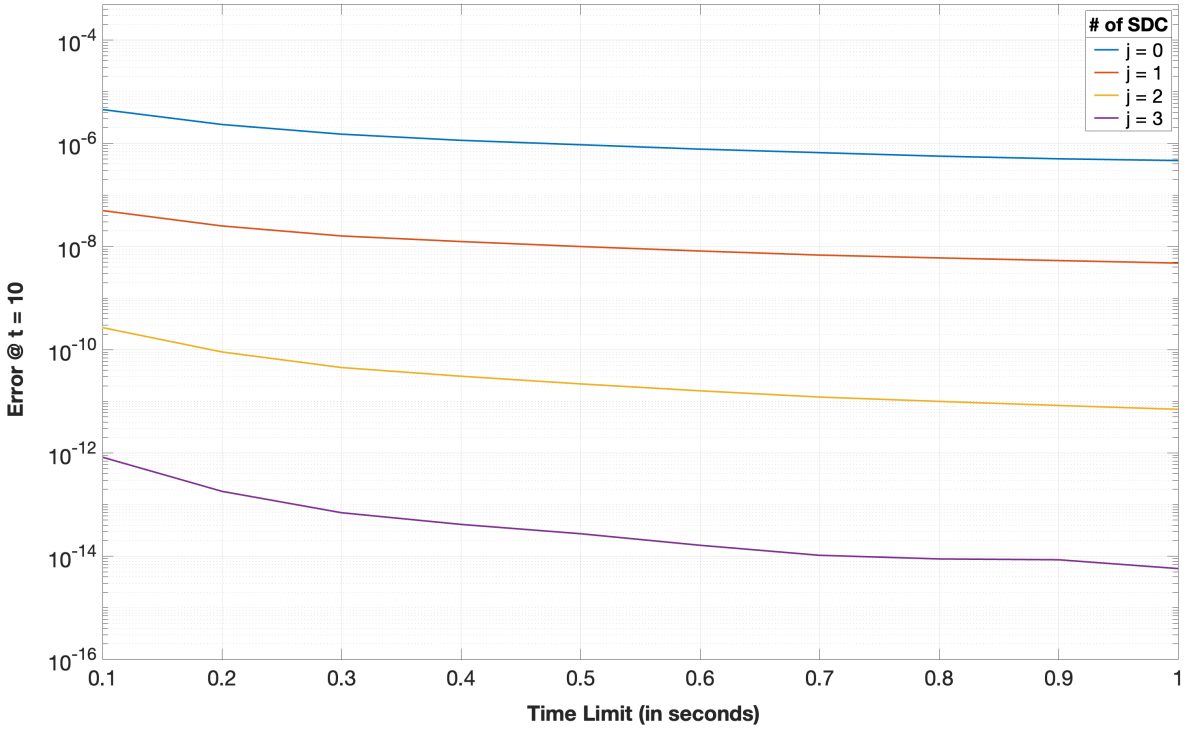

FIGURE 7. Trig ODE, Explicit Euler

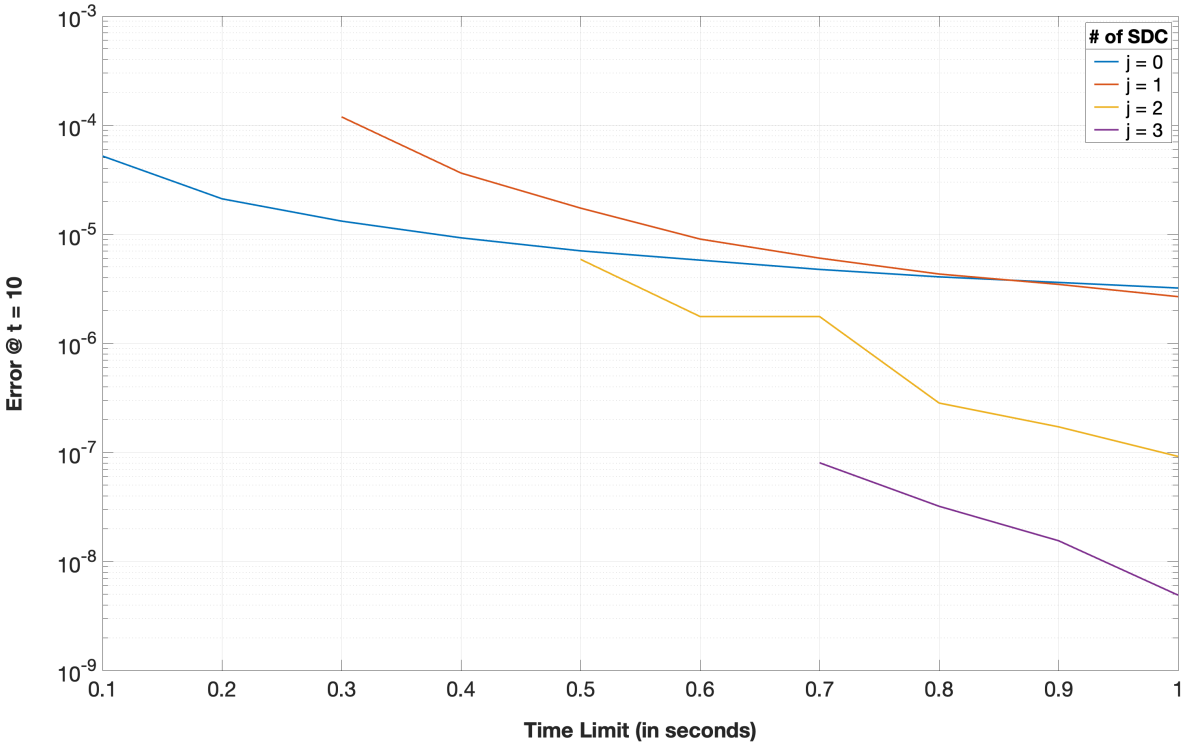

FIGURE 8. Trig ODE, Implicit Euler

6. Acknowledgments

Going into the summer, I knew that I wanted to work in applied mathematics, but I had no idea what particular project I wanted, or what sort of research was out there. That is where Kingsley stepped in. He set me up with a project, gave me measurable goals to work towards, and gave me confidence with the foundations of numerical analysis, which I knew next-to-nothing about beforehand. This summer was a mix of learning theory and working on my own implementation of the theory, and Kingsley helped me along the whole way. I know I would have been lost this summer without someone to give me guidance and keep me on track when I needed it, and for that I am very thankful for my REU mentor, Zhen Wei (Kingsley) Yeon.

References

[1] Alok Dutt, Leslie Greengard, and Vladimir Rokhlin. Spectral Deferred Correction Methods for Ordinary Differential Equations.
[2] Segment from unknown textbook, https://www.mathworks.com/matlabcentral/answers/1716860-using-implicit-euler-method-with-newton-raphson-method.
[3] J. C. Butcher. Numerical Methods for Ordinary Differential Equations. Wiley. 2016.
[4] https://en.wikipedia.org/wiki/Heat_equation
[5] https://en.wikipedia.org/wiki/Numerical_integration
[6] https://en.wikipedia.org/wiki/Lagrange_polynomial
[7] https://en.wikipedia.org/wiki/Stiff_equation