Numerical Solutions to Differential Equations

Alex Pizzuto

May 2, 2017

1 Introduction

It is often the case that many problems in mathematics are beyond the scope of analytic solutions. Additionally, some times methods exist to solving certain problems, but they are often unwieldy and there are times when close approximations can stand in for exact analytical solutions. Although there are myriad remedies to these problems, the most ubiquitous in the mathematics community is the implementation of numerical methods.

Throughout this paper, we explore three main regimes of numerical applications and approximations. In section 2 we will explore fixed point problem and finding roots, certain types of fixed point iteration used to extract properties of functions, such as the the root of a given function. Although extremely useful, possibly an even more accessible application of numerical techniques to the beginning mathematics student is the use of numerical techniques to approximate integrals. While exposure to these techniques is often a form of motivation for integration in general, there are many functions that cannot be integrated analytically, and thus the best attempt at understanding the behavior of these integrals is through the use of numerical integration. We will motivate multiple different algorithms for approximating integrals throughout section 3 and highlight some relative advantages and disadvantages of each technique. Finally, we will explore numerical methods to solving differential equations in section 4. Again, these methods prove useful for illuminating the nature of solutions to complex and intractable differential equations.

2 Fixed Point Problems and Finding Roots

In numerical analysis, we often wish to develop an algorithm that will converge to a desired point. Fixed-point iteration is a way of computing fixed points of iterative functions. In other words, suppose we have a function, $f(x)$, and we pick a point in the domain of this function, x_0. Then, we can generate a sequence of points, x_n, where $x_n = f(x_{n-1})$ where $n \in \mathbb{Z}^+$. It is often the case that certain manipulations of these iterations can provide us with certain points that describe the function, such as where the function takes a value of zero. We explore some of these methods in the following sections.

2.1 Heron's Algorithm

Before the age of calculators, the method for finding square roots relied on algorithms to find approximations for the square roots (especially in the case of the square roots of prime numbers, all of whose square roots are irrational). One of the first and most efficient algorithms for approximating the square root of a number $z \in \mathbb{R}^+$ was developed by Heron, best known for his formula for the area of a triangle given the lengths of the sides. This method calls for one to make a guess as to what the desired root is, and then compute the average of the guess and the quotient of z with the guess. This computation produces the next term in the sequence of approximations, and if it is in agreement with the previous term, then the desired root has been found.

In other words, Heron's Algorithm can be stated as follows. Given a $z \in \mathbb{R}^+$, choose a x_0 as a guess for $\sqrt{z}$. Then, compute

$$x_{n+1} = \frac{1}{2}\big(x_n + \frac{z}{x_n}\big)\,.$$

Continue iterating the previous step, and the sequence of the x_n will converge to $\sqrt{z}$. Repeat this step until the desired accuracy for the approximation is obtained.

Although it seems that this intuitive algorithm will produce the correct root, a rigorous proof is necessary to be sure that this algorithm holds for all $z \in \mathbb{R}^+$. In order to approach this, however, we reduce our domain by the use of scaling. For instance, suppose that $0 < z < \frac{1}{2}$. Then, there exists a $k \in \mathbb{Z}^+$ such that $4^k z = \tilde{z}$ is between $\frac{1}{2}$ and 2. Now, suppose that $2 < z$. Then there exists a $k \in \mathbb{Z}^-$ such that $4^k z = \tilde{z}$ is between $\frac{1}{2}$ and 2. It is worth noting that this k is unique as division or multiplication by 4 of a number falling in between $\frac{1}{2}$ and 2 will no longer be in this range. Thus, if Heron's Algorithm is valid for all $\tilde{z}$ such that $\frac{1}{2} < \tilde{z} < 2$, then

$$\sqrt{z} = \sqrt{4^{-k}\tilde{z}} = 2^{-k}\sqrt{\tilde{z}}$$

and we need only find the k and $\sqrt{\tilde{z}}$ to extend Heron's Algorithm to be valid for all $z \in \mathbb{R}^+$.

However, it still remains to be shown that Heron's Algorithm will converge to z for all z in the range $[\frac{1}{2}, 2]$.

Theorem 1 *Without loss of generality, assume that $\frac{1}{2} < z < 2$. We define $e_n = x_n - \sqrt{z}$. Then $e_{n+1} = \frac{1}{2}e_n^2(\frac{1}{x_n})$.*

Proof:

$$\begin{aligned}
e_{n+1} = x_{n+1} - \sqrt{z} &= \frac{1}{2}\Big(x_n + \frac{z}{x_n}\Big) - \frac{1}{2}\Big(\sqrt{z} + \frac{z}{\sqrt{z}}\Big) \\
&= \frac{1}{2}\Big(x_n\sqrt{z} - \Big(\frac{z}{\sqrt{z}} - \frac{z}{x_n}\Big)\Big) \\
&= \frac{1}{2}\Big(e_n - z\Big(\frac{1}{\sqrt{z}} - \frac{1}{x_n}\Big)\Big) \\
&= \frac{1}{2}\Big(e_n - z\Big(\frac{x_n - \sqrt{z}}{\sqrt{z}x_n}\Big)\Big) \\
&= \frac{1}{2}\Big(e_n - \sqrt{z}\Big(\frac{e_n}{x_n}\Big)\Big)
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2}\Big(e_n\Big(1-\frac{\sqrt{z}}{x_n}\Big)\Big)\\
&= \frac{1}{2}\Big(e_n(\frac{x_n-\sqrt{z}}{x_n}\Big)\Big)\\
&= \frac{1}{2}e_n^2(\frac{1}{x_n})
\end{aligned}
$$

■

Now, all that remains to be shown is that the error terms are decreasing to 0, and thus the iterates x_n converge to $\sqrt{z}$.

Theorem 2 *Suppose $x_0 = 1$. Then for $n \geq 1$, the following hold:*

1. $e_n \geq 0$
2. $e_n \leq (\frac{1}{8})^{2^{n-1}}$
3. $e_{n+1} \leq e_n^2$

Proof: By Induction
Use the result form Theorem 1 repeatedly. For $n = 1$, $e_1 = \frac{1}{2}e_0^2 \geq 0$. So the first holds for the basis of induction. Also, as $e_0 = x_0 - \sqrt{z} = 1 - \sqrt{z}$, we have that $-\frac{1}{2} < e_0 < \frac{1}{2} \Rightarrow e_1 = \frac{1}{2}e_0^2 \leq \frac{1}{2}(\frac{1}{2})^2 = \frac{1}{8}$ which gives us 2. Finally, as $0 \leq e_1 = x_1 - \sqrt{z}$, then $x_1 \geq \sqrt{z} \geq \sqrt{\frac{1}{2}}$. So $e_2 = \frac{1}{2}e_1^2\frac{1}{x_1} \leq \frac{1}{2}e_1^2\frac{1}{\sqrt{x_2}} = \sqrt{\frac{1}{2}}e_1^2 \leq e_1^2$, so our entire basis of induction holds.
Induction Hypothesis: Suppose that 1, 2, and 3 hold for some $n = k \geq 1$. So $0 \leq e_k = x_k - \sqrt{z} \Rightarrow x_k \geq \sqrt{z} \geq \sqrt{\frac{1}{2}}$. Thus, $e_{k+1} = \frac{1}{2}e_k^2\frac{1}{x_k} \geq 0$ which gives us 1.
As $e_{k+1} \leq e_k^2$ and $e_k \leq \left(\frac{1}{8}\right)^{2^{k-1}}$, by our induction hypothesis,

$$e_{k+1} \leq e_k^2 \leq \left(\left(\frac{1}{8}\right)^{2^{k-1}}\right)^2 = \left(\frac{1}{8}\right)^{2^k};$$

we have that 2 holds. Finally, as $e_{k+1} \geq 0$, we have

$$0 \leq e_{k+1} = x_{k+1} - \sqrt{z} \Rightarrow x_{k+1} \geq \sqrt{z} \geq \sqrt{\frac{1}{2}}$$

and thus

$$e_{k+2} = \frac{1}{2}e_{k+1}^2(\frac{1}{x_{k+1}}) \leq \frac{1}{2}e_{k+1}^2(\frac{1}{\sqrt{\frac{1}{2}}}) = \sqrt{\frac{1}{2}}e_{k+1}^2 \leq e_{k+1}^2$$

Thus, the results hold for all $n \geq 1$ ♦

Working with the assumption that $\frac{1}{2} < z < 2$, and that $x_0 = 1$, then we can bound the limit of Heron's Algorithm, and use the squeeze theorem to show that $0 \leq \lim_{n\to\infty} e_n \leq \lim_{n\to\infty}(\frac{1}{8})^{2^{n-1}} = 0$. Thus, the limit of the error terms approaches zero and the $\lim_{n\to\infty} x_n = \sqrt{z}$, so the sequence converges.

This error formula proves to be of great importance. Not only is the error in the n+1 term bounded by the square of the error in the nth step, assuring us that the sequence of the x_n converges to $\sqrt{z}$, but it also informs us of the rapidity of the convergence. For example, one could compute $\sqrt{z}$ to over 100 places in 8 iterations of Heron's Algorithm.

Example 1 We will take the prime number $z = 3727$. In order to do this, we first find an integer k such that $\tilde{z} = 4^k z$ where $\frac{1}{2} \leq \tilde{z} \leq 2$. In order to scale to have a number between $\frac{1}{2}$ and 2, we use the following code in *Mathematica*.

```
ClearAll[x];
z=3727; zbar=z; While[zbar > 2, zbar = zbar/4];
k = Log[4, zbar/z];
```

Making the uneducated guess of the original number being its own root, we obtain the following for the sequence of approximations with their respective relative error terms:

n	x_n	$\frac{e_n - \sqrt{z}}{\sqrt{z}}$
0	3727.0000000000000	-60.04916051838878
1	1864.0000000000000	-29.532770380004479
2	932.99973175965665	-14.282761037780777
3	468.49718692566121	-6.674097120214190
4	238.22620539307191	-2.9022028045957346
5	126.93549988800041	-1.0792341583430262
6	78.148434241491692	-0.2800902351139192
7	62.919864421131600	-0.0306425819267299
8	61.076969963974318	-0.0004555254380141
9	61.049166849459762	$-1.037044723 * 10^{-7}$
10	61.049160518389113	$-5.4 * 10^{-15}$
11	61.049160518388785	0
12	61.049160518388785	0
13	61.049160518388785	0
14	61.049160518388785	0
15	61.049160518388785	0

♦

Looking closely at the sequence of error terms in Example 1, we note the validity of the error formula provided in theorem 1. This is most evident in terms 9 and 10, where the relative difference of the 10th term is less than the square root of the relative difference of the 9th. In addition, when changing starting values, as the error formula is recursive, it will dictate the initial magnitude of the error, e_0, and therefore the magnitude of all successive terms. Thus, in order to minimize error throughout the sequence of error terms, a more accurate initial guess is needed. However, the overall behavior of the error sequence is preserved regardless of initial guess.

2.2 Newton's Method

A broader extension of the previously explored Heron's Algorithm is Newton's Method, due to Sir Isaac Newton, and it serves the purpose of finding a solution of $f(x) = 0$. New-

ton's Method, like Heron's Algorithm, is a repeated manipulation that strives to eventually converge to the root of a given function. The process is as follows:

1. Choose an x_0 (the closer this guess is to the root, the more likely the method is to converge quickly).
2. $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
3. Repeat the previous step until desired precision is obtained

This process relies on finding the value of $f(x)$ at x_0 and then finding the tangent line from that point. That tangent line is traced until it intersects the x-axis, and this point of intersection becomes the next term in the sequence. The equation of the tangent line is given by $\frac{y-f(x_n)}{x-x_n} = f'(x_n)$. The point of intersection occurs when this line takes the value of zero. Thus, we have

$$\frac{-f(x_n)}{x-x_n} = f'(x_n) \Rightarrow \frac{-f(x_n)}{f'(x_n)} = x - x_n \Rightarrow x = x_n - \frac{-f(x_n)}{f'(x_n)},$$

which yields the next term in our sequence when we set the discovered x equal to x_{n+1}. This is displayed for one case below: Before giving a rigorous proof of the convergence of

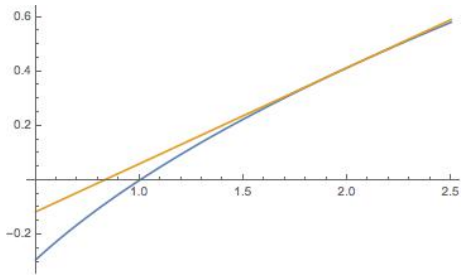

Figure 1: Graphical example of one iteration of Newton's Method from $x_0 = 2$

Newton's Method, we note that Heron's Algorithm is just a specific case of Newton's Method. In Newton's Method, take the function $f(x) = x^2 - z$, where z is a constant. Then, provided with an x_0 or having iterated up to a certain x_n, we have

$$x_{n+1} = x_n - \frac{x_n^2 - z}{2x_n} \Rightarrow x_{n+1} = \frac{x_n^2 - z}{2x_n} \Rightarrow x_{n+1} = \frac{1}{2}(x_n + \frac{z}{x_n}).$$

However, this was the iterative step for Heron's Algorithm. Thus, we note that Heron's Algorithm is just a special case of Newton's Method that will converge to $z^{1/2}$.

Before giving a rigorous proof of the nature of the error of Newton's Method, we explore an example.

Example 2 Suppose we are given the function $f(x) = 7\sin(0.1x)\cos(0.2x) + .01x^2 + 1.85$ with the knowledge that the function has two zeroes lying in the interval [0,20].

These zeroes will be found up to an accuracy of 17 places using Newton's Method. We will run 200 iterations for the starting values of $x_0 = 3, 6, 9, 12, 15, 18, 21$. Using *Mathematica*,

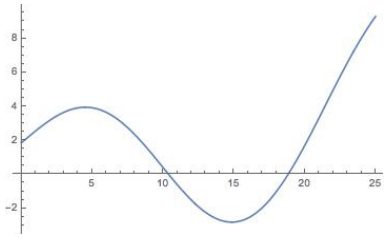

Figure 2: Graph of our given function

we obtain the following:

x_0	n	x_n	x_{n+1}	$f(x_n)$	$f(x_{n+1})$
3	30	10.388965154706545	10.388965154706543	-1×10^{-15}	1×10^{15}
6	8	10.388965154706545	10.388965154706543	-1×10^{-15}	1×10^{15}
9	4	10.388965154706547	10.388965154706547	-7×10^{-16}	-7×10^{-16}
12	4	10.388965154706547	10.388965154706547	-7×10^{-16}	-7×10^{-16}
15	58	10.388965154706547	10.388965154706547	-7×10^{-16}	-7×10^{-16}
18	4	18.836719960868102	18.836719960868106	-3×10^{-15}	4×10^{-15}
21	5	18.836719960868102	18.836719960868106	-3×10^{-15}	4×10^{-15}

We note that all of our provided x_0 values either converge to a zero of the function or oscillate near a given zero. Based on the magnitude of the value that the function takes at the suggested limits, we are led to believe that the function takes zeroes at the values 10.388965154706547 and 18.836719960868104. Our data reveals two of the possible outcomes of Newton's Method. The first is that the iterations converge to a limit such that successive iterations yield the same result (up to a certain level of accuracy). This was witness in the cases of $x_0 = 9, 12, 15$. This happens when the value that the function takes at the value of x found is either close enough to zero or if the derivative is sufficiently large while the function is taking a value close to zero such that a repeated iteration does not alter the term to a recognizable precision.

The second case, witnessed at values of $x_0 = 3, 6, 18, 21$, is when the iterations converge to a limit, but the last digit oscillates between two numbers. This can happen when the fraction of $\frac{f(x)}{f'(x)}$ is equal up to a sign for both values of x, leading to oscillations between two values.

A third possibility is possible as well. If we were to extend this function to see the other local maxima and minima, it could be shown that some initial values of x_0 would result in oscillations about those local extrema and not the zeroes that are being searched for.

♦

We will now show why this method will converge to the roots of a function as long as a certain iteration is close enough to the desired root.

Theorem 3 *Let $f(x)$ be a function such that $f''(x)$ exists. Using Newton's Method, if x_0 is sufficiently close to the root α, then the error, $e_n = x_n - \alpha$, is given by*

$$e_{n+1} \approx \frac{1}{2} e_n^2 \left(\frac{f''(\alpha)}{f'(\alpha)} \right) .$$

Proof: The Taylor series approximations to $f(x)$ and $f'(x)$ near α are given by

$$\begin{aligned} f(x) &\approx f(\alpha) + f'(\alpha)(x-\alpha) + \frac{1}{2}f''(\alpha)(x-\alpha)^2 \text{ and} \\ f'(x) &\approx f'(\alpha) + f''(\alpha)(x-\alpha). \end{aligned}$$

As $f(\alpha) = 0$, these approximations yield

$$\begin{aligned} e_{n+1} &= x_{n+1} - \alpha = x_n - \frac{f(x_n)}{f'(x_n)} - \alpha = (x_n - \alpha) - \frac{f(x_n)}{f'(x_n)} = e_n - \frac{f(x_n)}{f'(x_n)} \\ &\approx e_n - \frac{f'(\alpha)(x_n-\alpha) + \frac{1}{2}f''(\alpha)(x_n-\alpha)^2}{f'(\alpha) + f''(\alpha)(x_n-\alpha)} \\ &= e_n - \frac{f'(\alpha)e_n + \frac{1}{2}f''(\alpha)e_n^2}{f'(\alpha) + f''(\alpha)e_n} \\ &= \frac{f'(\alpha)e_n + f''(\alpha)e_n^2 - f'(\alpha)e_n - \frac{1}{2}f''(\alpha)e_n^2}{f'(\alpha) + f''(\alpha)e_n} \\ &= \frac{1}{2}e_n^2 \frac{f''(\alpha)}{f'(\alpha) + f''(\alpha)e_n}. \end{aligned}$$

Assuming $f'(\alpha) \neq 0$, if e_n is close to zero, we have

$$e_{n+1} \approx \frac{1}{2}e_n^2 \frac{f''(\alpha)}{f'(\alpha) + f''(\alpha)e_n} \approx \frac{1}{2}e_n^2 \left(\frac{f''(\alpha)}{f'(\alpha)}\right).$$

■

This error formula has many implications. However, possibly the two most important are that if x_0 is close enough to a given root, then Newton's Method will always converge, and if x_0 is close enough to a given root, then Newton's Method will always converge to that root and not to another root. As the error from one step to the next goes down proportionately to the square of the previous error, if e_0 is small, then squaring this error will yield an even smaller error for successive iterations. Additionally, when iterating from one step to the next, the magnitude of the difference is dependent on the value that the function takes at the point. If x_0 is close enough to a given root, then $f(x_0)$ will be close to zero, meaning iterating from one step to the next will not lead to a value appreciably far away. This, combined with the error decreasing proportionately with the square of the previous error assures one that the next term will be even closer to the same root that you began close to.

2.3 Bisection

In addition to Newton's Method, there is another common method used to find the root of a given function. The method of bisection, albeit much slower than Newton's Method, is an algorithm to locate the root of a given function that does not behave improperly when iterating from a point with derivative close to zero.

The method of bisection is the following. Suppose we are given a function, $f(x)$, and that there are two values, a, b in the domain of $f(x)$ such that $f(a)f(b) < 0$. Then, if $f(x)$ is

continuous, either $f(a) < 0$ or $f(b) < 0$, exclusively. Thus, the Intermediate Value Theorem assures us that there exists a c with $a < c < b$ such that $f(c) = 0$. In order to find this value, introduce a new value, m, and set $m = \frac{a+b}{2}$. Then, one of the following happens:

1. $f(m)$=0 and you are done.
2. $f(a)f(m) < 0$ or $f(b)f(m) < 0$ but not both. If $f(m) \neq 0$, then $f(a)$, $f(b)$, $f(m)$ cannot all have the same sign as $f(a)$ and $f(b)$ have opposite signs. Thus, either $f(a)$ and $f(m)$ or $f(b)$ and $f(m)$ have opposite signs, but not both.

We formalize the method below. In order to solve $f(\alpha) = 0$,

1. Choose a, b such that $f(a)f(b) < 0$.
2. Let $m = \frac{a+b}{2}$. If $f(m) = 0$, stop. If $f(a)f(m) < 0$, replace b by m. Otherwise, replace a by m.
3. Repeat the previous step.

Although this method is possibly more intuitive than Newton's Method, the rate of convergence is much slower. Since $|a - m| = |b - m| = \frac{1}{2}(b - a)$, each iteration cuts the interval containing the root, α, in half. So if n iterations are performed to obtain the sequence of midpoints $m_1, m_2, \ldots, m_n$, then we have

$$|m_n - \alpha| \leq \left(\frac{1}{2}\right)^n (b - a).$$

This expression gives us a bound for the error after n iterations of the method of bisection.

Example 3 Suppose we are given the function $f(x) = 0.01x^2 + x + 50$. As $\lim_{x\to-\infty} f(x) = -\infty$, $\lim_{x\to+\infty} f(x) = +\infty$, $f(x)$ is continuous, we know that $f(x)$ has at least one real root. Additionally, as the derivative, $f'(x) = 0.05x^4 + 1$ is strictly positive, we know that our initial function is monotonically increasing and therefore has exactly one real root. We will find this root using the method of bisection along with *Mathematica*. First, we desire an estimate for the root. With the knowledge that the root lies somewhere in the interval $[-12, 12]$, we plot the function to obtain a more precise estimate. From the plot, it appears as if the real

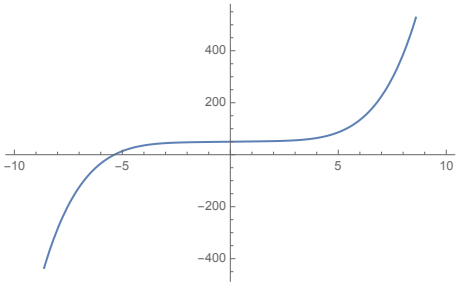

Figure 3: Plot to obtain estimate of the root of the given function

root lies in the interval $[-6, -5]$. With this set as our initial bounds for bisection (that is,

$a = -6$, $b = -5$), we iterate 20 times to obtain a value for the root of $f(x)$. Some of the code is displayed below:

```
ClearAll[l, r, m]; Clear[f] f[x_] := 0.01 x^5 + x + 50;
l[n_] := l[n]; r[n_] := r[n]; m[n_] := m[n] = (l[n] + r[n])/2;
l[1] = -6; r[1] = -5; i = 1;
Do[l[i + 1] = l[i];
r[i + 1] = r[i];
If[f[l[i]] f[m[i]] < 0, r[i + 1] = m[i], l[i + 1] = m[i]], i, 20];
Table[N[l[i], 17], SetPrecision[f[l[i]], 17], N[r[i], 17],
SetPrecision[f[r[i]], 17], i, 20]
```

left	f(left)	right	f(right)
−6.0000000000000000	−33.760000000000005	−5.0000000000000000	13.750000000000000
−5.5000000000000000	−5.8284374999999997	−5.0000000000000000	13.750000000000000
−5.5000000000000000	−5.8284374999999997	−5.2500000000000000	4.8662011718749980
−5.3750000000000000	−0.23841644287109176	−5.2500000000000000	4.8662011718749980
−5.3750000000000000	−0.23841644287109176	−5.3125000000000000	2.3724639415740967
−5.3750000000000000	−0.23841644287109176	−5.3437500000000000	1.0819257941842082
−5.3750000000000000	−0.23841644287109176	−5.3593750000000000	0.42551291414536507
−5.3750000000000000	−0.23841644287109176	−5.3671875000000000	0.094491907102344896
−5.3710937500000000	−0.071725834726521498	−5.3671875000000000	0.094491907102344896
−5.3710937500000000	−0.071725834726521498	−5.3691406250000000	0.011442080007341815
−5.3701171875000000	−0.030127108349645937	−5.3691406250000000	0.011442080007341815
−5.3696289062500000	−0.009338829257769240	−5.3691406250000000	0.011442080007341815
−5.3696289062500000	−0.009338829257769240	−5.3693847656250000	0.0010525512262589132
−5.3695068359375000	−0.004142905162659360 5	−5.3693847656250000	0.0010525512262589132
−5.3694458007812500	−0.001545119298391739 4	−5.3693847656250000	0.0010525512262589132
−5.3694152832031250	−0.000246269618862982 04	−5.3693847656250000	0.0010525512262589132
−5.3694152832031250	−0.000246269618862982 04	−5.3694000244140625	0.00040314440796151985
−5.3694152832031250	−0.000246269618862982 04	−5.3694076538085937	0.00007843829562403925 6
−5.3694114685058594	−0.000083915436356107875	−5.3694076538085937	0.00007843829562403925 6
−5.3694095611572266	−0.000002738514048417 0734	−5.3694076538085937	0.00007843829562403925 6

Table 1: Data from 20 iterations of the bisection method on *Mathematica*

As seen in Table 1, it appears as though the left bound is producing a value for $f(x)$ that is closer to zero. Thus, we choose -5.3694095611572266 to be our root. However, using *Mathematica*, we find the actual root to be -5.369409496812525, which is an error of 6.43447×10^{-8}.

♦

Example 3 also lends credibility to the error bound derived above. From the example, we had 20 iterations. Thus, we can bound the error from above by $\left(\frac{1}{2}\right)^{20} = 9.53674 \times 10^{-7}$. The error obtained after 20 iterations was 6.43447×10^{-8}, which we notice is less than the bound, so our maximum error agrees with numerical results for one set of data.

3 Quadrature

It is often the case that we encounter integrals that we cannot solve exactly. Although many methods exist to approximate integrals numerically, one of the most powerful and ubiquitous techniques of numerical integration is the employment of a quadrature formula.

Definition 1 A formula of the form $\sum_{i=1}^{n} A_i f(x_i)$ used to approximate $\int_a^b f(x)dx$ is called a quadrature formula or quadrature rule. If this quadrature formula is capable of integrating all polynomials of degree at most D, where $D \geq 0$, but not at least one polynomial of degree $D+1$ then the quadrature formula is said to have degree of precision D.

As with any approximation, however, there exists an error that propagates. This, of course, is the difference between the desired integral and our numerical approximation.

Definition 2 $E(f)$ is the error in the quadrature formula.

$$E(f) = \int_a^b f(x)dx - \sum_{i=1}^{n} A_i f(x_i)\,.$$

For all of the quadrature formulae that we will encounter, we cite that the error can be written in the form

$$E(f) = c\Big(\frac{b-a}{2}\Big)^{D+2} \frac{d^{D+1}}{dx^{D+1}} f(\xi) \tag{1}$$

where c is some constant and ξ satisfies $a < \xi < b$.

We can extend the definition of the degree of precision, which will prove useful as the degree of precision aids in exploiting other properties of certain quadrature formulae. Furthermore, in order to be sure that we have obtained the proper expressions for the degree of precision and constants in expressions for error, we cite two theorems, but leave the proof to the reader.

Theorem 4 *A quadrature formula has degree of precision D if and only if the quadrature formula satisfies two conditions. First,*

$$E(x^j) = 0 \quad \forall\, 0 \leq j \leq D.$$

And second,

$$E(x^{D+1}) \neq 0.$$

We can also use the degree of precision to help find the constant that is cited in the form of the error formula.

Theorem 5 *Suppose a quadrature formula has degree of precision D. Then*

$$c = \frac{E(x^{D+1})}{\left(\frac{b-a}{2}\right)^{D+2}(D+1)!}\,. \tag{2}$$

With this in mind, the algorithm for finding a quadrature formula becomes readily accessible. In order to find a quadrature formula $\sum_{i=1}^{n} A_i f(x_i)$ for $\int_{-1}^{1} f(x)dx$,

1. Choose $x_1, x_2, \ldots, x_n$ and use part one of Theorem 4 to find the A_i until part 2 of the same Theorem holds.

2. Use Equation 2 and the degree of precision obtained from Step 1 to fix c.

3. Change variables to $t = \frac{b-a}{2}x + \frac{b+a}{2}$, and convert $\int_{-1}^{1} f(x)dx$ to $\int_a^b f(t)dt$. You may then use this Step 1 to extend to a quadrature formula for $\int_a^b f(t)dt$.

It can be shown that Step 3 of this algorithm will yield

$$\int_a^b f(t)dt = \Big(\sum_{i=1}^{n} (\frac{b-a}{2}) A_i f(\frac{b-a}{2}x_i + \frac{b+a}{2}) \Big) + c(\frac{b-a}{2})^{D+2} \frac{d^{D+1}}{dt^{D+1}} f(\xi_t) \tag{3}$$

where $\xi_t = \frac{b-a}{2}\xi + \frac{b+a}{2}$. Although we have generalized our result thus far to be a quadrature formula applicable on any subdomain of the domain of our integrand, the larger the interval we are integrating, the larger our potential error will be for functions that are not integrated exactly. Thus, in order to reduce this error, we break up our interval into subintervals, and perform integrals by applying our quadrature formula to each of the integrals on the smaller domains. In other words, we break our integral into $N \in \mathbb{Z}^+$ smaller integrals by setting $h = \frac{b-a}{N}$ and create the sequence with $x_i = a + ih$. Then,

$$a = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = b$$

and

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \cdots + \int_{x_{N-1}}^{x_N} f(x)dx\,.$$

When finding quadrature formulae, we will apply the formula we found to each one of these integrals, but the error that propagates will have terms including the function evaluated at points on each subinterval. However, leaving the proof to the reader, we are able to get around this.

Theorem 6 *Let $g(x)$ be a function continuous on the interval $[a, b]$. If given a sequence $\xi_1, \xi_2, \ldots, \xi_N$ in $[a, b]$, then there exists some $\xi \in [a, b]$ such that*

$$\frac{g(\xi_1) + g(\xi_2) + \ldots + g(\xi_N)}{N} = g(\xi).$$

Equipped with this methodology, we are able to find quadrature formulae for common numerical integration methods.

3.1 Right-Hand Riemann Sum

We now will use this algorithm to find a quadrature formula when $n = 1$ and $x_1 = 1$. With these conditions, we have $A_1 f(1) + E(f)$. We know that D must be at least 0, so we begin with $E(1) = 0$, and

$$\begin{aligned} \int_{-1}^{1} dx &= A_1 f(1) \\ 2 &= A_1 \end{aligned}$$

Increasing the order of the polynomial by 1, we set $f(x) = x$, and then have

$$\begin{aligned} \int_{-1}^{1} x dx &= A_1 f(1) + E(x) \\ \frac{x^2}{2}\Big|_{-1}^{1} &= 2 + E(x) \\ 0 &= 2 + E(x) \end{aligned}$$

So $E(x) = -2$ and $D = 0$ by Theorem 4. Now, by equation 5, we have

$$\begin{aligned} c &= \frac{E(x)}{\left(\frac{b-a}{2}\right)^{0+2}(0+1)!} \\ &= \frac{-2}{\left(\frac{1-(-1)}{2}\right)^{2}(1)!} \\ &= -2\,. \end{aligned}$$

And thus we have

$$\int_{-1}^{1} f(x)dx = 2f(1) - \frac{1}{2}f'(\xi)$$

by Equation 1. Now, changing variables, and using Equation 3, we obtain

$$\begin{aligned} \int_{a}^{b} f(t)dt &= \int_{-1}^{1} f(\frac{b-a}{2}x + \frac{b+a}{2})(\frac{b-a}{2})dx \\ &= \frac{b-a}{2}2f(\frac{b-a}{2}(1) + \frac{b+a}{2}) - 2(\frac{b-a}{2})^2\frac{d}{dt}f(\xi_t) \\ &= (b-a)f(b) - \frac{1}{2}(b-a)^2 f'(\xi_t) \end{aligned}$$

Now, we break our integral into $N \in \mathbb{Z}^+$ smaller integrals by setting $h = \frac{b-a}{N}$ and creating the sequence with $x_i = a + ih$. Then

$$a = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = b$$

and

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \cdots + \int_{x_{N-1}}^{x_N} f(x)dx\,.$$

Applying our quadrature formula to each one of these integrals, we obtain

$$\begin{aligned}\int_a^b f(x)dx &= \left(hf(x_1) - \frac{1}{2}h^2 f'(\xi_1)\right) + \ldots + \left(hf(x_N) - \frac{1}{2}h^2 f'(\xi_N)\right) \\ &= \sum_{i=1}^{n}\left(hf(x_i) - \frac{1}{2}h^2 f'(\xi_i)\right)\end{aligned}$$

Where $x_0 < \xi_i < x_i$. However, we know that given a sequence $\xi_1, \xi_2, \cdots, \xi_N$ in $[a, b]$, if $g(x)$ is continuous on $[a, b]$, then there exists some ξ such that

$$\frac{g(\xi_1) + g(\xi_2) + \ldots + g(\xi_N)}{N} = g(\xi).$$

Thus, we can simplify our expression to

$$\int_a^b f(x)dx = \sum_{i=1}^{N}\left(hf(x_i)\right) - \frac{1}{2}h(b-a)f'(\xi)\,,$$

which we recognize as the form of the Right-hand Riemann Sum that is often an introduction to integration formalism.

3.2 3-point Gaussian

We will now change our initial conditions to develop the quadrature formula for the 3-point Gaussian Quadrature Formula. The idea of any Gaussian quadrature rule is to allow one's self the freedom to choose not only the weighting coefficients for preselected points in the subintervals of integration, but also the location of the x-values at which the function is to be evaluated[2]. This, effectively, doubles the degrees of freedom, which partially explains the high order which we will witness in this formula. To do this, we now use the algorithm with different conditions. We will find a quadrature formula, $\sum_{i=1}^{n} A_i f(x_i) + E(f)$ for $\int_{-1}^{1} f(x)dx$ where $n = 3$ with $x_3 = -x_1$, $x_2 = 0$, and $A_1 = A_3$. So

$$\sum_{i=1}^{n} A_i f(x_i) = A_1 f(x_1) + A_2 f(0) + A_1 f(-x_1).$$

We begin by imposing that the error be 0 for as high order polynomial as possible. We begin iterating with a polynomial of degree 0. However, we quickly note from the form of our sum that when we are integrating any odd polynomial we will only obtain a trivial expression that $0 = A_1 + 0A_2 - A_1$, so we omit these iterations.

$$E(1) = 0 \Rightarrow \int_{-1}^{1} dx = 2 \quad = \quad A_1 f(x_1) + A_2 f(0) + A_1 f(-x_1)$$

$$
\begin{aligned}
& & = & \; 2A_1 + A_2 \\
E(x^2) = 0 \Rightarrow \int_{-1}^{1} x^2 dx = \frac{2}{3} & = & A_1 f(x_1) + A_2 f(0) + A_1 f(-x_1) \\
\frac{1}{3} & = & A_1 x_1^2 \\
E(x^4) = 0 \Rightarrow \int_{-1}^{1} x^4 dx = \frac{2}{5} & = & A_1 f(x_1) + A_2 f(0) + A_1 f(-x_1) \\
\frac{1}{5} & = & A_1 x_1^4
\end{aligned}
$$

We have now arrived at 3 equations for our 3 unknowns, so as long as these equations are linearly independent, we should be able to solve. With the help of *Mathematica*, we obtain the following (ignoring the negative solution for x_1):

$$x_1 = \sqrt{\frac{3}{5}}, \quad A_1 = \frac{5}{9}, \quad A_2 = \frac{8}{9}.$$

We now move to examine the error that arises from integrating a polynomial of degree 6 with these constants set:

$$
\begin{aligned}
\int_{-1}^{1} x^6 dx = \frac{2}{7} & = & \frac{5}{9}\Big(\frac{3}{5}\Big)^3 + \frac{5}{9}\Big(\frac{3}{5}\Big)^3 + E(x^6) \\
\frac{2}{7} & = & \frac{10}{9}\frac{27}{125} + E(x^6) \\
\frac{8}{175} & = & E(x^6)
\end{aligned}
$$

And we therefore recognize that $D = 5$. As before, we will compute the constant c, by using equation 5. We have

$$
\begin{aligned}
c & = & \frac{E(x^6)}{\Big(\frac{b-a}{2}\Big)^7 6!} \\
& = & \frac{\frac{8}{175}}{\Big(\frac{2}{2}\Big)^7 720} \\
& = & \frac{1}{15750}
\end{aligned}
$$

And thus we have

$$\int_{-1}^{1} f(x)dx = \frac{5}{9} f\Big(\sqrt{\frac{3}{5}}\Big) + \frac{8}{9} f(0) + \frac{5}{9} f\Big(-\sqrt{\frac{3}{5}}\Big) + \frac{1}{15750}\Big(\frac{b-a}{2}\Big)^7 f^{(6)}(\xi)$$

To extend this to an integral with different bounds of integration, we change variables.

$$\int_a^b f(t)dt \;\; = \;\; \int_{-1}^{1} f(\frac{b-a}{2}x + \frac{b+a}{2})(\frac{b-a}{2})dx$$

$$= \frac{1}{9}\frac{b-a}{2}\Big(5f(\frac{b-a}{2}\sqrt{\frac{3}{5}}+\frac{b+a}{2})+8f(\frac{b+a}{2})+5f(\frac{a-b}{2}\sqrt{\frac{3}{5}}+\frac{b+a}{2})\Big)$$
$$+ \frac{1}{15750}\Big(\frac{b-a}{2}\Big)^7 f^{(6)}(\xi_t)$$

Now, we break our integral into $N \in \mathbb{Z}^+$ smaller integrals by setting $h = \frac{b-a}{N}$ and creating the sequence with $x_i = a + ih$. Then

$$a = x_0 < x_1 < x_2 < \ldots < x_{N-1} < x_N = b$$

and

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \ldots + \int_{x_{N-1}}^{x_N} f(x)dx\,.$$

Applying our quadrature formula to each one of these integrals, we obtain

$$\int_a^b f(t)dt = \frac{h}{18}\sum_{i=1}^{n}\Big(5f(\frac{h}{2}\sqrt{\frac{3}{5}}+m_i)+8f(m_i)+5f(\frac{-h}{2}\sqrt{\frac{3}{5}}+m_i)\Big)$$
$$+\frac{1}{15750}\frac{h^6}{2^7}(b-a)f^{(6)}(\xi)$$

where the m_i represent the midpoint between x_{i-1} and x_i.

3.3 Other Common Quadrature Formulae

Although we will not develop these formulae in depth, there are three other common quadrature rules that are extremely common.

The first of these is the trapezoid rule. It can be derived with the initial conditions $n = 2$, $x_1 = -1$, and $x_2 = 1$. Carrying out the three steps we get

$$\int_a^b f(x)dx = \Big(\frac{b-a}{2}(f(a)+f(b))\Big) - \frac{1}{12}(b-a)^3 f''(\xi).$$

We can extend this to a composite rule by dividing our interval and we arrive at

$$\int_a^b f(x)dx = \Big(h\big(\frac{1}{2}f(a)+f(x_1)+f(x_2)+\ldots+f(x_{N-1}+\frac{1}{2}f(b)\big)\Big)$$
$$-\frac{1}{12}h^2(b-a)f''(\xi).$$

where we have divided our integrand into N pieces of equal length and defined $h = \frac{b-a}{N}$ and ξ is some point in the interval.

Now we turn to the Midpoint Rule. This formula can be obtained by setting using the same technique as $n = 1$ but not fixing x_1 and hoping to obtain a degree of precision greater than or equal to one. We finally arrive at the result

$$\int_a^b f(x)dx = \Big((b-a)f\big(\frac{a+b}{2}\big)\Big) + \frac{1}{24}(b-a)^3 f''(\xi)$$

We can extend this to a composite rule by dividing our interval, with similarly defined quantities as in the composite Trapezoid Rule, and we arrive at

$$\int_a^b f(x)dx = \Big(h\big(f(\frac{a+x_1}{2}) + f(\frac{x_1+x_2}{2}) + \ldots + f(\frac{x_{N-1}+b}{2})\big)\Big) - \frac{1}{24}h^2(b-a)f''(\xi).$$

Lastly, we look to obtain Simpson's Rule, a quadrature formula with $n = 3$, $x_1 = -1$, $x_2 = 0$, and $x_3 = 1$, and we arrive at

$$\int_a^b f(x)dx = \Big(\frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b))\Big) - \frac{1}{2880}(b-a)^5 f^{(4)}(\xi)$$

We can extend this to a composite rule by dividing our interval, with similarly defined quantities as in the composite Trapezoid Rule, and we arrive at

$$\int_a^b f(x)dx = \Big(\frac{h}{6}\big(f(a) + 4(f(\frac{a+x_1}{2}) + 2f(x_i) + 4f(\frac{x_1+x_2}{2}) + 2f(x_2) + \ldots + 2f(x_{N-1}) + 4f(\frac{x_{N-1}+b}{2}) + f(b)\big)\Big) - \frac{1}{2880}h^4(b-a)f^{(4)}(\xi).$$

We close with an example of the use of these composite rules.

Example 4 Using *Mathematica*, we apply five different numerical integration techniques in order to approximate

$$\int_a^b e^{x^2}dx$$

with $a = 0.70$ and $b = 2.70$. We will use various numbers of subdivisions (10, 100, and 1000), and use techniques that were outlined in previous notes and homeworks, namely the Right-hand Riemann Sum, the Composite Trapezoid Rule, the Composite Midpoint Rule, the Composite Simpson's Rule, and the Composite 3-point Gaussian Quadrature Formula. Samples of code for each technique are presented below, but final results are saved for the next section.

For the Right-hand Riemann Sum, we establish the shell of our code for 10 subdivisions as:

```
Clear[f]; f[t_] := Exp[t^2];
a = 0.70; b = 2.70;
ndiv = 10; h = (b - a)/ndiv;
sum = 0; i = 1;
Do[xi = a + h*i; sum = sum + h*f[xi], {i, ndiv}];
SetPrecision[sum, 17]
```

For the Composite Trapezoid Rule, we use the same structure as the method prior, but replace the `Do[ ]` control structure with the following:

```
Do[xim1 = a + h (i - 1); xi = xim1 + h;
sum = sum + h (f[xim1] + f[xi])/2, i, ndiv];
```

As before, to compute the Composite Midpoint Rule, we just alter the `Do[ ]` structure with:

```
Do[xim1 = a + h (i - 1); xi = xim1 + h; mid = (xim1 + xi)/2;
sum = sum + h*f[mid], i, ndiv];
```

For the Composite Simpson's Rule, we change the loop to:

```
Do[xim1 = a + h (i - 1); xi = xim1 + h; mid = (xim1 + xi)/2;
sum = sum + h (f[xim1] + 4 f[mid] + f[xi])/6, i, ndiv];
```

Finally, to compute the 3-point Gaussian Quadrature Formula, we use the following loop in the original shell:

```
Do[xim1 = a + h(i-1); xi = xim1 + h; mid = (xim1 + xi)/2;
sft = h*Sqrt[3/5]/2;
sum = sum + h(5f[mid-sft]+8f[mid]+5f[mid+sft])/18, i, ndiv];
```

Our results are presented below, keeping 17 decimal accuracy in all cases. Although these

	Number of Subdivisions		
Method	10	100	1000
RHRS	467.93287647659974	310.66389755219643	297.22742463823869
Trapezoid	321.53903837819672	296.02451374235613	295.76348625725439
Midpoint	283.09247022463182	295.62903980074151	295.75953037058667
Simpson	295.90799294248677	295.76086444794635	295.76084899947602
Gaussian	295.76049751171365	295.76084899755529	295.76084899793017

Table 2: Numerical integration of e^{x^2} with provided bounds in *Mathematica*

computations seem to converge to similar values, thorough analysis calls for a comparison of the error. However, in order to do this, we need the first 6 derivatives of our function, $f(x) = e^{x^2}$. Recognizing that the k^{th} derivative of this function can be written in the form $f(x)p_k(x)$, where $p_k(x)$ is a degree k polynomial with positive integer coefficients, we provide this first 6 derivatives in this form:

$$\begin{aligned}
f'(x) &= (2x)e^{x^2} \\
f''(x) &= (2+4x^2)e^{x^2} \\
f^{(3)}(x) &= (12x+8x^3)e^{x^2} \\
f^{(4)}(x) &= (16x^4+48x^2+12)e^{x^2} \\
f^{(5)}(x) &= (32x^5+160x^3+120x)e^{x^2} \\
f^{(6)}(x) &= (64x^6+480x^4+720x^2+120)e^{x^2}
\end{aligned}$$

It is clear that for all $x > 0$, $f^{(k+1)}(x) > 0$ which means that the k^{th} derivative is always increasing when x is positive and will therefore take a maximum on the given domain at the right boundary, $x = 2.70$. In a given computation, if there are N subdivisions, and we set $h = \frac{b-a}{N}$, then the errors can be bounded by

$$|E(f)| \leq |c|h^k(b-a)e^{2.70^2}p_k(2.70),$$

where c and k are constants dependent on the quadrature rule being analyzed.

In general, we know that if a quadrature formula has degree of precision D, then we can find c for the case of one division (which we extend to composite rules later) by

$$c = \frac{E(x^{D+1})}{\left(\frac{b-a}{2}\right)^{D+2}(D+1)!}.$$

Thus, in order to calculate these constants, we first cite the different degrees of precision for each quadrature formula: We have seen in all of our different quadrature techniques that

Method	D
RHRS	0
Trapezoid	1
Midpoint	1
Simpson	3
Gaussian	5

Table 3: Degree of Precision for quadrature formulae

the error term takes the form of

$$|E(f)| = |c|h^{D+1}(b-a)f(\xi),$$

so we can simply cite our desired value for k from D, and we compute the various values of $|c|$ using *Mathematica*. Equipped with these constants, we calculate our error upper bounds, which are presented below:

			Number of Subdivisions		
Method	$\lvert c\rvert$	k	10	100	1000
RHRS	$\frac{1}{2}$	1	1.58282×10^{3}	1.58282×10^{2}	1.58282×10^{1}
Trapezoid	$\frac{1}{12}$	2	3.04448×10^{2}	3.04448×10^{0}	3.04448×10^{-2}
Midpoint	$\frac{1}{24}$	2	1.52224×10^{2}	1.52224×10^{0}	1.52224×10^{-2}
Simpson	$\frac{1}{2880}$	4	1.97400×10^{0}	1.97400×10^{-4}	1.97400×10^{-8}
Gaussian	$\frac{1}{2016000}$	6	5.18048×10^{-3}	5.18048×10^{-9}	5.18048×10^{-15}

Table 4: Bounding constants for numerical integration error

Thus, by looking at the error estimate from the 3-point Gaussian Quadrature Formula with 1000 subdivisions, we can guarantee that our numerical result is accurate to up to 13 decimal places, as the error only appears as an additive term to the 15th decimal place, which could potentially alter the digit immediately adjacent. Additionally, by looking at the error estimate of the Midpoint Rule with 1000 subdivisions, we can only ensure that our

computation is accurate up to the tens digit, as variation in the tenths place allows for a change of the ones digit.

We now change our analysis to the Right-hand Riemann Sum estimates. As our function is concave up and increasing, if we use a fewer number of subdivisions, then the numerical integration will yield a result too high as it is computing the integral by using an estimated average on each subdomain that is too large. It is for this same reason that they are over-estimates of the true integral.

Shifting focus yet again, we look to the Trapezoid Rule estimates. The estimates decrease when increasing the number of subdivisions. This is because the trapezoid rule takes the two endpoints of the function and interpolates linearly in order to integrate. However, if one is interpolating an increasing concave up function linearly, the line will always be above the true function, and as the function is increasing for all positive x for all ordered derivatives, the function is increasing faster and faster, thus the larger the interval for a interval, the farther away the interpolating line will be away from the function. This does, however, explain why the estimates are over-estimates, as the line always sits above the actual function to be approximated.

Finally, we consider the Midpoint Rule estimates. In these cases, the estimates are under-estimates. Again, we consider the form of the k^{th} derivative of the function for positive values of x. These higher order derivatives are always increasing, which tells us that the function is not only increasing, but the rate of increase is also increasing (concave up), and so on. Because of this, if we approximate the value of the function at the midpoint on an interval, the amount that the function grew from the left boundary to the midpoint is less than the amount that the function will grow from the midpoint to the right boundary. Thus, the value of the function at the midpoint is an underestimate to the average value of the function on the interval, and thus, the estimates from the Midpoint Rule are under-estimates of the true integral of our function. ⧫

4 Numerical Methods for Differential Equations

Differential Equations exist for the purpose of hoping to describe systems that are subject to change. As, in application, systems often have levels of complexity or form that escape the analytic techniques that mathematicians have developed, it is often necessary to develop methods to approximate solutions when analytic solutions are not accessible. Additionally, it might also be the case that even if a solution is possible, it might be complicated to the point were a numerical approximation proves to be more useful.

As is the case with analytic solutions to differential equations, different orders of differential equations lend themselves more readily to different numerical techniques. Arguably the most ubiquitous of these techniques apply to what we call first order ordinary differential equations.

Definition 3 A first order ordinary differential equation is an equation of the form

$$\frac{dy}{dx} = f(x, y)\,. \tag{4}$$

A solution of equation 4 on the interval $a \leq x \leq b$ is a function $y(x)$ that satisfies

$$\frac{dy}{dx} = f(x, y(x))$$

However, we can also require that the function takes a certain value for a certain x; in other words, we stipulate that

$$y(a) = A \tag{5}$$

for some A, called an inital condition. Combining equations 4 and 5, we have an initial value problem.

We will describe three specific techniques that are applicable for approximating first order initial value problems. However, before applying different techniques to approximating the solution to the differential equation, it is important to note two assumptions that are often left implicit about the function $f(x, y)$. These assumptions are:

1. We will assume that $f(x, y)$ is a continuous function for all $x \in [a, b]$ and for all y.
2. We will assume that $f(x, y)$ also satisfies a Lipschitz condition: There exists a constant L, called a Lipschitz constant, such that

$$|f(x, u) - f(x, v)| \leq L|u - v| \tag{6}$$

   for all $x \in [a, b]$ and for all u, v.

Clearly, if one Lipschitz constant exists for a function, then there are infinitely many. Thus, we present a way of fixing the Lipschitz constant. By the Mean Value Theorem, if we consider x fixed, then

$$\frac{\partial f}{\partial y}(x, c) = \frac{f(x, u) - f(x, v)}{u - v}$$

for some c between u and v. Thus, if

$$L = \max_{\substack{x \in [a,b] \\ \forall y}} \left| \frac{\partial f}{\partial y}(x, c) \right| \tag{7}$$

exists, then

$$\begin{aligned} & \left| \frac{f(x, u) - f(x, v)}{u - v} \right| \leq L \\ \Rightarrow \qquad & |f(x, u) - f(x, v)| \leq L|u - v|, \end{aligned}$$

and this L can be used for the Lipschitz constant. We note an interesting theorem that arises from these assumptions.

Theorem 7 *Let $f(x, y)$ be continuous for $x \in [a, b]$ and for all y. Suppose $f(x, y)$ also satisfies a Lipschitz condition. Then the initial value problem in 4 and 5 has a unique solution $y(x)$ defined for all $x \in [a, b]$.*

4.1 Euler's Method

In order to approximate a solution to a differential equation numerically for all $x \in [a, b]$, we must create a sequence of points in the interval, x_i, and generate a corresponding sequence, y_i, such that $y_i \approx y(x_i)$. We create this sequence by setting $x_{i+1} - x_i = h = \frac{b-a}{N}$, giving us N evenly spaced points in the interval.

The simplest way to obtain a sequence, y_i is Euler's Method, which we motivate below.

Given a function, $y(x)$, we can Taylor expand around the point $x = x_i$, and this expansion is given by

$$y(x) = y(x_i) + y'(x_i)(x - x_i) + \frac{1}{2}y''(\xi)(x - x_i)^2$$

for some ξ between x_i and x. Letting $x \to x_{i+1}$, and noting $x_{i+1} - x_i = h$ and that $y'(x_i) = f(x_i, y(x_i))$,

$$y(x_{i+1}) = y(x_i) + hf(x_i, y(x_i)) + \frac{h^2}{2}y''(\xi_i) \tag{8}$$

for some ξ_i between x_i and x_{i+1}. Thus, $y(x_{i+1}) \approx y(x_i) + hf(x_i, y(x_i))$. And since we have satisfied the condition of $y_i \approx y(x_i)$, we arrive at a feasible numerical approximation to our differential equation. We formalize this method, Euler's Method, as follows:

$$y_0 = A \quad \text{and} \quad y_{i+1} = y_i + hf(x_i, y_i) \tag{9}$$

for $i = 0, 1, 2, \ldots, N$.

While this algorithm does seem intuitive, it is possibly best motivated graphically. As is shown in Figure 4, if we create a tangent line at the point $(x_i, y(x_i))$ with the slope $y'(x_i) = f(x_i, y(x_i))$, then we extend this line to the point x_{i+1}, then we set y_{i+1} equal to the value the line takes at this point. More rigorously, we have

$$\begin{aligned} & \frac{y_{i+1} - y_i}{x_{i+1} - x_i} &&= f(x_i, y_i) \\ \Rightarrow \quad & \frac{y_{i+1} - y_i}{h} &&= f(x_i, y_i) \\ & \Rightarrow \quad y_{i+1} &&= y_i + hf(x_i, y_i), \end{aligned}$$

which is how it appears in equation 9.

As with any approximation, we would like to know how close our result is to the exact solution. In the case of Euler's Method, errors accumulate with each iteration as successive estimates are created by using previous terms in our generated series which already carry some error. We are interested in bounding this error for the computation of $y(x_N) = y(b)$.

Theorem 8 *When approximating a solution to an ordinary differential equation on the interval $[a, b]$ with Euler's Method, the error can be bounded above by*

$$\max_{0 \le i < N} |y(x_i) - y_i| \le h\left(\frac{M}{2L}(e^{L(b-a)} - 1)\right),$$

where $M = \max_{x \in [a,b]} |y''(x)|$ and L is the Lipschitz constant in equation 7.

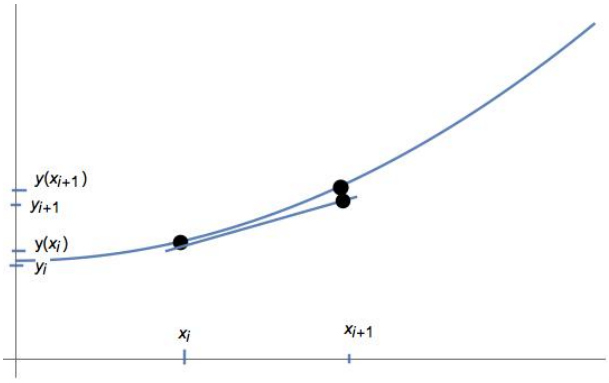

Figure 4: Graphical Explanation of Euler's Method

Proof: Let $E_i = y(x_i) - y_i$, the error for the i^{th} estimate. From equation 8 we have:

$$y(x_{i+1}) = y(x_i) + hf(x_i, y(x_i)) + \frac{h^2}{2}y''(\xi_i),$$

and from equation 9 we have:

$$y_{i+1} = y_i + hf(x_i, y_i).$$

Subtracting the top equation from the bottom gives us

$$\begin{aligned} y(x_{i+1}) - y_{i+1} &= y(x_i) - y_i + h\Big(f(x_i, y(x_i)) - f(x_i, y_i)\Big) + \frac{h^2}{2}y''(\xi_i) \\ \Rightarrow E_{i+1} &= E_i + h\Big(f(x_i, y(x_i)) - f(x_i, y_i)\Big) + \frac{h^2}{2}y''(\xi_i) \end{aligned}$$

From equation 6, $|f(x_i, y(x_i)) - f(x_i, y_i)| \leq L|y(x_i) - y_i|$ where from equation 7,

$$L = \max_{\substack{x\in[a,b] \\ \forall y}} \Big|\frac{\partial f}{\partial y}(x, y)\Big|.$$

Let $M = \max_{x\in[a,b]} |y''(x)|$. It then follows that

$$\begin{aligned} |E_{i+1}| &\leq |E_i| + hL|y(x_i) - y_i| + \frac{h^2}{2}M \\ &= |E_i| + hL|E_i| + \frac{h^2}{2}M \\ &= (1 + hL)|E_i| + \frac{h^2}{2}M \end{aligned}$$

In order to clean this up a little bit, we substitute $\delta = hL$, $|E_i| = d_i$, and $M^* = \frac{h^2}{2}M$, and thus we get

$$d_{i+1} \leq (1 + \delta)d_i + M^* \tag{10}$$

Claim: $d_{k+1} \leq (1 + \delta)^{k+1}d_0 + M^*(1 + (1 + \delta) + \cdots + (1 + \delta)^k)\ \forall\ k \geq 0$.

We will prove this claim by induction on k. For the basis of induction, when $k = 0$, we have equation 10 when $i = 0$. Now, for the inductive hypothesis, assume that $d_k \leq (1+\delta)^k d_0 + M^*(1 + (1+\delta) + \cdots + (1+\delta)^{k-1})$ for some $k \geq 0$. Then, by equation 10, we have

$$\begin{aligned} d_{k+1} &\leq (1+\delta)d_k + M^* \\ &\leq (1+\delta)[(1+\delta)^k d_0 + M^*(1 + (1+\delta) + \cdots + (1+\delta)^{k-1})] + M^* \\ &= (1+\delta)^{k+1} d_0 + M^*((1+\delta) + \cdots + (1+\delta)^k) + M^* \\ &= (1+\delta)^{k+1} d_0 + M^*(1 + (1+\delta) + \cdots + (1+\delta)^k). \end{aligned}$$

Thus our claim follows by induction. We note that $1 + (1+\delta) + \cdots + (1+\delta)^k$ is just a geometric series, which we can sum to

$$\frac{(1+\delta)^{k+1} - 1}{1 + \delta - 1} = \frac{(1+\delta)^{k+1} - 1}{\delta}.$$

When we substitute this into the claim we then have

$$d_{k+1} \leq (1+\delta)^{k+1} d_0 + M^*\Big(\frac{(1+\delta)^{k+1} - 1}{\delta}\Big). \tag{11}$$

Additionally, we recall that the Taylor series expansion of e^x about $x = 0$ gives $e^x = 1 + x + \dfrac{x^2}{2}e^\xi$ for some ξ between 0 and x. $\forall \xi$, we know that $e^\xi > 0$ and $\forall x \ \frac{x^2}{2} \geq 0$, so we have $1 + x \leq e^x$. When $x = \delta$, $1 + \delta \leq e^\delta \ \Rightarrow \ (1+\delta)^{k+1} \leq (e^\delta)^{k+1} = e^{(k+1)\delta}$. Putting this into equation 11 and set $\delta = hL$, $|E_i| = d_i$, and $M^* = \frac{h^2}{2}M$, we have

$$\begin{aligned} |E_{k+1}| &\leq (1+hL)^{k+1}|E_0| + \frac{h^2}{2}M\Big(\frac{(1+hL)^{k+1} - 1}{hL}\Big) \\ &\leq e^{(k+1)hL}|E_0| + h\Big(\frac{M}{2L}(e^{(k+1)hL} - 1)\Big) \end{aligned}$$

But $E_0 = y(x_0) - y_0 = A - A = 0$. And replacing $k+1$ with i, we obtain $|E_i| \leq h\Big(\frac{M}{2L}(e^{ihL} - 1)\Big)$. However, e^x is an increasing function, so

$$\max_{0 \leq i < N} |E_i| \leq h\left(\frac{M}{2L}(e^{NhL} - 1)\right).$$

But with one last substitution, we notice that $Nh = b - a$, so

$$\max_{0 \leq i < N} |y(x_i) - y_i| \leq h\left(\frac{M}{2L}(e^{L(b-a)} - 1)\right),$$

which was what we wanted. ■

Looking at this equation, we see that the term inside parenthesis is a constant independent of how many subdivisions are used in Euler's Method. Although having an upper bound for the error is helpful in determining the power of a certain algorithm, in general it is impossible to compute the M and L that we used because $y''(x)$ and $\frac{\partial f}{\partial x}$ often involve y and we do not know how to bound this function. An important identifier for the methods we discuss is the order of an algorithm.

Definition 4 *An algorithm has order p if the maximum error over a fixed interval is bounded above by h^pC for some constant C.*

With this definition, we note that Euler's Method is an algorithm of order 1, and the implication of this is discussed in detail later.

Before moving to a different technique, we posit a potential solution to the propagation of error from Euler's Method. One of the key pitfalls of this method is that it estimates the derivative only at one end of every subinterval. A better approach would be to use the average derivative over the subinterval, which is the idea behind the improved Euler's Method, which is given in one variable in [1] by

$$y_{i+1} = y_i + \frac{1}{2}h\Big(f(x_i) + f(\hat{x}_{i+1})\Big),$$

where $\hat{x}_{i+1} = x_i + f(x_i)h$.

4.2 Runga-Kutta Techniques

Although Euler's Method is an intuitive way to approximate a solution to an ordinary differential equation, this method can be improved. In the case of Euler's Method, for each iteration we only have one evaluation. If we replace $f(x_i, y_i)$ by an expression involving more than one evaluation of f at each step, then we have a Runga-Kutta Method of Order p, where p is the number of evaluations of f at each step. This can be done for $p = 1, 2, 3, 4$ but not 5. In the case of $p = 1$, we have Euler's Method. We will explain the cases for $p = 2$ and $p = 4$.

4.2.1 Runga-Kutta of Order 2

Runga-Kutta Method of Order 2, also referred to as Henn's Method, is obtained as follows:

$$\begin{aligned} y_0 &= A \\ k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + h, y_i + hk_1) \\ y_{i+1} &= y_i + h\Big(\frac{k_1 + k_2}{2}\Big) \text{ for } i = 0, 1, 2, \ldots \end{aligned}$$

It is important to note that the k_1 and k_2 appearing in these equations depend on the step that you are in. Additionally, in $k_2 = f(x_i + h, y_i + hk_1)$, $x_i + h = x_{i+1}$ and $y_i + hk_1 = y_i + hf(x_i, y_i)$ is the Euler approximation of $y(x_{i+1})$. Ultimately, the distinction between Euler's Method and the Runga-Kutta Method of Order 2 is that in Euler's Method, in the computation of y_{i+1}, the step size is multiplied by a function value, while in the Runga-Kutta Method of Order 2, you use the step size multiplied by the average of two function values. Most generally, we are able to boung the maximum error at each step over a fixed interval $[a, b]$ by h^2C for some constant C, thus the algorithm rightfully is named an algorithm of order 2.

4.2.2 Runga-Kutta of Order 4

There also exists a Runga-Kutta method of Order 4[1]. It is as follows:

$$\begin{aligned} y_0 &= A \\ k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1) \\ k_3 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2) \\ k_4 &= f(x_i + h, y_i + hk_3) \\ y_{i+1} &= y_i + h\Big(\frac{k_1 + 2k_2 + 2k_3 + k_4}{6}\Big) \text{ for } i = 0, 1, 2, \ldots \end{aligned}$$

As with Runga-Kutta of Order 2, the k_i are dependent on the step. Also, in $k_2 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1)$, $x_i + \frac{h}{2}$ is half way between x_i and x_{i+1}. If we set $x_{i+\frac{1}{2}} = x_i + \frac{h}{2}$, then we have $y_i + \frac{h}{2}k_1 = y_i + \frac{h}{2}f(x_i, y_i)$ is the Euler approximation of $y(x_i + \frac{1}{2})$. Additionally, as we had a weighted average of two terms in Runga-Kutta of order 2, in this method we compute y_{i+1} by using the weighted average of four function values. Finally, as an error bound is important in determining how accurate a technique is, we note that the maximum error at each step over a fixed interval $[a, b]$ is bounded above by h^4K for some constant K, which makes the algorithm of order 4.

Example 5 Throughout this example, we will use aforementioned techniques to attempt to numerically solve the initial value problem

$$\frac{dy}{dx} = 1 - Ax^{A-1}y \quad \text{where} \quad y(0) = 0$$

and then compare our numerical results to the exact solution. We will employ Euler's Method, Runga-Kutta of Order 2, and Runga-Kutta of Order 4, each with different subdivisions, to approximate $y(0)$, $y(1)$, $y(2)$, $y(3)$, $y(4)$, and $y(5)$.

Before utilizing our numerical techniques, we verify that $y(x) = e^{-x^A}\displaystyle\int_0^x e^{t^A}dt$ is a solution to the given problem. We do this by direct substitution. When taking the derivative of the expected solution, we need to exploit the product rule, and we recognize that given a function, f, continuous on an interval containing 0 and x, then

$$\frac{d}{dx}\int_0^x f(t)dt = f(x).$$

Thus, we have,

$$\frac{dy}{dx} = \frac{d}{dx}\Big(e^{-x^A}\int_0^x e^{t^A}dt\Big)$$

[1]Further reading on Runga-Kutta techniques reveal that many versions exist. However, Runga-Kutta of Order 4 is often referred to as classical Runga-Kutta as it was the first version of this method published.

$$
\begin{aligned}
&= \Big(\int_0^x e^{t^A} dt\Big) \frac{d}{dx} e^{-x^A} + e^{-x^A} \frac{d}{dx} \int_0^x e^{t^A} dt \\
&= -Ax^{A-1} e^{-x^A} \int_0^x e^{t^A} dt + e^{-x^A} e^{x^A} \\
&= -Ax^{A-1} y + e^0 \\
&= 1 - Ax^{A-1} y
\end{aligned}
$$

which was what we wanted. All that remains is to show that the initial condition is satisfied.

$$
\begin{aligned}
y(0) &= e^{-0^A} \int_0^0 e^{t^A} dt \\
&= \int_0^0 e^{t^A} dt \\
&= 0\,,
\end{aligned}
$$

which again is what we wanted. For our numerical evaulations, we will set $A = 3.6$, and use the aforementioned methods of solving differential equations. For comparison, we also present the exact results. Though we only provide these results once, we will supply error for every iteration of these methods. In order to find the exact results, we use the following *Mathematica* code:

```
exct = Table[ Re[SetPrecision[Exp[-t^A] Integrate[Exp[u^A], {u, 0, t}], 18]],
{t, 0, 5, 1}];
exct
```

And we reproduce the list that is generated below:

	Exact Value
$y(0)$	0
$y(1)$	0.4766633723095992
$y(2)$	$4.9072509768637800\times10^{-2}$
$y(3)$	$1.6194099094281732\times10^{-2}$
$y(4)$	$7.5944245802513962\times10^{-3}$
$y(5)$	$4.2396971232010645\times10^{-3}$

Table 5: Exact values calculated using *Mathematica*

However, we are still left to find our actual numerical solutions. In order to use Euler's Method, we use the following code in *Mathematica*, where we also keep the exact solutions for before to generate the error:

```
ClearAll[x, y]; Clear[f]
A = 3.6; f[a_, b_] := 1 - A*a^(A-1)*b;
ndiv = 10;
h = 5/ndiv; mult = ndiv/5; x[0] = 0; y[0] = 0;
x[n_] := x[n] = x[n - 1] + h; y[n_] := y[n];
```

```
i = 1;
Do[y[i] = y[i - 1] + h f[x[i - 1], y[i - 1]], i, ndiv];
app = Table[SetPrecision[y[mult n], 16], n, 0, 5];
exct = Table[ Re[SetPrecision[Exp[-t^A] Integrate[Exp[u^A], {u, 0, t}], 18]],
{t, 0, 5, 1}];
err = Abs[app - exct];
```

We change the number of divisions (`ndiv`) to also be 100 and 1000, and provide the data for both the numerical solution as well as the error from the exact results using Euler's Method.

	Number of Subdivisions					
	10		100		1000	
	Approx.	Error	Approx.	Error	Approx.	Error
$y(0)$	0	0	0	0	0	0
$y(1)$	0.8515553600380494	3.7489×10^{-1}	0.5012871769768208	2.4624×10^{-2}	0.4790085623774049	2.3452×10^{-3}
$y(2)$	1.254967504457188	1.2059×10^{0}	4.885489402901744$\times10^{-2}$	2.1762×10^{-4}	4904794635479909$\times10^{-2}$	2.4563×10^{-5}
$y(3)$	2.213395150571904$\times10^{2}$	2.2132×10^{2}	1.618691407184251$\times10^{-2}$	7.1850×10^{-6}	1.61933610467721173$\times10^{-2}$	7.3805×10^{-7}
$y(4)$	3.070306812577003$\times10^{5}$	3.07030×10^{5}	1.578958449996877$\times10^{-3}$	6.0155×10^{-3}	7.59433802306463783$\times10^{-3}$	8.6557×10^{-8}
$y(5)$	1.778123528444048$\times10^{9}$	1.7781×10^{9}	$-4.115525673436650\times10^{15}$	4.1155×10^{15}	4.239680116906033$\times10^{-3}$	1.7006×10^{-8}

Table 6: Euler's Method results using *Mathematica* for our given differential equation

We will now repeat this process but for the Runga-Kutta Method of Order 2. In order to do this, we must replace the `Do[ ]` control structure from before with the following code:

```
 Do[k1 = f[x[i - 1], y[i - 1]];
k2 = f[x[i - 1] + h, y[i - 1] + h*k1];
y[i] = y[i - 1] + h (k1 + k2)/2, i, ndiv];
```

And the results from these iterations are presented below:

	Number of Subdivisions					
	10		100		1000	
	Approx.	Error	Approx.	Error	Approx.	Error
$y(0)$	0	0	0	0	0	0
$y(1)$	0.1431412995333095	3.3352×10^{-1}	0.4753855048901734	1.2779×10^{-3}	0.4766520903514587	1.1282×10^{-5}
$y(2)$	-12.40601419690317	1.2455×10^{1}	0.04934768406482726	2.7517×10^{-4}	4.907373571110158$\times10^{-2}$	1.2259×10^{-6}
$y(3)$	-322386.6272773682	3.22390×10^{4}	0.07380599330110689	5.7612×10^{-2}	1.619423356476438$\times10^{-2}$	1.3447×10^{-7}
$y(4)$	$-3.337375337313056\times10^{11}$	3.3374×10^{11}	5.580075172804100$\times10^{15}$	5.5801×10^{15}	7.594467170507162$\times10^{-3}$	4.2590×10^{-8}
$y(5)$	$-5.033985433526746\times10^{18}$	5.0340×10^{18}	6.734937289063125$\times10^{45}$	6.7349×10^{45}	4.239721635235913$\times10^{-3}$	2.4512×10^{-8}

Table 7: Runga-Kutta Method of Order 2 results using *Mathematica* for our given differential equation

Finally, we turn to the Runga-Kutta Method of Order 4. In order to make this transition, we change the loop stsructure from our previous program to be:

```
 Do[k1 = f[x[i - 1], y[i - 1]];
k2 = f[x[i - 1] + h/2, y[i - 1] + h*k1/2];
```

```
k3 = f[x[i - 1] + h/2, y[i - 1] + h*k2/2];
k4 = f[x[i - 1] + h, y[i - 1] + h*k3];
y[i] = y[i - 1] + h (k1 + 2 k2 + 2 k3 + k4)/6, i, ndiv];
```

And the results are presented below:

	Number of Subdivisions					
	10		100		1000	
	Approx.	Error	Approx.	Error	Approx.	Error
$y(0)$	0	0	0	0	0	0
$y(1)$	0.4576365068812699	1.9027×10^{-2}	0.4766623811885676	9.9112×10^{-7}	0.4766633722288154	8.0784×10^{-11}
$y(2)$	-3.143101667172594	3.1922×10^{0}	0.04908190296446983	9.3932×10^{-6}	$4.907251032789623\times10^{-2}$	5.5926×10^{-10}
$y(3)$	$-6.423356373481491\times10^{7}$	6.4234×10^{7}	0.01632546765856335	1.3137×10^{-4}	$1.619409962705142\times10^{-2}$	5.3277×10^{-10}
$y(4)$	$-1.972271792284101\times10^{18}$	1.9723×10^{18}	$7.915062027069412\times10^{15}$	7.9151×10^{15}	$7.594425289342698\times10^{-3}$	7.0909×10^{-10}
$y(5)$	$-1.268165421884272\times10^{31}$	1.2682×10^{31}	$8.409532118999917\times10^{60}$	8.4095×10^{60}	$4.239698141101379\times10^{-3}$	1.0179×10^{-9}

Table 8: Runga-Kutta Method of Order 4 results using *Mathematica* for our given differential equation

A general assumption seems to be that the farther x_i is from the initial x_0, the larger the error in the computation of the estimate of $y(x_i)$. In most cases, this seems to be true. Looking at all of our computations, the error increases in magnitude. However, this does not prove to be the case for Euler's Method or Runga-Kutta Method of Order 2 when using 1000 subdivisions. This may be due to the fact that the large number of subdivisions has led to a convergence to the analytical result instead of compounding the error as the distance increases.

Another general assumption seems to be that, fixing the number of subdivisions, the estimate at a given $y(x_i)$ is best for the Runga-Kutta Method of Order 4, then next best for the Runga-Kutta Method of Order 2, and worst for Eulers Method. This seems to be the case, so long as x is close to the initial $x = 0$. It seems that if the initial point farther away, in many cases, the error can grow rapidly for methods that would be predicted to result in greater accuracy.

Next, we will look at the behavior of the errors for each algorithm given its order. We know that Euler's method is of order 1, and the Runga-Kutta methods are of order 2 and 4. This means that whereas Euler's Method evaluates the function at the two endpoints on a subinterval and assumes a constant derivative in between the points, the Runga-Kutta method of order n computes the derivative at n points to generate n secants connecting the interpolated function on the divided subinterval. Thus, if we start with a certain number of subdivisions with an order n algorithm, then if we multiply our number of subdivisions by k, we expect the error to go down by a factor of k^n. Take, for example, the evaluation of $y(1)$ for the Runga-Kutta Method of Order 4. As the number of subdivisions is increased by factors of 10, the error goes down from 10^{-2} to 10^{-7} to 10^{-11}, which is approximately a decrease of 10^4 each iteration. For Runga-Kutta of Order 2 for $y(1)$ we have error on the order of 10^{-1} to 10^{-3} to 10^{-5}, a factor of 100 each time, which is expected. Finally, for Euler's Method, we have 10^{-1} to 10^{-2} to 10^{-3}, which decreases by the predicted factor of 10 for each increase of number of subdivisions by a factor of 10.

Unfortunately, in a realistic situation you cannot obtain an error bound and you do not have an exact solution to the initial value problem. If our goal is to obtain an estimate of $y(x)$ to 10 decimal places, then we can use repeated numerical estimations to give us an indication on when we are converging to the correct answer. For example, begin with a small number of subdivisions, and generate an estimate for $y(x)$. Then, increase the number of subdivisions by a factor of, perhaps, 10, and continue doing this until the estimates first 10 decimals stop oscillating or jumping between values. In order to get this convergence quicker, if run time on computers is not an issue, we would recommend using a technique of higher order, such as Runga-Kutta of Order 4. ♦

This example, complemented by the implication of the order of a method for solving differential equations reveals some properties of the error bound for the three methods. First, in the case of Euler's Method, we note it is an algorithm of order 1. This means that if we double the number of subdivisions, then h is cut in half. Thus, the error should be cut approximately in half as well. This was witnessed in the example as we increased the number of subdivisions by a factor of 10, the error went down by a factor of 10.

Additionally, as was noted in the example, the behavior of the error with respect to the number of subdivisions in the Runga-Kutta Methods extends beyond just the function in example 5. For Runga-Kutta of order 2, if the number of subdivisions goes up by a factor of n, then the error goes down by a factor of n^2, and in Runga-Kutta of order 4, if the number of subdivisions goes up by a factor of n, then the error goes down by a factor of n^4.

5 Conclusion

Throughout the evolution of mathematics, the vast expanse of techniques for producing analytical solutions has been outrun by the complexity of problems that need to be solved. To remedy this discrepancy, numerical techniques are of the utmost salience to move forwards in fields beyond mathematics. The numerical methods presented in this paper, whether they were for solving fixed point problems or differential equations, have proved beyond useful in understanding behaviors of systems that current mathematics cannot yet solve exactly.

References

[1] S. Strogatz, "Nonlinear Dynamics and Chaos: with Applications to Physics, Biology, Chemistry, and Engineering." Boulder, CO: Westview Press, 2015. 33. Print.

[2] W. Press et al., "Numerical Recipes in C: The Art of Scientific Computing," 2nd Ed. New York, NY: Cambridge University Press, 1992. 147. Print.