Classifying Solver Viability Given an Arbitrary Function or Differential Equation

Erik James Knee

January 2026

1 Abstract

Differential equations and functions historically have been solved analytically to find existing roots, or solutions given initial or boundary conditions. However, Some equations do not have such methods of doing them by hand, which thus requires utilization of numerical methods to find solutions at predefined points of interest. These methods can vary in utilization based on the particular problem, where there are identifiable characteristics of equations that determine whether a certain method will actually converge to a solution, or will converge inefficiently compared to other methods. This projects aim is to record a methodology without needing excessive background knowledge in determining when common solvers will crash by analyzing the particular function or differential equation, so the reader can build intuition on what cannot work when solving a problem numerically. That way, in future research projects the reader is able to minimize failed attempts before arriving at a solution.

2 Introduction

.

To begin our study on when certain kinds of numerical methods fail, we would first like to introduce examples to the reader.

Newton's Method (Root Finding)

Newton's Method is an iterative numerical technique used to approximate solutions of an equation

$$f(x) = 0,$$

where the function f is differentiable and its derivative is known.

Starting from an initial guess x_0 that is reasonably close to a true root, Newton's Method constructs a sequence $\{x_n\}$ by repeatedly using the tangent line to f at the current approximation. The point where this tangent line intersects the x-axis is taken as the next approximation.

The iteration formula is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Geometrically, this formula comes from the linear approximation of f at x_n:

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n).$$

Setting this approximation equal to zero and solving for x yields the Newton update formula.

When the initial guess is sufficiently close to the actual root and $f'(x) \neq 0$ near the root, Newton's Method converges rapidly, often doubling the number of correct digits at each step. However, poor initial guesses or points where the derivative is small or undefined can lead to slow convergence or divergence[1, section 4.9].

The Bisection Method

The bisection method is a root-finding algorithm for solving a nonlinear equation

$$f(x) = 0$$

when the function f is continuous on a closed interval $[a, b]$ and satisfies

$$f(a)\, f(b) < 0.$$

This condition guarantees, by the Intermediate Value Theorem, that at least one root lies in the interval[2, pg. 56].

The method proceeds by repeatedly halving the interval. At each step, the midpoint

$$m = \frac{a + b}{2}$$

is computed and the sign of $f(m)$ is examined. If $f(a)\, f(m) < 0$, then the root lies in the subinterval $[a, m]$; otherwise, it lies in $[m, b]$. The interval containing the root is therefore reduced by a factor of two at each iteration.

This process is continued until the interval length becomes smaller than a prescribed tolerance or until the function value at the midpoint is sufficiently close to zero. The midpoint of the final interval is then taken as an approximation to the root.

The bisection method is guaranteed to converge as long as the initial sign-change condition holds. Its convergence is linear, and although it is slower than methods such as Newton's method, it is highly reliable and insensitive to the shape of the function. For this reason, it is often used as a starting method or as a benchmark for more rapidly convergent algorithms.

Finite Difference Method

The goal of the finite difference method is to approximate solutions to differential equations. That is, we aim to find a function (or a discrete approximation of it) that satisfies the relationships between its derivatives over a given spatial and/or temporal domain, along with prescribed boundary conditions. Since exact, analytic solutions are rarely available, numerical methods are required.

The finite difference method works by replacing derivatives with algebraic approximations using values of the function at discrete points. This converts the differential equation into a finite system of algebraic equations that can be solved on a computer.

Before applying this to differential equations, it is helpful to consider the simpler problem of approximating derivatives of a known function using only its values at nearby points. This introduces fundamental concepts such as the order of accuracy of an approximation.

Let $u(x)$ be a smooth function (i.e., differentiable several times), and consider a point x_n. A simple finite difference approximation for the first derivative at x_n is the forward difference:

$$u'(x_n) \approx \frac{u(x_n + h) - u(x_n)}{h},$$

where h is a small spacing between points. This is a one-sided approximation because it uses only points $x \geq x_n$. Similarly, the backward difference uses points $x \leq x_n$:

$$u'(x_n) \approx \frac{u(x_n) - u(x_n - h)}{h}.$$

A more accurate alternative is the central difference approximation, which uses points on both sides of x_n:

$$u'(x_n) \approx \frac{u(x_n + h) - u(x_n - h)}{2h}.$$

The central difference is a two-sided approximation and generally has a higher order of accuracy than the forward or backward difference. Specifically, for sufficiently smooth functions, the truncation error of the central difference is proportional to h^2, whereas forward and backward differences have errors proportional to h. This improved accuracy makes the central difference particularly useful in numerical schemes for differential equations, as it often allows for larger step sizes without significantly sacrificing precision.

Similarly, second derivatives can also be approximated using central differences. For instance, the second derivative at x_n can be written as:

$$u''(x_n) \approx \frac{u(x_n + h) - 2u(x_n) + u(x_n - h)}{h^2}.$$

By systematically replacing derivatives with such finite difference approximations, a differential equation defined over a spatial or temporal domain can

be converted into a discrete system of algebraic equations on a grid, which can then be solved numerically using linear algebra techniques[4, pgs. 3-11].

Fixed Point Iteration

Paraphrasing Atkinson's book [2, pg. 76], fixed-point iteration is a method commonly utilized for finding a root of the nonlinear equation

$$f(x) = 0 \tag{1}$$

by rewriting it in the form

$$x = g(x), \tag{2}$$

where g is a function chosen such that the fixed point x^* of g corresponds to a solution of the equation $f(x) = 0$. We then write the iteration as

$$x_{k+1} = g(x_k), \quad k = 0, 1, 2, \ldots, \tag{3}$$

where convergence of the method is guaranteed if g is continuous on an interval $[a, b]$ (containing the root) and satisfies

$$|g'(x)| \leq k < 1 \quad \text{for all } x \in [a, b]. \tag{4}$$

Under these conditions, the sequence $\{x_k\}$ converges to x^* for decent initial guesses $x_0 \in [a, b]$. It can be shown that most times the convergence of this method is linear, and convergence is faster when $|g'(x^*)|$ is smaller.

Additionally, there are practical strategies for choosing $g(x)$. One approach is to algebraically rearrange $f(x) = 0$ into $x = g(x)$. Another approach is to use a relaxation or damping method:

$$g(x) = x - \lambda f(x), \tag{5}$$

where λ is chosen to ensure $|g'(x^*)| < 1$. It is important to note that the success of the method depends critically on the choice of $g(x)$ and the proximity of the initial guess x_0 to the root.

3 Methodology

To gain an understanding of when different methods fail, we will first solve a typical question where the method is successful, and then show a contrasting example where the method may not be used.

3.1 Bisection Method

While continuing to reference from Atkinson's book[2, pg. 56], A typical problem where the bisection method works well is for the function

$$f(x) = x^3,$$

where we wish to find the root x = 0 on the interval [-1,2]. We choose this interval because using an interval such as [-1,1] is symmetric to where the root is on our function, hence it would only take one iteration which is uninteresting. This would be the same as choosing a more advanced polynomial in regards to the number of iterations, so we do this for convenience. Hence, by using the bisection method with a given tolerance of $\epsilon = 10^{-3}$, we summarize:

Table 1: Bisection Method Convergence

Iteration (n)	a_n	b_n	x_n (midpoint)	Error $\lvert x_n - x_{n-1}\rvert$
1	-1	2	0.5	–
2	-1	0.5	-0.25	0.75
3	-0.25	0.5	0.125	0.375
4	-0.25	0.125	-0.0625	0.1875
5	-0.0625	0.125	0.03125	0.09375
6	-0.0625	0.03125	-0.015625	0.046875
7	-0.015625	0.03125	0.0078125	0.0234375
8	-0.015625	0.0078125	-0.00390625	0.01171875
9	-0.00390625	0.0078125	0.001953125	0.005859375
10	-0.00390625	0.001953125	-0.0009765625	0.0029296875

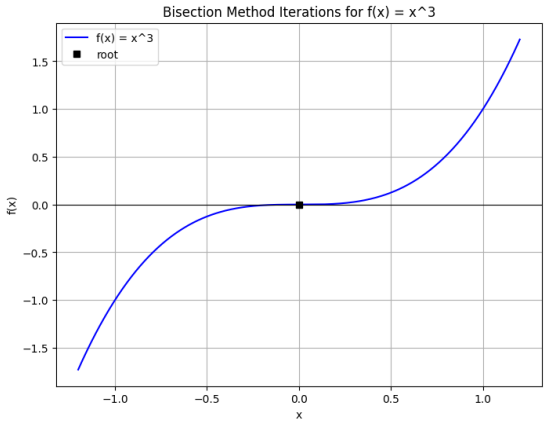

Figure 1: Function and desired root

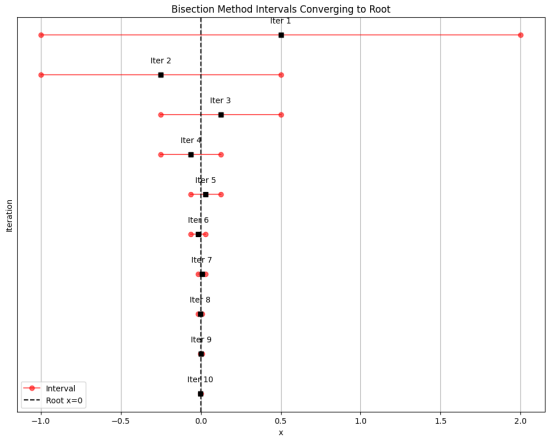

Figure 2: Iteration progress

An overly simple way we can get this algorithm to fail is to violate its necessary condition of the function needing to satisfy $f(a)f(b) < 0$. For a function that is positive everywhere besides the root such as $f(x) = x^2$, the function only touches the x-axis rather than crossing it. This means we cannot establish the 'bracketing' required to decrease the size of the interval in which the solution can be found. If we do try anyway, this is the result:

Table 2: Bisection Method Convergence

Iteration (n)	a_n	b_n	c_n (midpoint)	Error $\lvert c_n - c_{n-1} \rvert$
1	-1	2	0.5	–
2	0.5	2	1.25	0.75
3	1.25	2	1.625	0.375
4	1.625	2	1.8125	0.1875
5	1.8125	2	1.90625	0.09375
6	1.90625	2	1.953125	0.046875
7	1.953125	2	1.9765625	0.0234375
8	1.9765625	2	1.98828125	0.01171875
9	1.98828125	2	1.994140625	0.005859375
10	1.994140625	2	1.9970703125	0.0029296875

As we can see, the method incorrectly finds the solution to be $x = 2$, because the midpoint can never switch signs to 'bracket' itself into the correct interval for which the solution exists.

To illustrate a more non-obvious example, we have to be more creative. The bisection algorithm is very robust in that it is guaranteed to converge given the necessary conditions for convergence is fulfilled. However, in practice, many problems in science do not have as strong of properties that the bisection method

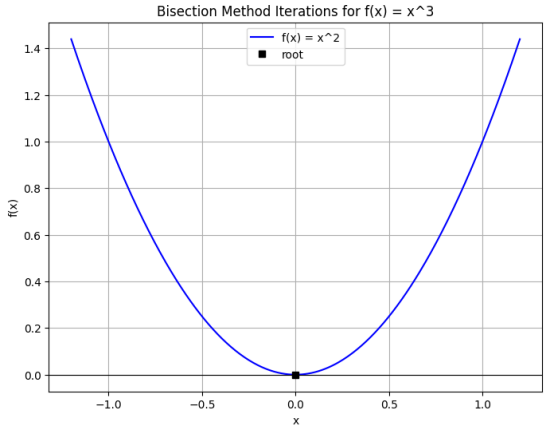

Figure 3: Function and desired root

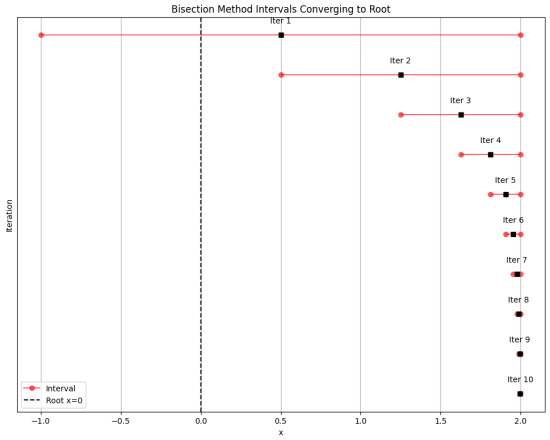

Figure 4: Function and desired root

requires of being continuous everywhere on a closed interval, so it is important to consider the functions in science that could have point discontinuities that stops the method from 'bracketing' like before.

Another thing to consider about bisection is its efficiency. The bisection method is guaranteed linear convergence if the problem fulfills its requirements, but can be out-competed by Newton's method from quadratic convergence. For example, if we look at the same function of $f(x) = x^3$ only this time we are interested in a much more global interval such as $[-10^{50}, 10^{49}]$, we see that Newton's method reaches the required solution much quicker, even with a really poor initial guess of $x = 10^{12}$ and same tolerance of 1e-3:

Table 3: Bisection Method Convergence (Selected Iterations)

Iteration (n)	a_n	b_n	c_n (midpoint)	Error $\lvert c_n - c_{n-1}\rvert$
1	-1.0×10^{50}	1.0×10^{49}	-4.5×10^{49}	–
2	-4.5×10^{49}	1.0×10^{49}	-1.75×10^{49}	2.75×10^{49}
3	-1.75×10^{49}	1.0×10^{49}	-3.75×10^{48}	1.375×10^{49}
4	-3.75×10^{48}	1.0×10^{49}	3.125×10^{48}	6.875×10^{48}
5	-3.75×10^{48}	3.125×10^{48}	-3.125×10^{47}	3.4375×10^{48}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
~ 50	-4.20×10^{0}	5.21×10^{0}	5.05×10^{-1}	~ 2.35
~ 60	-1.85	5.05×10^{-1}	-6.71×10^{-1}	~ 1.18
~ 70	-8.32×10^{-2}	5.05×10^{-1}	2.11×10^{-1}	$\sim 2.94 \times 10^{-1}$
~ 80	-9.66×10^{-3}	6.38×10^{-2}	2.71×10^{-2}	$\sim 3.68 \times 10^{-2}$
177	-4.76×10^{-4}	6.73×10^{-4}	9.87×10^{-5}	5.74×10^{-4}

Table 4: Newton's Method Convergence (Initial Guess $x_0 = 10^{12}$)

Iteration (n)	x_n	Error $\lvert x_n - x_{n-1}\rvert$
0	$1.0000000000 \times 10^{12}$	–
1	$6.6666666667 \times 10^{11}$	$3.3333333333 \times 10^{11}$
2	$4.4444444444 \times 10^{11}$	$2.2222222223 \times 10^{11}$
3	$2.9629629630 \times 10^{11}$	$1.4814814814 \times 10^{11}$
4	$1.9753086419 \times 10^{11}$	$9.8765432108 \times 10^{10}$
5	$1.3168724280 \times 10^{11}$	$6.5843621395 \times 10^{10}$
$\vdots$	$\vdots$	$\vdots$
20	$3.0072865982 \times 10^{8}$	$1.5036421141 \times 10^{8}$
30	$7.8226425763 \times 10^{6}$	$3.9133212881 \times 10^{6}$
40	$9.0437726838 \times 10^{4}$	$4.5218993419 \times 10^{4}$
50	$1.5683285455 \times 10^{3}$	$7.8416427255 \times 10^{2}$
60	$2.7197216389 \times 10^{1}$	$1.3598608064 \times 10^{1}$
70	4.0795824584	2.0397912290
75	1.5917895280	$7.9589476402 \times 10^{-1}$
80	$1.3974558271 \times 10^{-1}$	$6.9872791354 \times 10^{-2}$
84	$1.6156014688 \times 10^{-3}$	$8.0780073454 \times 10^{-4}$

3.2 Newton's Method

Now that we have contrasted how the bisection method can sometimes be less useful than Newton's method, it is natural that we next summarize why Newton's method sometimes is not the best choice either. We continue to use the

text Calculus Volume 1 as reference for this section [1, section 4.9]. Like mentioned in the introduction, Newton's method heavily relies on the initial guess to be close enough to the solution you are looking for, and that there is a baseline level of 'smoothness' in that the function values do not change abruptly so that large fluctuations in derivatives do not lead to stability issues. Additionally, if the curve is too 'flat' around the local area of where your initial guess is, then the small derivatives can lead to divergence.

An example function we can use to illustrate this is the polynomial

$$g(x) = \arctan(x),$$

where there is a transition point that determines whether Newton's method will converge or not. According to Leonid Kovalev's website[3], if we choose our initial guess for Newton's method to have distance less than ~ 1.3917 from the root of $x = 0$, the method will converge, but if we choose any more than this transition then the method will diverge. As we can see from the figure, the transition point reflects intuitively of when the derivatives past it get too small to allow for sufficient descent, and leads to numerical instability. We also provide tables to demonstrate progress of Newton's method both inside and outside this transition ($x = 1.3$ and $x = 1.4$), to show what happens within the desired tolerance of 1e-3.

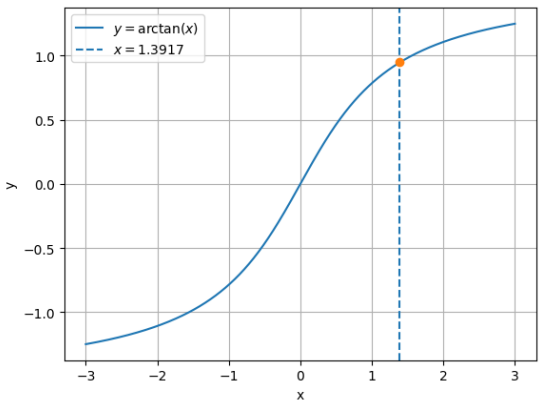

Figure 5: arctan(x) and the transition point

Table 5: Newton's Method Iteration History ($x_0 = 1.4$)

Iteration (n)	x_n	$f(x_n)$	$f'(x_n)$	x_{n+1}	Error $\lvert x_{n+1} - x_n \rvert$
1	1.40	0.95	0.34	-1.41	2.81
2	-1.41	-0.96	0.33	1.45	2.86
3	1.45	0.97	0.32	-1.55	3.00
4	-1.55	-0.998	0.29	1.85	3.40
5	1.85	1.08	0.23	-2.89	4.74
6	-2.89	-1.24	0.11	8.71	11.60
7	8.71	1.46	0.013	-103.25	112.00
8	-103.25	-1.56	0.000094	16540.56	16640.00
9	16540.56	1.57	0.000000004	-429721482.90	429721484.00

Table 6: Newton's Method Iteration History ($x_0 = 1.3$) with Scientific Notation for Small Values

Iteration (n)	x_n	$f(x_n)$	$f'(x_n)$	x_{n+1}	Error $\lvert x_{n+1} - x_n \rvert$
1	1.30	9.15e-01	3.72e-01	-1.16	2.46
2	-1.16	-8.60e-01	4.26e-01	0.86	2.02
3	0.86	7.10e-01	5.76e-01	-0.37	1.23
4	-0.37	-3.58e-01	8.77e-01	3.40e-02	0.41
5	3.40e-02	3.40e-02	9.99e-01	-2.62e-05	3.40e-02
6	-2.62e-05	-2.62e-05	1.00e+00	0.00e+00	2.62e-05

3.3 Finite Difference Method

Now that we have covered some root finding algorithms, we can discuss some fundamentals of finite difference methods failing when solving differential equations. We assume that the linear algebra techniques and methods for creating the discrete system of equations are familiar to the reader. Consider the 1D poisson equation

$$-u''(x) = f(x), \quad u(0) = u(1) = 0, \quad x \in (0, 1)$$

where

$$f(x) = \pi^2 \sin(\pi x)$$

and the exact solution is

$$u(x) = \sin(\pi x).$$

This function is well posed for the finite difference method, as the solution is smooth enough in that there are no rapid oscillations that would lead to instability issues. Hence, if we use central difference, we should only require

relatively small amounts of grid points to achieve a decent approximation to the real solution. This solution also has amplitude of order 1, meaning

$$||u||_\infty = \max_{x\in[0,1]} |u(x)| = 1,$$

which will mean that it is likely our relative and absolute error norms should be meaningful if small enough. Hence, by choosing a reasonable number of grid points, say $N = 50$, we get the approximate solution and the following information:

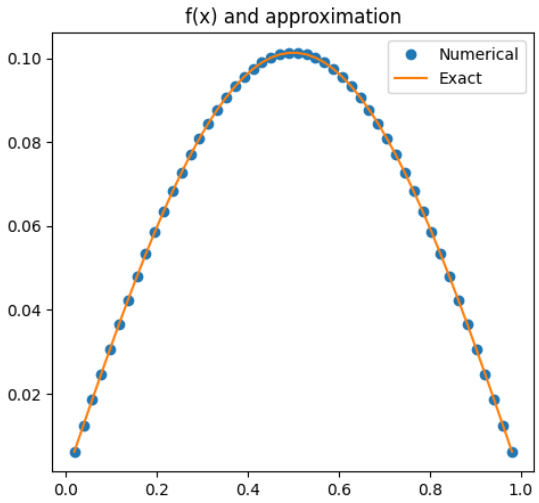

Figure 6: Finite Difference plot of stable function

Quantity	**Value**
Grid spacing (h)	1.96078×10^{-2}
Points per wavelength	102.00
Absolute max error	3.20298×10^{-5}
Relative max error	3.16272×10^{-4}
Relative L_2 error	3.16272×10^{-4}
Relative residual	9.63020×10^{-14}
Computation time (s)	0.230259

Table 7: Numerical error metrics and performance statistics for the good case

As we can see, the relative maximum and L_2 errors are both on the order of 10^{-4}, meaning we have achieved high accuracy and uniform error distribution across the domain. Additionally, our relative residual confirms that our discrete linear system is solved to near machine precision, and our grid spacing allows for sufficient points per wavelength to ensure we have sufficient resolution. Thus,

this results in an accurate reflection on second-order convergence of the finite difference method for smooth solutions. If we wish to consider a more difficult problem for regular finite-difference methods, we want to show how this sometimes is not the most efficient way to approach problems because we are working on a uniform mesh. From Kopteva and O'Riordan[5] we consider the 1D singularly perturbed convection-diffusion boundary value problem

$$\begin{aligned} -\varepsilon u''(x) + u'(x) &= 0, \qquad x \in (0,1), \\ u(0) &= 0, \\ u(1) &= 1, \\ 0 < \varepsilon \ &<< 1, \end{aligned}$$

with exact solution

$$u(x) = \frac{e^{(x-1)/\varepsilon} - e^{-1/\varepsilon}}{1 - e^{-1/\varepsilon}}, \qquad 0 \le x \le 1.$$

This problem has a right boundary layer at $x = 1$, which means in basic terminology that we are far from $x = 1$, the solution is almost linear and when close, the solution changes rapidly over a width of

$$\mathcal{O}(\varepsilon),$$

which can be intuitively understood since

$$u(x) \sim e^{(x-1)/\varepsilon}.$$

So, for finite difference method, we have

$$h = \frac{1}{N},$$

where if we wish to resolve the boundary layer imposed by this problem, we need

$$h << \varepsilon \implies N >> \varepsilon^{-1}.$$

Given the parameters

$$\begin{aligned} \varepsilon &= 10^{-3}, \\ N &= 128, \\ h &\sim 7.8 \times 10^{-3}, \end{aligned}$$

we observe that the layer can be thinner than the mesh spacing. Since this is the case, we expect that the finite-difference method that we used before samples this layer samples the layer with too few points that are required for the approximation to be meaningful, and hence the numerical solution should show a shift in the layer as the error depends specifically on ε. So instead, it is proposed that in order to get a better approximation while using less nodes that we use a Shishkin mesh. To explain what is happening with Shishkin meshes differently and more basically without going into in depth analysis, the mesh

we are using is piecewise-uniform rather than strictly uniform, meaning we use more coarse spacing where the solution is smooth, and finer spacing where the boundary layer is. Similar from Kopteva and O'Riordan's paper[5], the mesh is tuned to the decay rate of the exact solution as

$$\sigma = \min(\frac{1}{2}, 2\varepsilon \ln(N))$$

which in turn places a transition point at

$$x = 1 - \sigma \sim 1 - 2\varepsilon \ln(N),$$

so that the exponential layer satisfies

$$e^{-(\sigma/\varepsilon)} = e^{-2 \ln N} = N^{-2}.$$

This is important, as when we finally reach the transition point, the layer will have decayed to $\mathcal{O}(N^{-2})$, and be smooth enough once past. So, we split the mesh into a coarse region of $[0, 1 - \sigma]$, where

$$h \sim \frac{1}{N}$$

, and a fine region of $[1 - \sigma, 1]$, where

$$h \sim \frac{\varepsilon \ln N}{N},$$

and thus, $h << \varepsilon$ within the layer, being able to resolve it. So, by using $N = 128$, $\varepsilon = 10^{-3}$, we see the following:

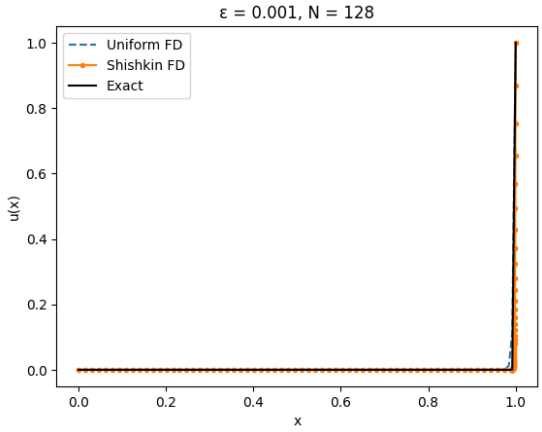

Figure 7: Shishkin mesh vs. uniform mesh

Table 8: Maximum norm error comparison between uniform and Shishkin meshes

Mesh type	**Maximum error**
Uniform mesh	0.1130705321
Shishkin mesh	0.0263248634

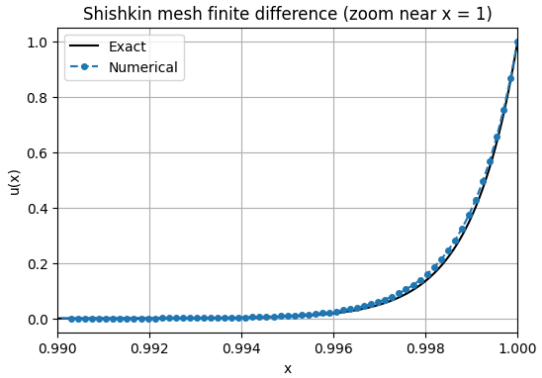

Figure 8: Shishkin mesh close to transition point

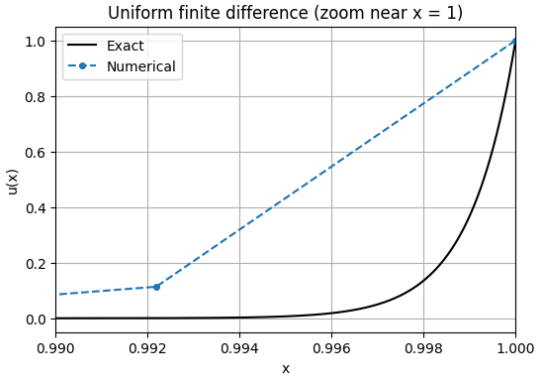

Figure 9: FD uniform mesh close to transition point

As we can see, the Shishkin mesh does better since we are particularly focusing on placing more nodes where they matter in the problem, unlike regular finite-difference methods where each section of the equation receives the same amount of nodes uniformly. Since the equation that we are trying to approximate has this 'skew' where only one small, condensed region has rapid changes while the rest stays relatively similar, it makes sense that we try harder to approximate the layer that can be more problematic rather than give it the same treatment as everywhere else, which explains why our max error for Shishkin is much smaller. So, we can conclude that finite difference alone may not be the best idea if we have high variance in the approximate solution in a very localized region of the problem, as the uniform mesh will not be fine enough in some regions to resolve properly, requiring excess nodes for similar convergence and thus computational waste compared to using a Shishkin mesh instead.

3.4 Fixed Point Iteration

To illustrate where the fixed point iteration algorithm works well, consider the example

$$x = \cos(x).$$

Then, we have that

$$|g'(x)| = |\sin(x)| < 1,$$

So this method must converge for any initial guess $x \in [0, 1]$, based on the theory introduced earlier. Thus, by arbitrarily choosing our initial guess to be $x^* = 1$, we have the following results documented below.

We understand that convergence for this problem is particularly fast due to the first derivative of our $g(x)$ evaluated at x^* is sufficiently less than 1, or equivalently

$$|g'(x^*)| \sim |\sin(1)| \sim 0.841 < 1.$$

Iteration	x_n
0	1.0000000000000000
1	0.5403023058681398
2	0.8575532158463934
3	0.6542897904977791
4	0.7934803587425656
⋮	⋮
31	0.7390838469650002
32	0.7390859996481299
33	0.7390845495752126
34	0.7390855263619245

Table 9: Fixed-point iteration progress for $x = \cos(x)$, showing first few and last few iterations.

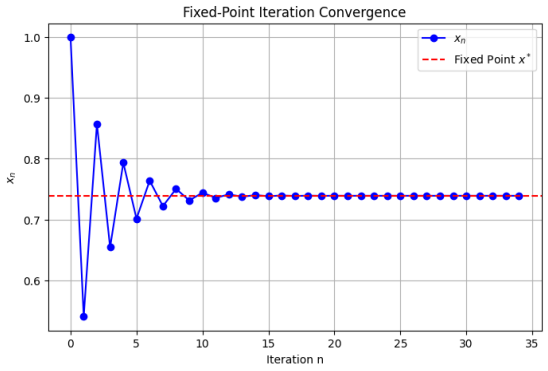

Figure 10: Fixed point plot of stable function

This is important, as this is the determining factor on whether we will achieve sufficient decrease. So, by construction we can create an example problem such that this first derivative of g is very close to one, and observe that the closer it is the more iterations are required. Take for example the problem

$$x = 1 - \epsilon + e^x,$$

for some arbitrarily small $\epsilon > 0$. If we look at the derivative, we have

$$|g'(x)| = |-e^x| = e^x,$$

which for convergence requires any $x < 0$. So, with initial guess of $x^* = -0.002$, we have results of very slow convergence since

$$|g'(-0.002)| = e^{-0.002} \sim 0.998 < 1,$$

which is very close to violating the requirements of the method.

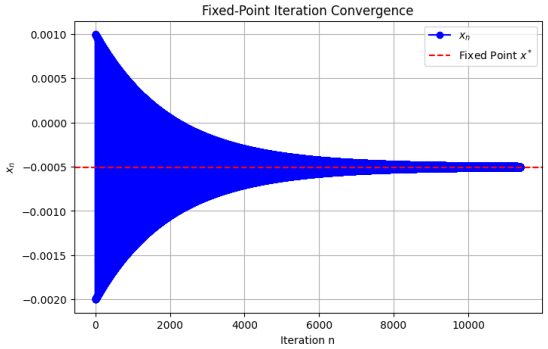

Figure 11: Fixed point plot of unstable function

Iteration n	x_n
0	-0.002000
1	-0.999002
2	-0.368880
3	-0.690234
4	-0.500870
⋮	⋮
11406	-0.00050506

Table 10: Fixed-point iteration values for $x_{n+1} = 1 - \varepsilon - e^{x_n}$ with $\varepsilon = 10^{-3}$. The method converges after 11407 iterations with tolerance 10^{-5}.

As we can see, even with an initial guess relatively close to the exact solution, we still have relatively slow convergence. This is because the exponential function acts inversely proportionally to the 'quality' of the guess as our epsilon gets smaller, meaning that we must get closer to the solution at a nonlinear rate. For our guess of $x^* = -0.002$ at $\epsilon = 10^{-3}$, we have after one iteration

$$g(-0.002) = 0.999 - e^{-0.002} = 0.0009980013,$$

which has relative error of

$$\frac{|0.0009980013 - 0.00050506|}{|-0.00050506|} \sim 0.9760 = 97.6\%,$$

which is extremely high comparatively to say, $\epsilon = 10^{-2}$ with the same inital guess (exact found being -0.005006255):

$$g(-0.002) = 0.99 - e^{-0.002} = -0.00800199866$$

relative error

$$\frac{|0.00800199866 - 0.005006255|}{|0.005006255|} \sim 0.5968 = 59.68\%.$$

Hence, when using this method it is important that we must have an arbitrary guess very close to the actual root of the equation. By construction, we were able to tell the exact solution easily for this example, but for more advanced functions seen in science, it is important to note that we cannot always know where the root may lie, hence this method may not be useful if we cannot find a close enough neighborhood for our initial guess.

4 Results

To summarize our discussion, it is convenient to create a short list of things the reader can keep in mind whenever they need to use a numerical solver. That way, in practice they can quickly recall the more obvious conditions for convergence without having to to have a global understanding of its utility.

4.1 Newton's Method

- Newton's method should be used for problems that require quadratic convergence rates, as several other methods only work up to linear rates which make them unusable for some problems.
- Newton's method cannot be used well when you cannot tell what a good neighborhood is to place your initial guess, or derivatives are too small leading to instability issues (i.e, neighborhoods that appear close to parallel to the x-axis).

4.2 Bisection Method

- Bisection method is best utilized when all of its conditions for convergence are satisfied and linear convergence is good enough for the problem you are working on, as you are guaranteed to find a solution.
- Bisection does not work well for many problems as continuity and sign change over an interval are too strict of conditions. Additionally, linear convergence is sometimes not good enough If the interval you are 'bracketing' over is too large.

4.3 Finite Difference Method

- When the solution you wish to approximate is smooth, finite difference works well as it achieves second-order convergence and uniform error distribution.
- If the solution has a localized region where there is a rapid change in value, finite difference on a uniform mesh will be unable to resolve the problematic area efficiently as increasing total nodes will also increase the nodes utilized on parts of the solution already resolved. Using a Shishkin

mesh instead would efficiently distribute more nodes to the parts of the function approximation that need them.

4.4 Fixed Point Iteration

- Similar to bisection method, most times when the conditions are met the method converges linearly, and can get faster the smaller the derivative of the chosen function evaluated at the guess is from 1. It is also guaranteed convergence when the conditions are satisfied, hence it can sometimes be better than bisection if the guess is close enough.
- When our initial guess is too far from the exact solution, like Newton's method efficiency decreases drastically. Additionally, we have strict conditions of the derivative of the chosen function $g(x)$ having to exist, which can more restrictive than bisection.

5 Conclusion and Future Work

In conclusion, this study introduces the reader to some commonly used numerical methods used to find roots of functions or to approximate solutions to differential equations. Through which, we summarized the basics of how each are utilized in several straightforward functions that would help build intuition on where to use them, and small details that can be easily checked before attempting the problem to understand more if they will work or not. The strengths of this report is that we were able to give a summary on both good examples and bad examples of each solvers utilization with minimal analysis background. Weaknesses would be primarily be due to time constraint, as there are some additional error plots and other solvers that could have been included. Additionally, for this study we only focused on single variable equations, which could be expanded to multiple dimensions.

Future work would be to correct the weaknesses of this report by including even more solvers used commonly in science, and more information about how fast error decreases with each iteration of the root-finding algorithms other than fixed point iteration. additionally, this project could be expanded on by assuming that the reader gradually increased their knowledge of numerical analysis, so we can connect them to the more informal techniques seen in this paper (for example, a more concrete definition of derivatives becoming too small for Newton's method to converge rather than visual understanding from the plot).

6 Declaration of AI

Artificial intelligence was used in this paper to aid in the construction of code and literature review, as well as checking for grammar and spelling. This was to efficiently find online sources and fix code errors, so more time could be spent

working on results. All content generated by AI was checked thoroughly for accuracy, and was always backed up with the original resource the content was generated from when possible.

References

[1] Edwin Herman and Gilbert Strang, *Calculus Volume 1*, OpenStax, Houston, TX, online edition, 2016. Updated 2026. Section 4.9: Newton's Method. Available at `https://openstax.org/details/books/calculus-volume-1`. ISBN 978-1-938168-02-4.

[2] K. E. Atkinson, *An Introduction to Numerical Analysis*, 2nd ed., John Wiley & Sons, New York, 1989.

[3] L. Kovalev, *Potential Issues with Newton's Method*, Numerical Methods with Programming, available at `https://drlvk.github.io/nm/section-newton-method-issues.html`, accessed January 24, 2026.

[4] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2007.

[5] N. Kopteva and E. O'Riordan, *Shishkin meshes in the numerical solution of singularly perturbed differential equations*, Int. J. Numer. Anal. Modeling, **7**(3) (2010), 393–415. :contentReference[oaicite:0]index=0