High Order Finite Differences and Grid Spacings

William Taylor, Niall Oswald, Moe Okawara, Yaofeng Lan,
and Nicholas Matheou

Supervisor: Dr Shahid Mughal

June 22, 2022

Acknowledgements

We would like to offer our sincere thanks to our supervisor Dr Shahid Mughal, without whose guidance and support this project would not have been possible. The direction towards new approaches and areas of interest has been invaluable when exploring this unfamiliar area of numerical methods.

Abstract

Finite differences are a fundamental technique in the world of solving ordinary and partial differential equations numerically. High order finite difference schemes can provide efficient methods for computing solutions to high degrees of accuracy, limiting the need for larger grids. The choice of grid in a finite difference scheme is often crucial in the accuracy of the solutions, and certain problems can, at times, require a specific grid spacing to achieve accurate results.

In this paper we introduce finite differences, looking at various choices of grids, evaluating the accuracy and relative speed of possible approaches, and how the choice of a specific grid may influenced by the problem at hand.

Contents

Chapter 1

Introduction

William Taylor, Niall Oswald, Moe Okawara, Yaofeng Lan, and Nicholas Matheou

We start in Chapter 2 with uniform grids, often the simplest choice for a grid spacing, yet can work well for many problems. We put this grid spacing to use in Chapter 3 in solving the Bratu problem, where despite the difficulty of the problem, we find that uniform spacing can actually yield highly accurate numerical solutions even for more challenging problems. We use the standard finite difference method as outlined in Chapter 2, as well as a non-standard finite difference scheme, comparing the accuracy of the two approaches.

However, uniform grids can fall foul of the issues related to Runge's phenomenon outlined in Chapter 4. For the case of polynomial interpolation, it is well known that Chebyshev roots can help remedy Runge's phenomenon, and in Chapter 5 we explore the building of finite difference schemes out of these Chebyshev roots and the related errors which they reduce.

Often, however, problems require substantially larger grids of many more points where it becomes unfeasible to have finite difference schemes of orders equal to the size of the grid. One way to overcome this is using the grid spacing discussed by Hermanns and Hernández. In Chapter 6 we discuss this spacing and introduce a problem on which we evaluate this scheme. From our findings in Chapter 6, it is clear that although this choice of grid has some benefits, we need a new approach to tackling this problem. Using some intuition, in Chapter 7 we explore the possibility of grid stretching as a suitable approach to overcome this problem, introducing a grid and evaluating the accuracy of our solution.

Chapter 2

Finite Differences

William Taylor (Sections 2.1, 2.3, 2.4) and Moe Okawara (Section 2.2)

2.1 Introduction to Finite Differences

Finite differences provide approximations of the derivative of functions at discrete points, and they are integral to numerical analysis. Finite differences originate from the mathematical definition of the first derivative [1]:

$$u'(x) = \lim_{h \to 0} \frac{u(x+h) - u(x)}{h} \tag{2.1}$$

There exist three basic types of finite difference:

- Forward Differences:

  $$u'(x) \approx \frac{u(x+h) - u(x)}{h} \tag{2.2}$$

  This is a one-sided approximation, as u is only evaluated at points $\geq$ x [2].

- Backward Differences:

  $$u'(x) \approx \frac{u(x) - u(x-h)}{h} \tag{2.3}$$

  This is also a one-sized approximation, as u is only evaluated at points $\leq$ x.

- Central Differences:

  $$u'(x) \approx \frac{u(x+h) - u(x-h)}{2h} \tag{2.4}$$

  This approximation incorporates both sides, being an average of the forward and backwards differences. Therefore it is more accurate.

These approximations are reliant on a small step size h.

The error for each approximation can be bounded with help from the Taylor series, provided we have a bound for $u''(x)$ [1]:

$$u(x+h) = u(x) + u'(x)h + \frac{u''(x)}{2}h^2 \tag{2.5}$$

for some $x \leq t \leq x+h$.

Therefore the error for forwards difference is bounded by

$$\frac{\sup_{x\leq t\leq x+h}|u''(t)|}{2}h \tag{2.6}$$

The error bound for backwards difference is the same - except corresponding to $x-h \leq t \leq x$ - and the error bound for central difference is calculated by subtracting the Taylor series for $u(x-h)$ from the Taylor series for $u(x+h)$, giving us a bound of

$$\frac{\sup_{x-h\leq t\leq x+h}|u'''(t)|}{6}h^2 \tag{2.7}$$

provided we have a bound for $u'''(x)$.

The formulae provided are of order of accuracy 1 for forward and backward, and of order of accuracy 2 for central. Order of accuracy refers to the size of the error proportional to the step size h: a finite difference of order n would have a truncation error of h^n. Alternatively, order of accuracy can be thought of as the highest power of h such that the truncation error tends to zero [2].

As the order of accuracy increases, the coefficients (or weights) of the finite difference become more difficult to calculate. Despite this, determining finite differences to a high order of accuracy is essential to solve numerical problems to a minimal degree of error. Here we present different methods for the calculation of theses weights.

2.2 Method of Undetermined Coefficients

We first explore using the undetermined coefficients method to evaluate the weights for higher order schemes. This utilises the Taylor expansion of a function. We have already seen the first order schemes and second order central difference schemes, so we will work on a fourth order scheme for the first derivative as an example. For a fourth order scheme, there will be 5 weights that need to be calculated. Depending on how close the point we are approximating is to the boundary, the values that will be incorporated into the finite difference scheme will change (i.e. whether it will be central, forward or backward difference). For example, near the boundary we cannot have a central difference as there are not enough points at one end.

We determine the weights for central difference of $u'(x)$. For this we need to incorporate the points $u(x-2h)$, $u(x-h)$, $u(x)$, $u(x+h)$ and $u(x+2h)$. We want to find the weights $a_1, \ldots, a_5$ in terms of h such that

$$u'(x) = a_1u(x-2h) + a_2u(x-h) + a_3u(x) + a_4u(x+h) + a_5u(x+2h) \tag{2.8}$$

The right hand side of (2.8) can be written in terms of just $u(x)$ and h and its derivatives by converting each term into its Taylor expansion[2]:

$$b_0u(x) + b_1u'(x)h + b_2u''(x)h^2 + b_3u^{(3)}(x)h^3 + b_4u^{(4)}(x)h^4 + \mathcal{O}(h^5) \tag{2.9}$$

If we then equate this with $u'(x)$ and compare coefficients, we get that

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1h \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \tag{2.10}$$

We can also write the b_i's properly in terms of the a_i's by expanding the right hand side of (2.8):

$$\begin{bmatrix} b_0 \\ b_1h \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2h & -h & 0 & h & 2h \\ \frac{(-2h)^2}{2} & \frac{(-h)^2}{2} & 0 & \frac{h^2}{2} & \frac{(2h)^2}{2} \\ \frac{(-2h)^3}{3!} & \frac{(-h)^3}{3!} & 0 & \frac{h^3}{3!} & \frac{(2h)^3}{3!} \\ \frac{(-2h)^4}{4!} & \frac{(-h)^4}{4!} & 0 & \frac{h^4}{4!} & \frac{(2h)^4}{4!} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \tag{2.11}$$

Now we combine these two equations and simplify some of the terms to get:

$$\frac{1}{h}\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \\ -8 & -1 & 0 & 1 & 8 \\ 16 & 1 & 0 & 1 & 16 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \tag{2.12}$$

We can easily solve this matrix system using a computer to get that:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \frac{1}{h}\begin{bmatrix} \frac{1}{12} \\ -\frac{2}{3} \\ 0 \\ -\frac{1}{12} \\ \frac{2}{3} \end{bmatrix} \tag{2.13}$$

For off-centred finite differences, the terms we are summing would be different. For example, if we are one point away from the left boundary, we would use the terms $u(x-h)$, $u(x)$, $u(x+h)$, $u(x+2h)$ and $u(x+3h)$. However, this does not change the method and the exact same procedure can be repeated. The same can be said for changing the derivative.

This undetermined coefficients method can actually be generalised to any stencil, any order, and any derivative by repeating the procedure but with different inputs [3].

Although the method of undetermined coefficients works well for relatively lower orders, for higher orders, it is not very efficient. The matrices would

get extremely large and we would also need to solve more matrices for each off-centred scheme [2]. This leads us to look into more efficient methods.

In the next section, we look at the Fornberg Algorithm, which is a much more efficient method for computing the weights of a finite difference scheme.

2.3 Fornberg Algorithm

The Fornberg algorithm outputs the weights of any required finite difference. The algorithm takes an input of evenly spaced grid points and outputs the coefficients for any chosen derivative [4]. This algorithm can be translated into python code which gives weights to a high accuracy up to the fourth derivative [5].

Before Fornberg's algorithm, methods to find weights were of unhelpful complexity and limited to low order derivatives, often inciting systematic errors. Therefore, the development of Fornberg's algorithm - which is numerically fast and stable - was of incredible use for the solving of ODEs using finite differences [4].

Inputting the parameters `np.arange(-6,7), x0=0, n=2` refers to central finite differences of 12th order accuracy, accurate around 0, up to the 2nd derivative. This outputs 3 arrays of weights (for $u(x), u'(x), u''(x)$). These weights can be utilised to produce the following finite differences (accurate to 3 significant figures for display purposes):

$$u'(x) \approx \frac{\begin{pmatrix} 1.80\times10^{-4}u(x-6h) - 2.60\times10^{-3}u(x-5h) \\ +1.79\times10^{-2}u(x-4h) - 7.94\times10^{-2}u(x-3h) \\ +2.68\times10^{-1}u(x-2h) - 8.57\times10^{-1}u(x-h) \\ +1.48\times10^{-16}u(x) + 8.57\times10^{-1}u(x+h) \\ -2.68\times10^{-1}u(x+2h) + 7.94\times10^{-2}u(x+3h) \\ -1.79\times10^{-2}u(x+4h) + 2.60\times10^{-3}u(x+5h) \\ -1.80\times10^{-4}u(x+6h) \end{pmatrix}}{h} \tag{2.14}$$

$$u''(x) \approx \frac{\begin{pmatrix} -6.01\times10^{-5}u(x-6h) + 1.04\times10^{-3}u(x-5h) \\ -8.93\times10^{-3}u(x-4h) + 5.29\times10^{-2}u(x-3h) \\ -2.68\times10^{-1}u(x-2h) + 1.71u(x-h) - 2.98u(x) \\ +1.71u(x+h) - 2.68\times10^{-1}u(x+2h) \\ +5.29\times10^{-2}u(x+3h) - 8.93\times10^{-3}u(x+4h) \\ +1.04\times10^{-3}u(x+5h) - 6.01\times10^{-5}u(x+6h) \end{pmatrix}}{h^2} \tag{2.15}$$

The Fornberg algorithm is flexible in that it can also calculate any possible

form of forward or backward difference [4]. We will show this has great utility for solving differential equations next.

2.4 Solving ODEs

Having obtained the weights for first and second order finite differences with 12th order accuracy, we can now use the finite difference method to solve ODEs. In this case, we will be solving the second order ODE

$$\varepsilon \frac{d^2u}{dx^2} + \frac{du}{dx} = 1 + 2x \tag{2.16}$$

subject to the restrictions

$$0 \leq x \leq 1, \quad 0 < \varepsilon << 1 \tag{2.17}$$

and the boundary conditions

$$u(0) = 0, \quad u(1) = 1 \tag{2.18}$$

This is a two-point boundary value problem. To solve it, we will compute a grid function consisting of values $U_0, U_1, ..., U_m, U_{m+1}$ where U_j is our approximation to the solution $u(x_j)$, with $x_j = jh$ [6].

We will first solve the differential equation using the most basic central difference approximation

$$u'(x) \approx \frac{u(x+h) - u(x-h)}{2h}, \quad u''(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \tag{2.19}$$

In this case we will use 11 grid points, meaning $m = 9$ and the mesh width $h = \frac{1}{10}$. We know that $U_0 = 0$ and $U_{10} = 1$ from the boundary conditions. This leaves us with 9 unknown values, U_1, ... , U_9, to compute [6].

Replacing the derivatives in the differential equation with these 2nd order finite difference approximations gives us a set of algebraic equations

$$\frac{\epsilon}{h^2}\left(U_{j-1} - 2U_j + U_{j+1}\right) + \frac{1}{2h}\left(-U_{j-1} + U_{j+1}\right) = 1 + 2x_j \tag{2.20}$$

for $j = 1, 2, \ldots, 9$ We have a linear system of 9 equations for 9 unknowns, which can be written in the form

$$AU + BU = F \tag{2.21}$$

where $U = [U_0, U_1, ..., U_9, U_{10}]^T$ [6]. Through the distributive property of matrices, this can be rewritten as

$$CU = F \tag{2.22}$$

where C is the 11×11 matrix

$$\begin{bmatrix} 1 & & & & & \\ \frac{\epsilon}{h^2}+\frac{1}{2h} & \frac{-2\epsilon}{h^2} & \frac{\epsilon}{h^2}+\frac{1}{2h} & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & \frac{\epsilon}{h^2}+\frac{1}{2h} & \frac{-2\epsilon}{h^2} & \frac{\epsilon}{h^2}+\frac{1}{2h} & \\ & & & & 1 \end{bmatrix} \tag{2.23}$$

Note that we have added our boundary conditions as rows 1 and 11 of the matrix for simplicity.

$$F = \begin{bmatrix} 0 \\ 1+2x_1 \\ \vdots \\ \vdots \\ 1+2x_9 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1.2 \\ \vdots \\ \vdots \\ 2.9 \\ 1 \end{bmatrix} \tag{2.24}$$

This tridiagonal linear system can be solved using linear algebra. We have

$$U = C^{-1}F \tag{2.25}$$

so we can calculate the solution for U, illustrated by this plot for $\epsilon = 10^{-3}$:

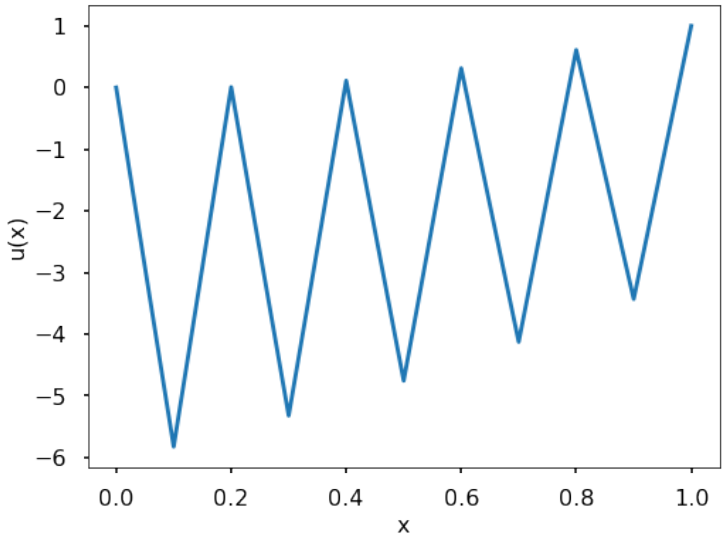


Figure 2.1: Graph showing 2nd order accuracy finite difference solution to the ODE using 10 grid points.

However, as one might guess from this plot, the selection of 11 grid points gives a very inaccurate approximation for this ODE.

To resolve this issue we can use python code to automate the process of solving the system of linear equations for larger numbers of grid points - which requires larger matrices [7]. Here we have plotted the solutions of this ODE for larger n values:

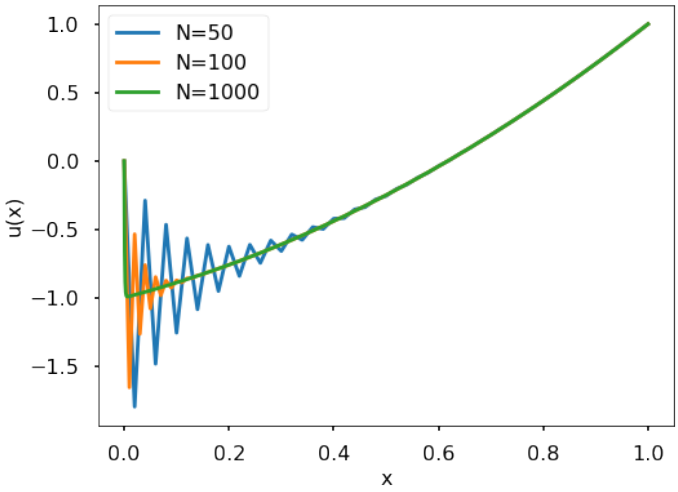


Figure 2.2: Graph showing 2nd order accuracy finite difference solutions to the ODE, varying number of grid points.

Additionally, using automated python code we can obtain solutions for various ϵ values with $n = 1000$:

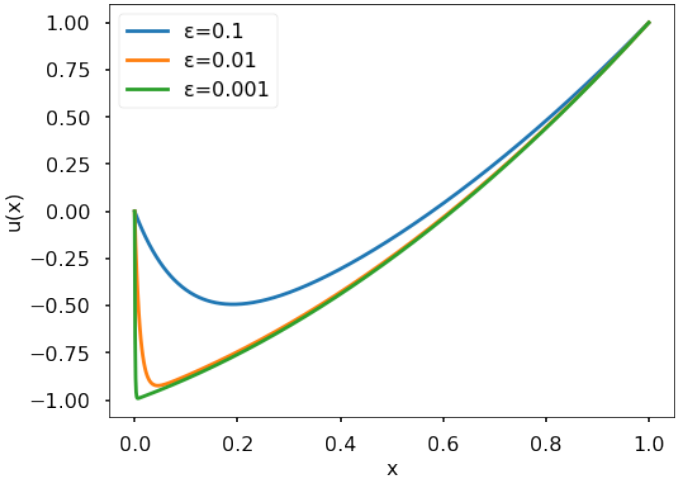


Figure 2.3: Graph showing 2nd order accuracy finite difference solutions to the ODE, varying epsilon.

For higher orders of accuracy, the process becomes more complicated. In this case, we will be solving the same ODE using 12th order finite differences obtained from the Fornberg algorithm. The matrix of weights we construct incorporates 5 types of forward difference and 5 types of backwards difference along with our standard 12th order accuracy central difference.

These finite differences range from $(x - h)$ to $(x + 11h)$ at the extremes to $(x - 11h)$ to $(x + 11h)$ at the extremes.

The matrix constructed can then be solved with the help of python to output our more accurate solution to this ODE. In a similar fashion to our 2nd order process we can vary both number of grid points and ϵ size:

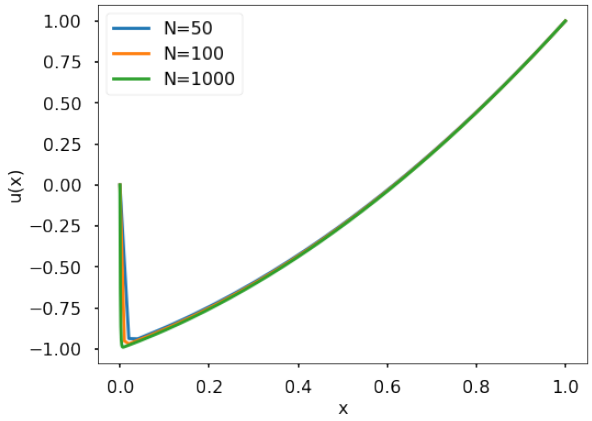


Figure 2.4: Graph showing 12th order accuracy finite difference solutions to the ODE, varying number of grid points.

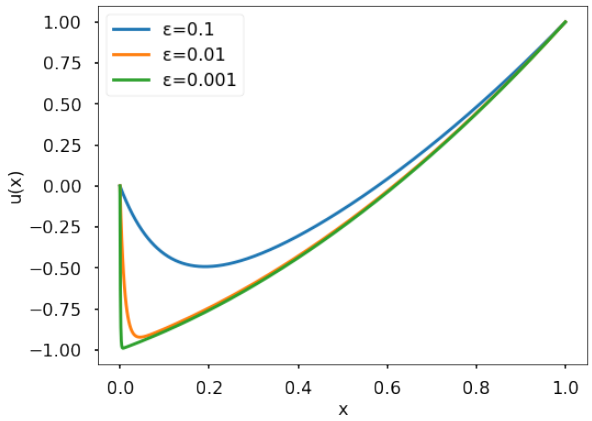


Figure 2.5: Graph showing 12th order accuracy finite difference solutions to the ODE, varying epsilon.

Comparing these graphs, we can observe that the oscillations present in our plotted 2nd order accuracy solution for $n = 50$ and $n = 100$ are not present in our plotted 12th order accuracy solution for $n = 50$ $n = 100$, as higher order of accuracy finite differences translates to higher accuracy ODE solutions.

Chapter 3

The Bratu Problem

Nicholas Matheou

3.1 Exact solutions for the Bratu Problem

Some of the numerical methods to solve the one dimensional non-linear Bratu problem given by

$$u'' + \lambda e^u = 0, \qquad 0 \leq x \leq 1 \text{ [8]} \tag{3.1}$$

where $\lambda > 0$, with boundary conditions

$$u(0) = 0, \qquad u(1) = 0 \text{ [8]} \tag{3.2}$$

are outlined in this section. The methods used are non-standard and standard finite difference methods and a comparison is made between the approximate solution and the exact solution [8].

The exact solution to the Bratu problem is given by

$$u(x) = -2 \ln \left[\frac{\cosh\left(\left(x - \frac{1}{2}\right)\frac{\theta}{2}\right)}{\cosh\left(\frac{\theta}{4}\right)} \right] \text{ [8]} \tag{3.3}$$

where θ satisfies

$$\theta = \sqrt{2\lambda} \cosh\left(\frac{\theta}{4}\right) \text{ [8]} \tag{3.4}$$

The solution is unique when $\lambda = \lambda_c$, where λ_c is the critical value which satisfies

$$1 = \frac{1}{4}\sqrt{2\lambda_c} \sinh\left(\frac{\theta}{4}\right) \text{ [8]} \tag{3.5}$$

Note that, when $\lambda < \lambda_c$ there are two solutions to the Bratu problem and when $\lambda > \lambda_c$ there are no solutions to the Bratu problem [8].

The exact value of λ_c can be calculated by using equations (3.4) and (3.5) and applying the Newton-Raphson method [9].

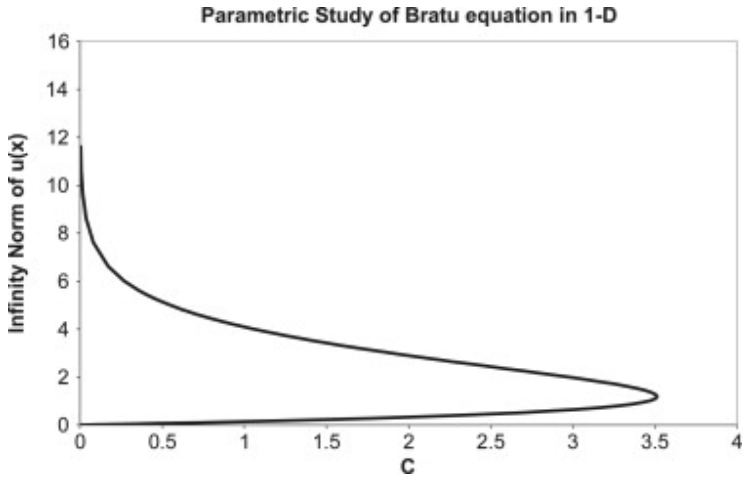


Figure 3.1: How the infinity norm of the solution varies with λ. This figure was taken from "A simple solution of the Bratu problem" by A.Mohson [10]. The letter C in the graph represents λ.

The graph above shows the supremum of the solution in the interval $[0, 1]$ against the value of λ. More specific, there are two values for the solution since all vertical lines $\lambda = a$, for $a < \lambda_c$, intersect the curve twice. On the other hand, when the vertical line $\lambda = \lambda_c$ intersects the graph once at the turning point, there is only one solution. For all vertical lines $\lambda = a$, for $a > \lambda_c$, there is no point of intersection with the graph, hence there are no solutions for those values of λ.

3.2 Finite Difference Methods

By using the finite difference methods, the interval $[0, 1]$ is split into N smaller intervals of equal length. A uniform grid is used for both the standard and non-standard finite difference methods but the difference is the denominator used.

For the standard finite difference method the denominator for a second order accurate solution is h^2 as shown in Chapter 2, whereas for the non-standard method it is $2\ln[\cosh(h)] = h^2 + o(h^2)$ [8]. As $h \to 0$ the standard and non-standard finite difference methods are equal [8]. The grid points are $\{x_i\}$, $i \in \{1, ..., N\}$ with a separation constant $h = 1/N$.

3.2.1 Standard finite difference

For the standard finite difference method, the points on the grid are $x_i = i/N$ for $i \in \{1, ..., N\}$. A second order accurate finite difference scheme is used to

obtain the following system of non-linear equations,

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \lambda e^{u_i} = 0, \qquad i \in \{1, \ldots, N\} \tag{3.6}$$

where

$$\frac{d^2 u_i}{dx^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \tag{3.7}$$

In order to solve the system of non-linear equations, Newton's Method is used. More specific, for a vector $\mathbf{v} = (v_1, ..., v_{N-1})$, the Newton function is given by,

$$N(\mathbf{v}) = \mathbf{v} - J_{\mathbf{v}}^{-1} f(\mathbf{v}) \text{ [11]} \tag{3.8}$$

$J_{\mathbf{v}}$ is the Jacobian of the function $f = (f^1, ..., f^{N-1})$ at $\mathbf{v}$, where

$$f^i(\mathbf{v}) = \frac{v_{i+1} - 2v_i + v_{i-1}}{h^2} + \lambda e^{v_i}, \quad i \in \{1, ..., N-1\} \tag{3.9}$$

Applying the above method to the solution of the Bratu problem, an initial vector $\mathbf{u}^0 = (u_1^0, ..., u_{N-1}^0)$ is chosen. Then, the boundary conditions (3.2) are applied, so $u_0^n = u_N^n = 0$ for all $n \in \mathbb{N}$. Recursively, define $\mathbf{u}^{n+1} = N(\mathbf{u}^n)$ until the vector $\mathbf{u}^n$ is at the accuracy required compared to the exact solution at the grid points [11].

3.2.2 Non-standard finite difference

For the non-standard finite difference method the following system of non-linear equations is obtained,

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{2\ln[\cosh(h)]} + \lambda e^{u_i} = 0, \qquad i \in \{1, ..., N\} \tag{3.10}$$

This is called a Mickens-difference scheme and it is proven that the non-standard finite difference method is more accurate than the standard method.

As in the case for the standard finite difference method, Newton's Method is applied but the function $f = (f^1, ..., f^{N-1})$ is different in this case.

First, define

$$f^i(\mathbf{v}) = \frac{v_{i+1} - 2v_i + v_{i-1}}{2\ln[\cosh(h)]} + \lambda e^{u_i} \tag{3.11}$$

Then again an initial vector $\mathbf{u}^0 = (u_1^0, ..., u_{N-1}^0)$ is chosen and recursively define $\mathbf{u}^{n+1} = N(\mathbf{u}^n)$ [8].

3.2.3 Convergence of solution

It is important to check whether the approximated solution converges after applying many iterations of the Newton's method. Suppose n iterations are applied. To test if the solution converges, two steps must be done:

- Show that the solution reaches a high enough accuracy close to the exact solution after N iterations, where $N < n$.
- Show that the difference between each iteration decreases to 0 as $n \to \infty$

3.3 Numerical Results

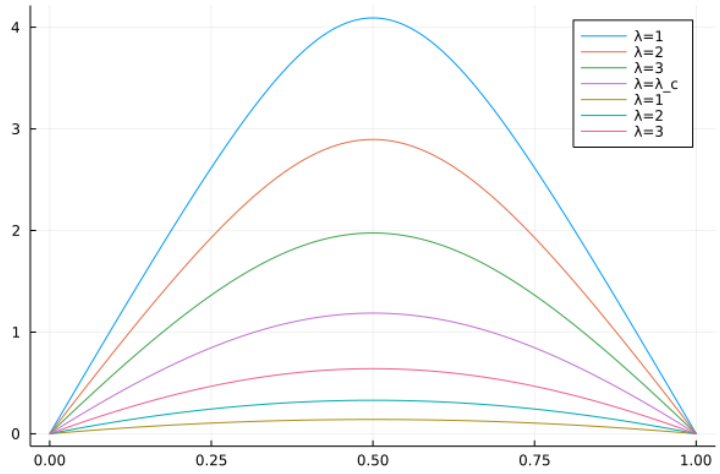


Figure 3.2: Graph showing the exact solution for $\lambda = 1$, $\lambda = 2$, $\lambda = 3$ and $\lambda = \lambda_c$.

For $\lambda = \lambda_c$ there is only one solution. For $\lambda = 1$, $\lambda = 2$, $\lambda = 3$ where $\lambda < \lambda_c$, there are two solutions, one above the unique solution and one below. As the value of λ increases to λ_c, the solutions above and below converge to the unique solution.

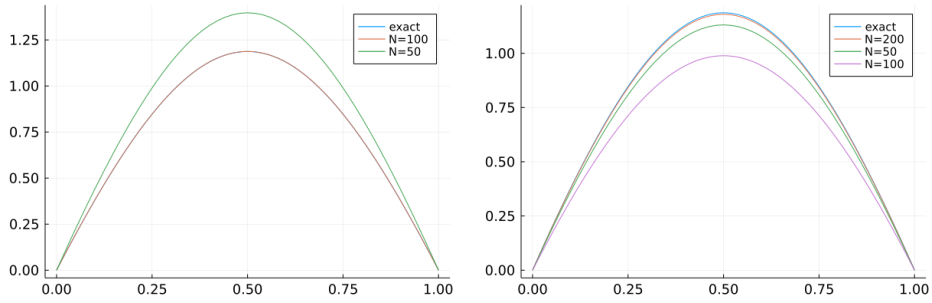
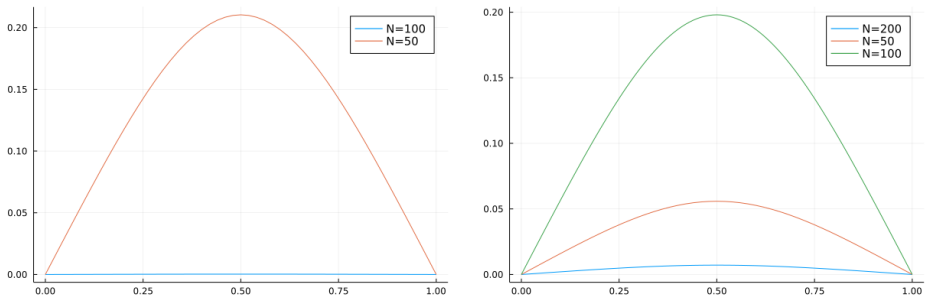


Figure 3.3: Standard finite differences (a) and non-standard finite differences (b) for $N = 50$, $N = 100$ and $N = 200$, compared to exact solution.

The graphs above shows how the curves for increasing values of N converge to the exact solution for $\lambda = \lambda_c$. One graph is by using the standard finite difference method and the other graph by using the non-standard finite difference method.



(a) (b)

Figure 3.4: Error for standard and non-standard finite differences for $N = 50$, $N = 100$ and $N = 200$.

Both graphs show the error when using different values of N for $\lambda = \lambda_c$. One graph is the error from the standard finite difference method and the other from the non-standard finite difference method. As the value of N increases the error approaches 0.

Chapter 4

Runge's Phenomenon

Niall Oswald

4.1 Polynomial Interpolation

As we will later see in Section 6.1, polynomial interpolation can also be used to generate finite difference schemes. In the context of high order finite difference schemes, this brings along with it the flaws of high degree polynomial interpolation. One such flaw it that of Runge's phenomenon, which can lead to large errors in polynomial interpolation towards its boundary points [12].

An example of this can be seen in Figure 4.1, where we use Lagrange interpolation to interpolate the Runge function (4.2) using a high degree polynomial. Also plotted is the Runge function itself, so it is clear to see that large errors arise away from the interpolation points as we approach the boundary.

Clearly, in the same manner, interpolations of derivatives would experience similar errors [12], which would then affect our finite difference schemes by inducing large errors in our discretised differential operators.

4.2 Convergence and Divergence of Lagrange Interpolation

In order to explain why Runge's phenomenon arises, we need to introduce the concept of a node density function [12]:

Definition 1 *A general node density function $\mu(x)$ gives the density distribution of grid points over the domain $[-1, 1]$. The node density function is normalised, so $\int_{-1}^{1} \mu(x)dx = 1$, leading to the number of nodes in a small interval of length dx to be $N\mu(x)dx$, where N is the total number of nodes.*

The basis of the argument is formed by the theorem given by Fornberg [12]:

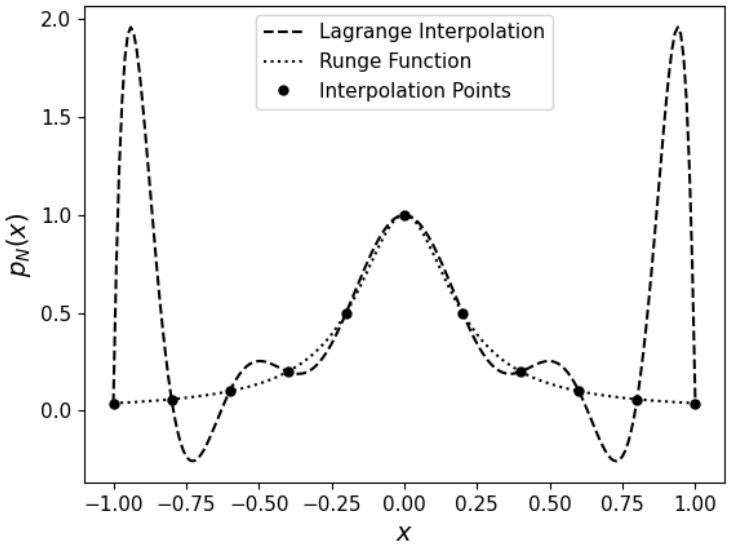


Figure 4.1: Lagrange interpolation of the Runge function (4.2) using $N = 11$ uniformly spaced grid points. Also pictured are the interpolation points used in the Lagrange interpolation and the true values of the Runge function as well.

Theorem 1 *Given a node density function $\mu(x)$ (on $[-1, 1]$), we form the potential function,*

$$\phi(z) = -\int_{-1}^{1} \mu(x) \ln|z - x| dx \, (+constant) \tag{4.1}$$

Then:

1. *The polynomials $p_N(z)$ (interpolating an analytic function $f(z)$ at N nodes on $[-1, 1]$) converges to $f(z)$ inside the largest equi-potential curve for $\phi(z)$ that does not enclose any singularity of $f(z)$, and diverges outside that curve.*

   Let z_0 denote the limiting singular point of $f(z)$ (or, for now, any point along this largest equi-potential curve).

2. *The rate of convergence/divergence is exponential, like $\alpha(z)^N$ where $\alpha(z) = e^{\phi(z_0)-\phi(z)}$.*

   This result is to be understood in the same sense as how a Taylor series around the origin converges/diverges like $\alpha(z)^N$ with $\alpha(z) = |z/z_0|$; i.e., the error satisfies $|R_N(z)|^{1/N} \to \alpha(z)$ as $N \to \infty$.

3. *Psuedospectral approximations to any derivative converge (or diverge) in the same fashion as the interpolant does to the function.*

We will see how this theorem is useful for explaining Runge's phenomenon as a worked example using the Runge function (4.2).

4.3 The Runge Function

Consider Runge's function [13], given towards the end of the paper:

$$f(x) = \frac{1}{1 + 25x^2} \tag{4.2}$$

In the case of uniformly spaced grid points, the node density can be reduced to $\mu(x) \equiv 1/2$ [12]. The potential $\phi(z)$ (4.1) can then be evaluated to the more usable form [12]:

$$\phi(z) = -\frac{1}{2}\Re\left[(1-z)\ln(1-z) - (-1-z)\ln(-1-z)\right] + C \tag{4.3}$$

As we aim to apply Theorem 1, we need to find the largest equi-potential curve on $\phi(z)$ that does not enclose any of the singularities of the Runge function (4.2).

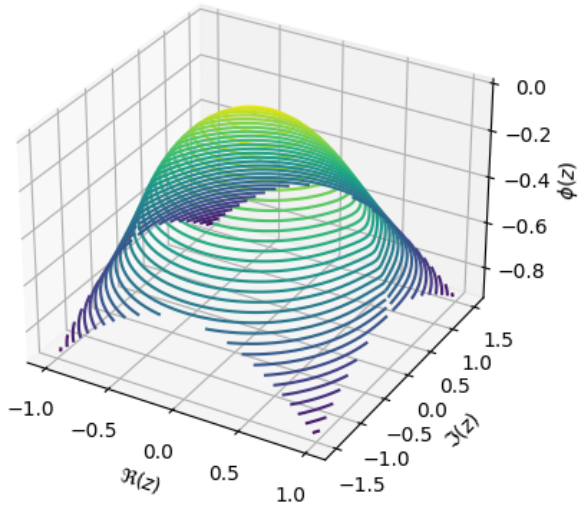


Figure 4.2: Contour plot of $\phi(z)$ (4.3) for uniform grid spacing with $C = 0$, noting that each contour is an equi-potential curve in the complex numbers.

As we are only interpolating along the reals, we only need to consider points from $[-1, 1]$, so as $f(x)$ has singularities at $z = \pm\frac{1}{5}i$, for $x \in [-1, 1]$ we consider the function:

$$\begin{aligned}\psi(x) &= \phi(x) - \phi(\pm i/5) \\ &= -\frac{1}{2}(1-x)\ln(1-x) - \frac{1}{2}(1+x) + \frac{1}{2}\ln\frac{26}{25} + \frac{1}{5}\arctan(5)\end{aligned} \tag{4.4}$$

Where $\psi(x)$ is negative, we know that x is outside of the largest equipotential curve which does not contain the singularities of $f(x)$. Hence, using Theorem 1, if we consider a domain on which there are points where $\psi(x)$ is negative, then we can expect the interpolation of $f(x)$ to diverge as $N \to \infty$, and converge otherwise.

So computing the roots for $\psi(x)$, given in Figure 4.3 (a), we get $x = \pm 0.7267$. Thus as N becomes large, we expect the Lagrange interpolation of the Runge function to converge for $|x| < 0.7267$ and diverge for $|x| > 0.7267$. This is also clearly seen in Figure 4.3 (b). As N increases, outside of the stable region ($|x| > 0.7267$) we see that Lagrange interpolants diverge, but inside of the stable region we see convergence with increasing N.

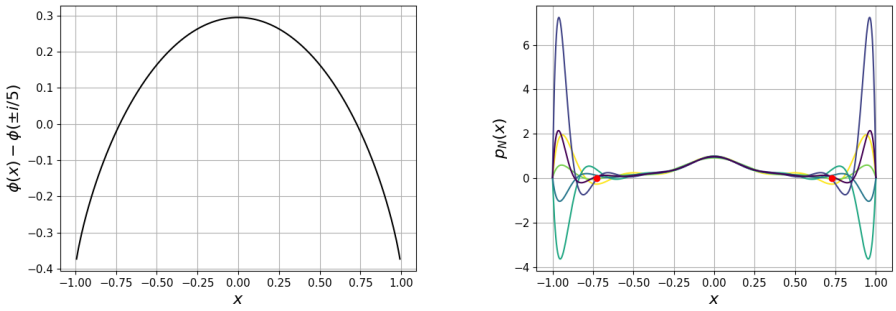


Figure 4.3: (a) $\psi(x)$ (4.4) plotted from -1 to 1. (b) Lagrange interpolations $p_N(x)$ of the Runge function (4.2) for various values of N from $N = 10$ (yellow) to $N = 15$ (purple), with the roots of $\psi(x)$ plotted in red.

Now that we understand Runge's phenomenon, we will now explore Chebyshev polynomials as one possible way to overcome this problem in Chapter 5.

Chapter 5

Chebyshev Polynomials

Yaofeng Lan

5.1 Maximal Approximation

As stated in Chapter 4, uniform grids do not perform well near the boundary of the functions. So we wish to obtain certain grids that could minimise the maximal error. Such grids do exist if we consider the case when $q = N$, and all stencils collapse into one. Then we could obtain the following formulae for pointwise error $\varepsilon(x)$ [14]:

$$\varepsilon(x) = \left| \pi(x) \frac{u^{(q+1)}(\xi)}{(q+1)!} \right| \tag{5.1}$$

where $\xi = \xi(x)$ is a certain value on $[-1, 1]$ determined by x, and $\pi(x)$ is the following polynomial:

$$\pi(x) = \prod_{i=1}^{N} (x - x_i) \tag{5.2}$$

As $u^{(q+1)}(\xi)/(q+1)!$ is determined by the function $u(x)$ and the value of x, they cannot be minimised by adjusting the grids [14]. We can only minimise the value of $\max_{x\in[-1,1]} |\pi(x)|$. Such a minimum can be achieved with the following nodes:

$$x_i = \cos\left(\frac{2i-1}{2N}\pi\right), \qquad i = 1, 2, \ldots, N \tag{5.3}$$

which are also known as Chebyshev roots. And for arbitrary nodes $\{x_i'\}_{i=1}^{N} \subset [-1, 1]$ the following equations can be proved[15]:

$$\max_{x\in[-1,1]} \left| \prod_{i=1}^{N} (x - x_i) \right| = \frac{1}{2^{n-1}} \tag{5.4}$$

$$\frac{1}{2^{n-1}} \leq \max_{x\in[-1,1]} \left| \prod_{i=1}^{N} (x - x_i') \right| \tag{5.5}$$

To prove (5.4), consider the definition of the first kind of Chebyshev polynomials [15]:

$$T_n(x) = \cos\left(n \cos^{-1} x\right), \qquad x \in [-1, 1] \tag{5.6}$$

And the following recursion form [15] can be obtained (the proof is in Appendix C):

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \tag{5.7}$$

By setting $T_n(x) = \sum_{i=0}^{n} b_i^{(n)} x^i$, we can prove by induction that $b_n^{(n)} = 2^{n-1}$ for $n \geq 1$. When $n = 1$, $T_1(x) = x$ and $b_1^{(1)} = 1$ as desired. Consider $b_n^{(n)} = 2^{n-1}$ holds for $1 \leq n \leq k$. Then:

$$\begin{aligned} T_{k+1}(x) &= 2xT_k(x) - T_{k-1}(x) \\ &= 2x \sum_{i=0}^{k} b_i^{(k)} x^i - \sum_{i=0}^{k-1} b_i^{(k-1)} x^i \\ &= 2b_k^{(k)} x^{k+1} + 2b_{k-1}^{(k)} x^k + \sum_{i=0}^{k-1} \left(2b_{i-1}^{(k)} - b_i^{(k-1)}\right) x^i \end{aligned}$$

Hence,

$$\begin{aligned} b_{k+1}^{(k+1)} &= 2b_k^{(k)} \\ &= 2^k \end{aligned} \tag{5.8}$$

As for x_i in (5.3), by definition of $T_n(x)$ in (5.6), we have:

$$\begin{aligned} T_N(x_i) &= \cos\left(N \cos^{-1} \cos\left(\frac{2i-1}{2N}\pi\right)\right) \\ &= 0 \end{aligned}$$

By fundamental theorem of algebra, x_i are roots of $T_N(x)$. With (5.8), we can rewrite $T_n(x)$ as the following:

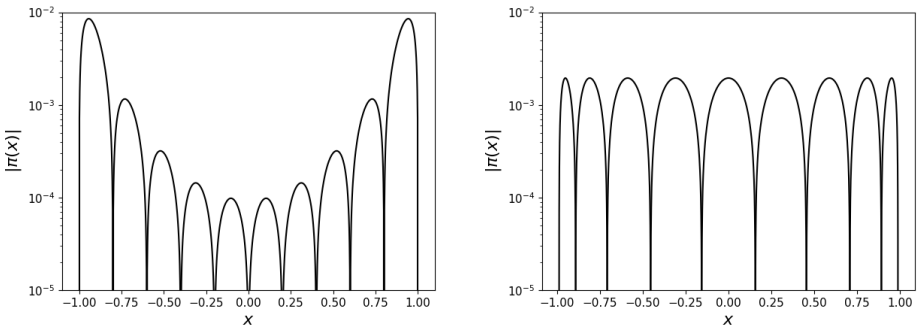
$$T_n(x) = 2^{n-1} \prod_{i=1}^{n} \left(x - \cos\left(\frac{2i-1}{2n}\pi\right)\right) \tag{5.9}$$

By (5.6), we have $\max_{x\in[-1,1]} |T_n(x)| = 1$. With (5.9), we can obtain (5.4).

Now we prove (5.5) by contradiction[15]. Assume $\exists P(x) = \prod_{i=1}^{n}(x - x_i')$ s.t. $\max_{x\in[-1,1]} |P(x)| \leq 1/2^{n-1}$. Then set $F(x) = P(x) - T_n(x)/2^{n-1}$. As $P(x)$ and $T_n(x)/2^{n-1}$ are both monic polynomials with degree n, $\deg(F(x)) \leq n-1$. With $|P(x)| \leq 1/2^{n-1}$, we could obtain the following:

$$\begin{aligned} F(1) &= P(1) - \frac{1}{2^{n-1}} && < 0 \\ F\left(\cos\left(\frac{\pi}{n}\right)\right) &= P\left(\cos\left(\frac{\pi}{n}\right)\right) + \frac{1}{2^{n-1}} && > 0 \\ F\left(\cos\left(\frac{2\pi}{n}\right)\right) &= P\left(\cos\left(\frac{2\pi}{n}\right)\right) - \frac{1}{2^{n-1}} && < 0 \end{aligned}$$

and so on. There would be at least $n+1$ sign changes in $F(x)$. By intermediate value theorem, there would be at least n zeroes of $F(x)$. Hence $\deg(F(x)) \geq n$, which is contradicted by $\deg(F(x)) \leq n-1$. Therefore, no such $P(x)$ exists.



(a) (b)

Figure 5.1: $\pi(x)$ using (a) uniform grids and (b) Chebyshev roots.

Thus, under the case of $q = N$, Chebyshev roots minimise the maximal errors of the finite difference method.

As shown in Figures 5.1, the $\pi(x)$ using uniform grid has larger maximum value than that using Chebyshev roots. And as expected from Runge's phenomenon, the $\pi(x)$ that uses uniform grid oscillates much larger than that using Chebyshev roots near the boundary.

We use ODE (2.16), with restriction (2.17) and boundary condition (2.18) as an example. And we set $\varepsilon = 10^{-2}$. Based on Appendix B, we can obtain analytical solution for ODE (2.16). Therefore, we can have Figure 5.3 being the error plot of different approximation methods with $q = N = 50$. It can be observed from Figure 5.3 that Chebyshev roots perform much better than uniform grids with regard to the ODE 2.16.

Moreover, it can be concluded that, when using same number of nodes, Chebyshev roots would perform better than the uniform grids. If we restrict both methods using same number of nodes, the only thing we can do to change the accuracy of using uniform grids is to decrease the value of q, but the errors would just increase. So errors of using uniform grids would still be higher than that of using Chebyshev roots.

5.2 Spectral Collocation Method

Chebyshev polynomials do hold great properties when it comes to numerical analysis. Spectral collocation method is another numerical method of applying Chebyshev polynomials. And it can be observed later that spectral collocation method can be much more accurate than finite difference method.

We still take ODE (2.16) with restrictions (2.17) and boundary conditions (2.18) as an example.

As we approximate $u(x)$ on interval $[0,1]$, which is different from the domain of $T_n(\theta)$, we could use affine transformation to normalise interval $x \in [0,1]$ to interval $\theta \in [-1,1]$ [16]:

$$\theta = 2x - 1 \tag{5.10}$$

Then we can set the following:

$$u(x) = \sum_{n=0}^{N} a_n T_n(\theta) \tag{5.11}$$

where $\{a_n\}_{n=0}^{N}$ are some constants in $\mathbb{R}$.

Inserting (5.10) and (5.11) into equation $\varepsilon d^2u/dx^2 + du/dx = 1 + 2x$:

$$\begin{aligned} \theta + 2 &= 4\varepsilon \frac{d^2u}{d\theta^2} + 2\frac{du}{d\theta} \\ &= \sum_{n=0}^{N} \left(4\varepsilon T_n^{(2)}(\theta) + 2T_n^{(1)}(\theta) \right) a_n \end{aligned} \tag{5.12}$$

By using following nodes, also known as Chebyshev extrema [17]:

$$\theta_k = \cos\left(\frac{k}{N}\pi\right), \qquad k = 0,\ 1,\ 2, \cdots,\ N \tag{5.13}$$

along with boundary conditions, we can obtain:

$$\begin{cases} \theta_k + 2 = \displaystyle\sum_{n=0}^{N} \left(4\varepsilon T_n^{(2)}(\theta_k) + 2T_n^{(1)}(\theta_k) \right) a_n, \quad k = 1,\ 2, \cdots,\ N-1 \\ \displaystyle\sum_{n=0}^{N} (-1)^n a_n = 0 \\ \displaystyle\sum_{n=0}^{N} a_n = 1 \end{cases} \tag{5.14}$$

Then we can obtain a system of equations:

$$Ax = b \tag{5.15}$$

Where $A = (a_{ij})_{(N+1)\times(N+1)}$, with:

$$a_{ij} = \begin{cases} (-1)^{j-1}, & i = 1 \\ 4\varepsilon T_j^{(2)}(\theta_{i-1}) + 2T_j^{(1)}(\theta_{i-1}), & 2 \le i \le N \\ 1, & i = N+1 \end{cases} \tag{5.16}$$

and:

$$x = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ \theta_1 + 2 \\ \theta_2 + 2 \\ \vdots \\ \theta_{N-1} + 2 \\ 1 \end{pmatrix} \tag{5.17}$$

In order to calculate every entry of A by computer, we need an algorithm to compute $T_n^{(k)}(x)$.

The equations we used are the following (we consider $T_n^{(0)}(x) = T_n(x)$) [18]:

$$T_0(x) = 1 \tag{5.18a}$$

$$T_1(x) = x \tag{5.18b}$$

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \qquad n \geq 2 \tag{5.18c}$$

$$T_1^{(k)}(x) = T_0^{(k-1)}(x), \qquad k \geq 1 \tag{5.18d}$$

$$T_n^{(k)}(x) = 2xT_{n-1}^{(k)}(x) - T_{n-2}^{(k)}(x) + 2kT_{n-1}^{(k-1)}(x), \qquad k \geq 1 \text{ and } n \geq 2 \tag{5.18e}$$

The proof of equations (5.18) is in Appendix C.

By solving x in (5.15) and inserting everything back to (5.11), we could obtain the approximated polynomial of $u(x)$.

Figure 5.2 shows approximations of different methods all with $q = N = 50$ and $\varepsilon = 10^{-2}$. And as shown in Figure 5.3, when using same number of nodes, the error of spectral collocation method is much smaller than that of the finite difference method.

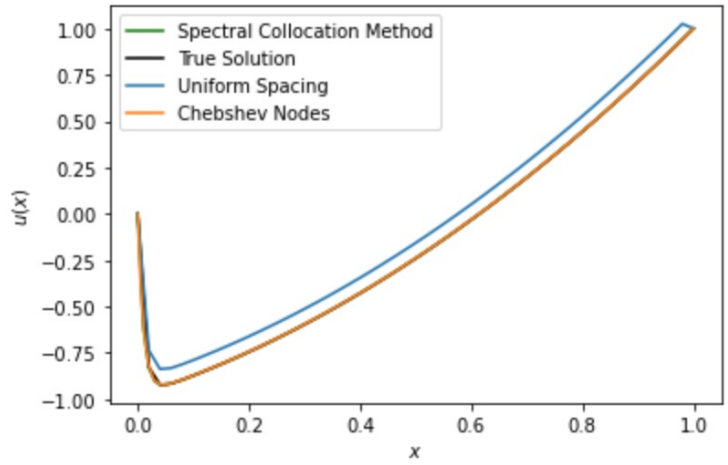


Figure 5.2: Approximation of $u(x)$ using the spectral collocation method.

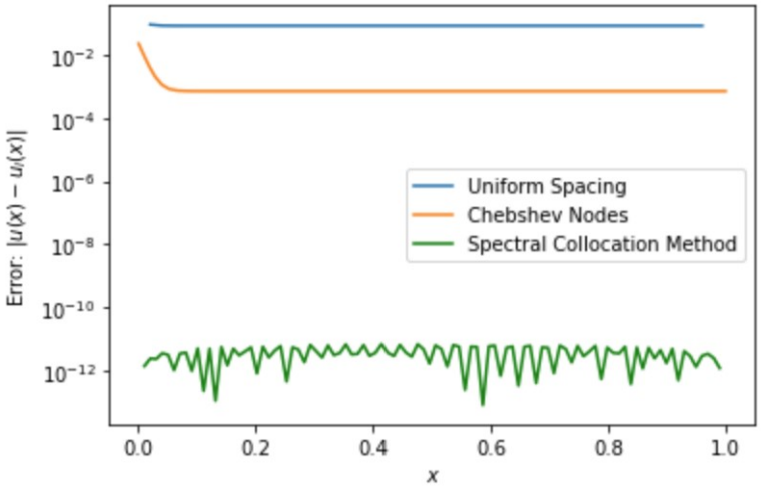


Figure 5.3: Errors of different approximations.

Chapter 6

Non-uniform Grid Point Spacing

Niall Oswald

6.1 Finite Difference Methods via Interpolation

A separate approach to finding finite difference schemes to that outlined in Chapter 2 is via interpolation. Difference operators can be approximated by formulating an interpolation of the unknown function $u(x,t)$ based on a collection of points $u_0,\dots,u_N$, where $u_i(t)=u(x_i,t)$ for a set of interpolation points $x_0,\dots,x_N$. This interpolation is often done piecewise, where a collection of interpolary polynomials of degree q will give rise to a finite difference scheme of order q [14].

We begin by fixing N, the number of nodes, and q, the degree of the interpolary polynomials. Then we consider a collection of interpolants $I_i(x)$ defined on $\Omega_i=[x_{i-1/2},x_{i+1/2})$, where,

$$I_i(x)=\sum_{j=s_i}^{s_i+q} l_{ij}(x)u_j, \qquad\qquad l_{ij}(x)=\prod_{\substack{m=0\\ s_i+m\neq j}}^{q}\frac{x-x_{s_i+m}}{x_j-x_{s_i+m}} \tag{6.1}$$

for $i=0,\dots,N$, which together build a piecewise approximation of the function $u(x,t)$ [14]. Also, the s_i are defined to be,

$$(s_0,\dots,s_N)=(\underbrace{0,\dots,0}_{q/2\text{ times}},0,1,\dots,N-q,\underbrace{N-q,\dots,N-q}_{q/2\text{ times}}) \tag{6.2}$$

for q even, and,

$$(s_0,\dots,s_N)=(\underbrace{0,\dots,0}_{(q-1)/2\text{ times}},0,1,\dots,N-q,\underbrace{N-q,\dots,N-q}_{(q+1)/2\text{ times}}) \tag{6.3}$$

for q odd [14].

6.2 Hermanns-Hernández Spacing

As we have seen in Chapter 4, using a uniform grid spacing is sub-optimal and can lead to large interpolation errors. One method to overcome this is to use a non-uniform grid spacing, which can reduce the interpolation error [14]. When $q = N$, i.e. when the order of the finite-difference scheme matches the number of grid points, the roots of the Chebyshev polynomial $T_N(x)$ minimise the interpolation error, as explored in Chapter 5. However, when $q < N$, the Chebyshev roots no longer minimise the errors and a new scheme is required [14].

The grid found to minimise the error is given by Hermanns and Hernández, where the grid is found by considering a collection of polynomial interpolations of degree $q-1$ of some unknown points $y_1, \ldots, y_N$ and imposing that each polynomial factor that builds the interpolation has the same maximum amplitude in their respective domains [14]. Hermanns and Hernández only give the method for interpolations of functions defined on $[-1, 1]$, however this can easily be modified to work with any domain, as outlined below, which we will make use of later in Section 6.4 [14].

Suppose that $q < N$ and we are trying to find a non-uniform spacing on the domain $[a, b]$ with $a < b$. Start by considering N points $y_1, \ldots, y_N$, which will be the points which divide domain into the $\Omega_i = [y_i, y_{i+1})$ as in Section 6.1.

Now consider each polynomial factor, $\pi_i(x)$ defined on Ω_i,

$$\pi_i(x) = \prod_{m=0}^{q-1} (x - y_{s_i+m}) \tag{6.4}$$

which appears in interpolation error, for $i = 1, \ldots, N-1$. For each of these factors, denote x_i to be the extremum of $\pi_i(x)$ on Ω_i. Complete this set with $x_0 = a$ and $x_N = b$. To minimise the errors, we impose that the polynomial factors have equal amplitudes, so,

$$|\pi_1(x_0)| = |\pi_1(x_1)| = |\pi_2(x_2)| = \cdots = |\pi_{N-1}(x_{N-1})| = |\pi_{N-1}(x_N)| \tag{6.5}$$

This equation can be quite tricky to solve, and so we use an approach whereby we start with a uniform distribution of $y_1, \ldots, y_N$ and then fit them to give equal amplitudes, as we will explore in more detail below in Section 6.3.

Once we have obtained the $x_0, \ldots, x_N$ and $y_1, \ldots, y_N$, we build the piecewise polynomial interpolation as given in Section 6.1, using the $x_0, \ldots, x_N$ as our interpolation points, and $a, y_1, \ldots, y_N, b$ as the respective endpoints of each subdomain, i.e.,

$$\Omega_0 = [a, y_1), \quad \Omega_i = [y_i, y_{i+1}), \quad \Omega_N = [y_N, b] \tag{6.6}$$

for $i = 1, \ldots, N-1$.

6.3 Implementing the Non-uniform Scheme

To solve the non-linear system of equations mentioned in (6.5), we aim to find a grid of points $y_1, \ldots, y_N$ for which, the amplitudes are all equal. As previously stated, this is quite a challenging system to solve, so we introduce a novel approach using an iterative scheme; aiming to fit an initial guess to the solution grid.

As the extrema $x_1, \ldots, x_{N-1}$ depend on the polynomials $\pi_1(x), \ldots, \pi_{N-1}(x)$, which in turn depend on the grid points $y_1, \ldots, y_N$, it is relatively straightforward to compute the extrema and their corresponding amplitudes for a given set of grid points. Therefore, it would be reasonable to propose an iterative scheme which takes a guess of initial grid points and computes the resulting amplitudes. Then, by some criterion we can use the pairwise differences in amplitudes to redistribute the initial guess of grid points.

Empirically, it is found that moving grid points closer together leads to lower maximal amplitudes between these roots. From this observation we can build a criterion whereby if we compare two adjacent amplitudes, say $|\pi_i(x_i)|$ and $|\pi_{i+1}(x_{i+1})|$ (a similar observation is found for the boundary nodes), then if we find the left amplitude ($|\pi_i(x_i)|$) to be greater, we choose to move the grid point between them (y_{i+1}) to the left and vice-versa.

It is of note that this criterion was developed empirically and has no rigorous basis. Despite the dubious nature of building an algorithm from intuition and empirical observations, this does not bring into question the efficacy of the proposed algorithm. This is due to the fact we are simply aiming to find a collection of grid points which satisfies (6.5) as accurately as possible. This does however lead to some concerns that the algorithm may not be guaranteed to produce a collection of grid points with small differences in corresponding amplitudes. Because of this, it would be of great benefit to look further into the behaviour of the polynomials $\pi_1(x), \ldots, \pi_{N-1}(x)$ when redistributing grid points, but this is beyond the scope of this paper.

Now that we have developed a criterion for redistributing the grid points, we introduce the following algorithm:

1. Start with a uniform spacing of $x_0, y_1, \ldots, y_N, x_N$, where $x_0 = a$ and $x_N = b$ are the boundary points as before.

2. Compute the extrema of each $\pi_1(x), \ldots, \pi_{N-1}(x)$ using the Newton-Raphson method, assuming convergence from an initial point in the centre of the domain, and label them $x_1, \ldots, x_{N-1}$ accordingly.

3. Evaluate the amplitudes $|\pi_1(x_0)|, |\pi_1(x_1)|, \cdots, |\pi_{N-1}(x_{N-1})|, |\pi_{N-1}(x_N)|$. Denote $h(k) = 1/(2k-1)$, then redistribute the y_i as follows:

   (a) If $|\pi_1(x_0)| > |\pi_1(x_1)|$, decrease y_1 by $(y_1 - x_0)h(k)$ and increase y_1 by $(x_1 - y_1)h(k)$ otherwise.

   (b) For $i = 1, \ldots, N-1$, if $|\pi_i(x_i)| > |\pi_{i+1}(x_{i+1})|$, decrease y_{i+1} by $(y_{i+1} - x_i)h(k)$ and increase y_{i+1} by $(x_{i+1} - y_{i+1})h(k)$ otherwise.

(c) If $|\pi_{N-1}(x_{N-1})| > |\pi_{N-1}(x_N)|$, decrease y_N by $(y_N - x_{N-1})h(k)$ and increase y_N by $(x_N - y_N)h(k)$ otherwise.

4. Repeat steps 2 and 3 for $k = 1, 2, \ldots, m$, where m is the maximum allowable number of iterations, until the amplitudes are found to converge significantly (typically aiming to a fraction of a percent). Alternately, a criterion can be set, terminating the algorithm one the errors in the amplitudes have reached sufficient accuracy.

The choice of step-size used when redistributing the grid points $y_1, \ldots, y_N$ is significant, as many approaches were trialled, with very few converging successfully within a reasonable number of iterations. To start with, the factors of the form $(x_{i+1} - y_{i+1})$, and similar, control the maximum allowable step size for a given iteration. This is to prevent the possibility of two grid points passing where we could possibly have $y_i > y_{i+1}$ for some i, which would not be permissible.

Secondly, the choice of $h(k)$ is equally vital to ensure a reasonable rate of convergence. As the nodes close to the boundaries start further away from their limiting location, a large step size for the first few iterations is preferred, as it allows these points to enter the neighbourhood of their limiting position at a much lower iteration number. Other deciding factors in the step size include that we require $\sum_{k=1}^{\infty} h(k)$ to diverge, as it is unknown at the start of the algorithm how far each node may be required to shift by, as well as requiring $h(k) \to 0$ as $k \to \infty$, as we quite clearly wish the step size to reduce in further iterations to allow the nodes to converge.

Trialling this approach in various problems, including, but not limited to, the ODE problem in Section 6.4, the algorithm is found to work well, achieving very small differences in the amplitudes of $\pi_1(x), \ldots, \pi_{N-1}(x)$, to the order of 10^{-16} for $N = 100$ and $q = 10$ within 5000 iterations; a relative error of 0.47% (see Appendix A).

In practice, the rate of convergence is less than desired, with 5000 iterations for $N = 100$ and $q = 10$ taking a few minutes on an Intel i7 processor, significantly slower than the algorithm used by Hermanns and Hernández [14]. This could definitely be an area of improvement, with future work on the choices of step sizes may achieve markedly better performance, or by a different initial distribution of grid points $y_1, \ldots, y_N$. Although the current approach does not directly result in lost accuracy, it does call into question the usefulness of the approach when a uniform grid of a greater density of points may outweigh any benefits of the grid proposed by Hermanns and Hernández while achieving similar execution times.

Despite its shortcomings, the execution time is reasonable for most problems and the algorithm is found to be accurate, achieving relative errors of $< 1\%$ when $N = O(100)$ and $q = O(10)$ within a few minutes, so any conclusions on the accuracy of the scheme using this algorithm are still within reason. For this reason, all plots in this paper which rely on this type of spacing will use at least 5000 iterations to ensure high levels of accuracy.

6.4 Example Problem

Now that we have introduced and developed an algorithm for computing the scheme, it would be of interest to test our scheme against a challenging problem, for which we can find an analytic solution.

Consider the boundary value ODE problem from Section 2.4:

$$\varepsilon \frac{d^2u}{dx^2} + \frac{du}{dx} = 1 + 2x, \quad 0 < x < 1, \quad \varepsilon \ll 1 \tag{6.7}$$

with boundary conditions,

$$u(0) = 0, \quad u(1) = 1 \tag{6.8}$$

This problem is especially interesting for small values of ε, as a discontinuity appears in the solution as $\varepsilon \to 0$ (see Figure 6.5 (a) and Appendix B), and so will make a difficult test case for the Hermanns-Hernández spacing when epsilon is small [19].

We make use of the method outlined in Section 6.1 on the domain $[0, 1]$ to obtain a collection of functions $I_i(x)$ on the domains $\Omega_i = [x_{i-1/2}, x_{i+1/2})$ for $i = 0, ..., N$, where the endpoints $x_{i\pm1/2}$ and the interpolation points x_i are given as in Section 6.2. These interpolants can then be expressed as a dot product $I_i(x) = L_i(x).u$, where,

$$L_i(x) = (0, ..., 0, l_{i,s_i}(x), ..., l_{i,s_i+q}, 0, ..., 0)^T \tag{6.9}$$

with $l_{i,s_i}(x)$ being in the $(s_i + 1)$-th entry.

Using (6.9), we can then build a new vector function $\mathcal{L}(x)$, where,

$$\mathcal{L}(x) = \begin{cases} L_0(x), & x \in \Omega_0 \\ \vdots & \\ L_N(x), & x \in \Omega_N \end{cases} \tag{6.10}$$

which simply maps x into its respective function $L_i(x)$, yielding the approximation $u(x) \approx \mathcal{L}(x).u$ built from the piecewise interpolation of $u(x)$.

Using this approximation, we can then express the ODE in the following form,

$$(\varepsilon\mathcal{L}''(x) + \mathcal{L}'(x)).u = 1 + 2x \tag{6.11}$$

where $\mathcal{L}'(x)$ and $\mathcal{L}''(x)$ simply denotes the element-wise derivatives of $\mathcal{L}(x)$ [2].

As the elements of u are defined to be $u_i = u(x_i)$, we can build a complete linear system of equations $Au = b$, by denoting,

$$A = \begin{bmatrix} e_1 \\ \varepsilon\mathcal{L}''(x_1)^T + \mathcal{L}'(x_1)^T \\ \vdots \\ \varepsilon\mathcal{L}''(x_{N-1})^T + \mathcal{L}'(x_{N-1})^T \\ e_{N+1} \end{bmatrix} \tag{6.12}$$

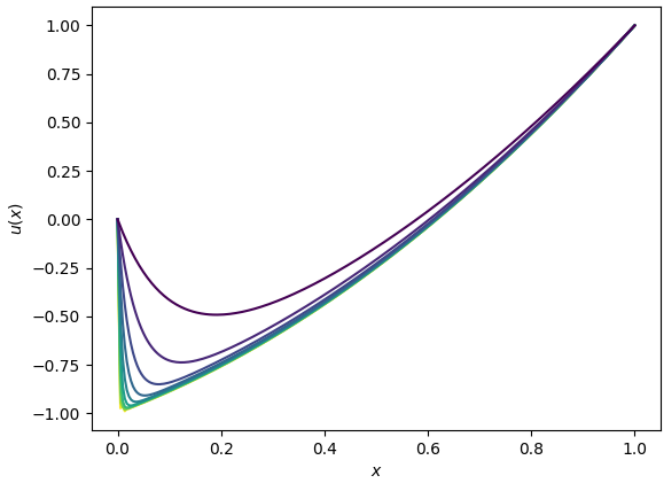


Figure 6.1: Numerical solutions to the ODE problem (6.7) obtained using the outlined method with $N = 250$ and $q = 4$ using a non-uniform grid spacing for varying ε from $\varepsilon = 10^{-3}$(green) to 10^{-1}(purple).

and,

$$b = (0, 1 + 2x_1, \ldots, 1 + 2x_{N-1}, u_N) \tag{6.13}$$

Inverting A, we then have a discrete solution $u = A^{-1}b$, which can be solved by a computer in a variety of ways. For our work, we make use of a PLU decomposition, which works for dense matrices with relative ease. There is a distinct possibility that this method introduces errors into our solution [20], so in future work, it may be beneficial to look into whether these errors are meaningful and if other methods provide any substantial improvements in speed or accuracy.

In Figure 6.1 we can see the solutions obtained using the above method for various values of ε.

Now that we can find numerical solutions using this method, it would be wise to ensure they are accurate and to compare them to other potential grids that we could have chosen. To start, we first need an analytic solution found by hand, which we can then use to evaluate our numerical solutions, as is derived in Appendix B.

Beginning with the accuracy of the scheme, we can vary N and q to see how the errors of the numerical solution changes. From Figure 6.2 (b) we can observe that as N increases, the errors away from $x = 0$ decay rapidly as larger values of N reduce the oscillations in the solution seen in Figure 6.2 (a). The errors around $x = 0$ are less affected by the increase in N, which is to do with our choice of grid spacing as we will explore later in Section 6.5.

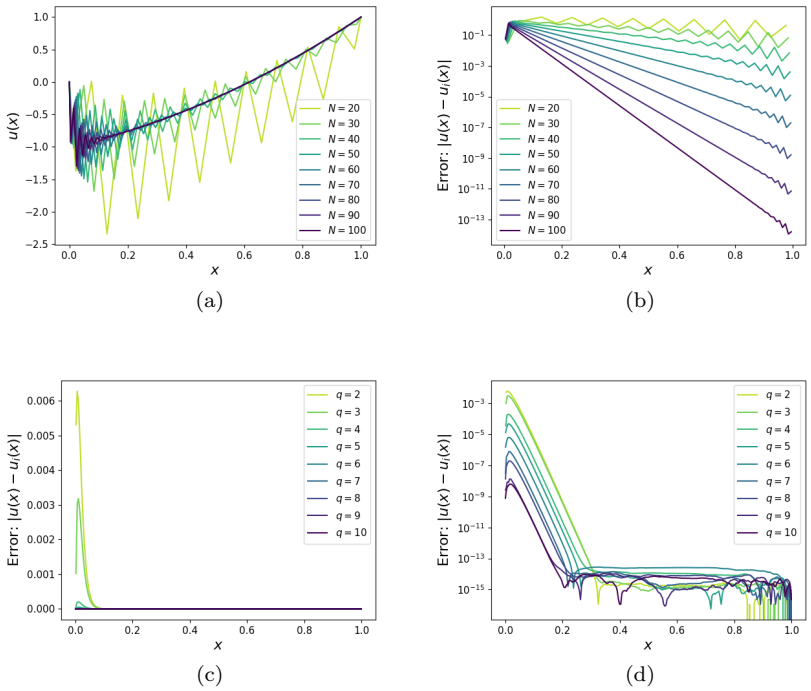


Figure 6.2: (a) Numerical solutions to the ODE problem for $\varepsilon = 10^{-3}$ using various values of N and $q = 4$. Plots of the errors of numerical solutions compared to the analytic solution from Appendix B using: (b) $\varepsilon = 10^{-3}$ and $q = 4$ for various values of N and $\varepsilon = 10^{-2}$ and $N = 250$ for various values of q on: (c) a linear scale and (d) a logarithmic scale. In (b), (c), and (d), the errors at the boundary nodes are omitted as they are set using boundary conditions.

We see similar behaviour in Figures 6.2 (c) and (d) when we increase q, the order of the finite difference scheme. In Figure 6.2 (c), it is clear to see the rapid convergence towards the true solution as q increases, which is made more apparent in Figure 6.2 (d), where for values of $x < 0.3$ we see rapid decay in errors as q increases, and beyond $x > 0.3$ the errors are stable at approximately machine precision.

Now that we have looked at how the scheme converges towards the true solution of this problem with both increasing N and q, we can now evaluate this scheme further by comparing to other possible choices of grids.

6.5 Evaluation of the Scheme

Earlier in Section 6.2 we defined the Hermanns-Hernández spacing by finding the grid which gives all of the π-factors (6.4) to have equal amplitudes. We didn't speak in too much detail why this is desirable and gives the smallest errors as this is already covered in great detail in the paper by Hermanns and Hernández and for the spectral collocation method in Chapter 5 [14]. However, we will briefly look into this now, as the reasoning for why they claim their choice of spacing leads to minimal errors may help us to see where this grid spacing is expected to perform well.

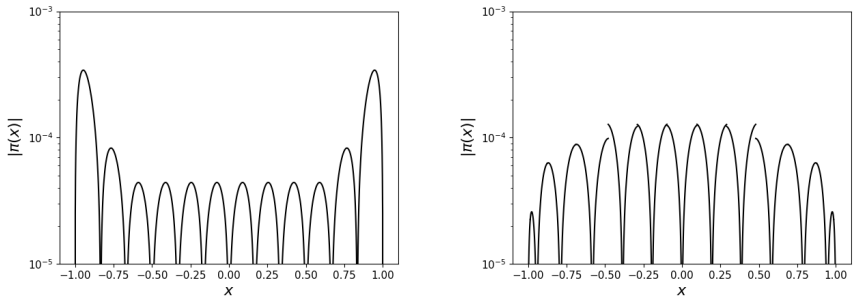
Firstly, we introduce the error of each interpolant from (6.1) [14]:

$$\varepsilon_i(x) = u(x,t) - I_i(x), \quad \forall x \in \Omega_i \tag{6.14}$$

Then as we are using Lagrange interpolation, under the assumption that $u(x,t)$ is sufficiently smooth, the error can be simplified to,

$$\varepsilon_i(x) = \pi_i(x)\frac{u^{(q+1)}(\xi)}{(q+1)!} \tag{6.15}$$

where $\xi = \xi(x) \in [x_{s_i}, x_{s_i+q}]$ [14].



(a) (b)

Figure 6.3: Plots of $|\pi_i(x)|$ for $N = 12$ and $q = 6$ using: (a) a uniform grid and (b) the Hermanns-Hernández grid.

Clearly we cannot control the $(q+1)$-st derivative of u, which leads us to only being able to influence the choice of grid spacing, which will affect the amplitudes of the π-factors. Earlier we chose the grid to lead to equal amplitudes for all of the π-factors. We see in Figure 6.3 (a) that a uniform spacing leads to amplitudes of increasing size as we approach the boundaries, whereas with the Hermanns-Hernández spacing in Figure 6.3 (b) the amplitudes remain consistently low. Clearly this choice of grid mitigates the affects of Runge's phenomenon, and leads to a minimal interpolation error for an unknown function $u(x,t)$ [14].

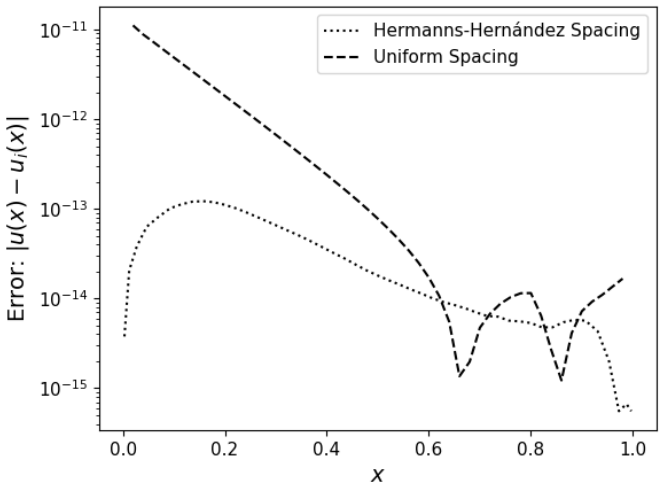
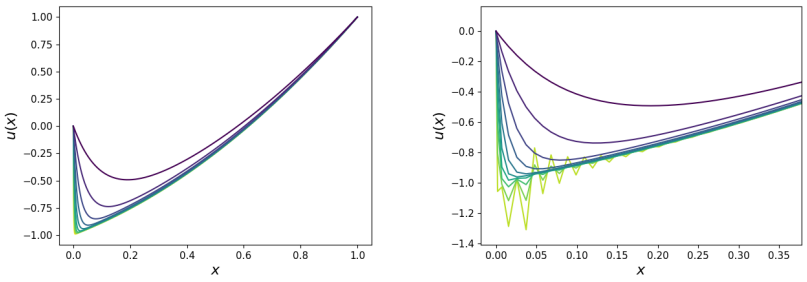


Figure 6.4: Plot comparing the errors of the numerical solutions to the ODE problem (6.7) using the Hermanns-Hernández and uniform grid spacings, with $N = 50$, $q = 12$, and $\varepsilon = 0.1$. Errors are omitted at the boundary as the numerical solutions here are set by boundary conditions.

In Figure 6.4, we can see the absolute error plotted for both the Hermanns-Hernández and uniform spacing for $N = 50$ and $q = 12$. Clearly, the Hermanns-Hernández spacing is substantially more accurate than the uniform grid, and for the few points between $x = 0.6$ and $x = 1.0$ where the uniform grid achieves lower errors, we are already accurate to machine precision so these errors are drastically less important in the overall accuracy of the scheme. As we have just mentioned with the derivation of this scheme though minimising interpolation errors, a more broad comparison is done in Chapter 5 for the grid spacings mentioned to this point.

Earlier we mentioned that, for this problem, we experience large errors close to the boundary at $x = 0$. This is a result of the discontinuity which begins to develop at $x = 0$ as ε becomes small, as seen in Figure 6.5 (a). In Figure 6.5 (b) we see the developing discontinuity induce numerical instability and oscillations in its proximity. The reason for this behaviour is that the choice of grid places too few nodes within the steep region, so there are insufficient samples of the solution to generate a smooth numerical solution near to the discontinuity [19].

To overcome this, an improved grid, specific to the problem, would have to be designed, whereby a higher density of nodes will be placed in proximity to the discontinuity to reduce the amplitude of the oscillations [19]. This is now explored in Chapter 7.



(a) (b)

Figure 6.5: (a) Analytic solutions to the ODE problem (6.7) from Appendix B for varying ε from $\varepsilon = 10^{-3}$ (green) to 10^{-1} (purple). (b) Numerical solutions to the ODE problem obtained using the outlined method with $N = 100$ and $q = 10$ using a non-uniform grid spacing for varying ε from $\varepsilon = 10^{-3}$ (green) to 10^{-1} (purple).

Chapter 7

Grid Stretching

Moe Okawara

7.1 Introduction to Boundary Layer Problems

As discussed at the end of Chapter 6, it is clear that there is a sudden jump in our solution near the boundary at $x = 0$, which eventually develops into a discontinuity as ϵ becomes small. This rapid jump causes problems in our numerical solution because it means we are more prone to errors in this region because of its rapidly changing nature.

To briefly explain this phenomenon, note that this ODE is a steady-state convection-diffusion equation (a convection-diffusion equation is of the form $\kappa\frac{\partial^2 u}{\partial x^2} - U\frac{\partial u}{\partial x} = \frac{\partial u}{\partial t}$ and when the partial time derivative is a constant it is a steady-state equation). Convection is characterised by the first derivative term and diffusion by the second derivative. The region near the boundary where the value of u changes rapidly is known as the boundary layer. When the Péclet number, which measures the ratio of the convection term to the diffusion term, is significantly large (i.e. when ϵ is significantly smaller than 1 like in our ODE), we can think of the substance moving as a whole with some velocity without spreading out too much. This is a problem because if we have two different-valued boundary conditions for the concentration but with low diffusion, then there is bound to be a jump near one of those boundary conditions. This can also be thought of as convergence towards a first-order ODE as ϵ approaches 0. A first-order ODE can essentially be solved with just one boundary condition so having two would mean we get an unnatural, discontinuous solution. [21]

As mentioned earlier, this causes a problem in our numerical solution because it means that in the boundary layer, we would need a high mesh density to account for the rapid changes in this region. However, increasing the mesh density everywhere would be a waste of effort as the solution outside the boundary layer is relatively mellow and smooth, so an adequate numerical approximation can be found with a much less dense grid spacing. If we take a look at Figure

6.2, we can see that as the number of grid points is increased, for larger values of x, the solution starts to converge, but even with large N, there is a big oscillation near $x = 0$, the boundary layer.

7.2 Grid Mappings

One way to go around this problem is to use grid stretching, where we use a transformation of the original grid to create a new spacing which is highly concentrated within the boundary layer. To do so, we want to find a mapping $x = g(\eta)$, where η represents the initial uniform grid spacing. For example, the mapping, $x = \eta^2$ would transform a uniform spacing $0.00, 0.25, 0.75, 1.00$ into $0.00, 0.625, 0.5625, 1.00$. Obviously, this does not work for the ODE we are working with so we work on trying to find a specific mapping.

We would like to find a mapping which has a shallow gradient near $x = 0$ and a steeper (but not too steep) one outside the boundary layer. This actually proves to be quite difficult because many functions will hold one property but not the other. It would also be desirable that $g(0) = 0$ and $g(1) = 1$ such that the domain of the problem stays consistent. However, note that if needed, a second mapping can always be applied to map the transformed domain to $[0, 1]$. [22]

It has been shown by Thompson and Mastin that the hyperbolic sine, hyperbolic tangent and error function are particularly nice building block functions for this problem in terms of the properties just discussed. More notably, sinh is particularly useful for the boundary layer properties while the latter two (especially erf) are useful for outside the boundary layer. [23]

Specific mappings each involving these functions have been formulated in the paper by Thompson and Mastin and are given by:

$$g(\eta) = \frac{\sinh(a\eta)}{\sinh(a)}, \tag{7.1}$$

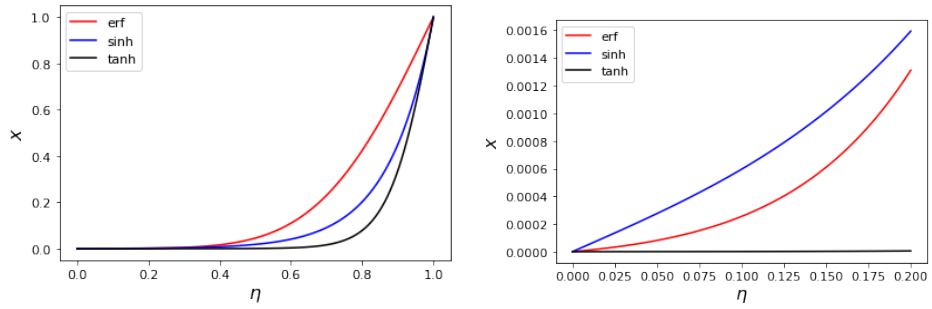
$$g(\eta) = 1 - \frac{\tanh(b(1-\eta))}{\tanh(b)}, \tag{7.2}$$

$$g(\eta) = 1 - \frac{\operatorname{erf}(\sqrt{c}(1-\eta))}{\operatorname{erf}(\sqrt{c})} \tag{7.3}$$

where a, b, c are constants that we can choose. It is important to note that for any a, b, c, we still have that $g(0) = 0$ and $g(1) = 1$, which saves us from having to apply another transformation. [23, 24]

These constants play a big role in our transformation because they determine the gradient of the mappings in and outside the boundary layer, which is the main criteria we focus on when looking for a mapping. For each of the three mappings (7.1), (7.2) and (7.3), increasing the constants decreases the rate of change of the mapping in the boundary layer (which is desired), but also increases the rate of change outside the boundary layer (which is not desired). As a result of this, the mapping stays shallow for a greater range of η and thus

creates a sharper graph because it is still required that $x = 1$ when $\eta = 1$. This can be seen with the example of the erf mapping in Figure 7.2.



(a) (b)

Figure 7.1: (a) Plot comparing mapping (7.3) (erf), mapping (7.1) (sinh) and mapping (7.2) (tanh) for $c = 8$. (b) Zoomed in plot focused around the boundary layer comparing the erf, sinh and tanh mappings for $c = 8$.

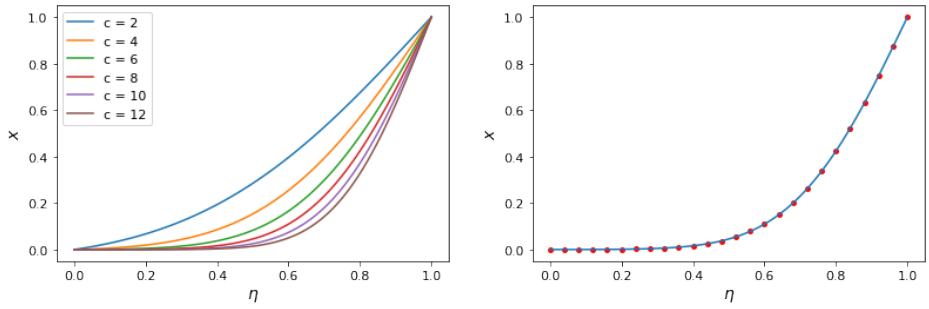
In order for us to be able to decide between the three, we compare the graphs of each mapping for fixed constant values. For example, when we have that $a = b = c = 8$, we get the graph in Figure 7.1 (a).

We can see that the error function does well outside the boundary layer; the sinh and tanh mappings are relatively steep for larger values of η, which as mentioned earlier, is not ideal. However, Figure 7.1 (a) also shows that the error function is the least shallow within the boundary layer, which is not ideal. Although, if we take a look at Figure 7.1 (b), we can actually see that for the smallest values of η, the error function actually does a lot better than the sinh function. This is actually alright, if not better, because the boundary layer of this problem is extremely small, so having a shallow graph for only the smallest values of η is much better. If we look at the tanh function, the increase in x is so small, that the change is not even observable in the range $[0, 0.2]$. This is overkill as it would mean that it would be too dense within the boundary layer. Therefore, in the case of $a = b = c = 8$, we can see that the error function mapping is the optimum out of the three.

In fact, if we change the value of the constant and compare the three graphs in the same way, we see the same trends. For this reason, we will use the erf mapping in Equation (7.3) to solve our ODE problem.

Now that we have decided on the mapping, we can compare the mapping for different values of c. If we look at Figure 7.2, we can see that any c values between 6 and 10 would suffice. The more extreme the values of c are, the less desirable the mapping, since they either get too steep outside the boundary layer or are not shallow enough within the boundary layer.

By plotting the mapping for $c = 8$ as in Figure 7.2, we can see how the uniform grid points of η, represented on the x-axis, compare to the grid points of x, which are represented on the y-axis. The step sizes of the points in the



(a) (b)

Figure 7.2: (a) The erf mappings for different values of c. (b) The erf mapping for $c = 8$.

boundary layer are much smaller than the step sizes outside the boundary layer.

7.3 Transforming the ODE Problem

Once we have find a mapping $x = g(\eta)$, we can plug this into (2.16) to obtain a new transformed ODE in terms of u and η.

To find this transformed ODE, let $\eta = q(x)$ (i.e. the inverse of g, which exists since our g is bijective). Using the chain rule, we get that:

$$\frac{d}{dx} = \frac{d}{d\eta}\frac{d\eta}{dx} = q'(x)\frac{d}{d\eta} \tag{7.4}$$

$$\frac{d^2}{dx^2} = q'(x)(q'(x)\frac{d^2}{d\eta^2} + \frac{dq'}{d\eta}\frac{d}{d\eta}) \tag{7.5}$$

By differentiating (7.3), we get:

$$g'(\eta) = \frac{2\sqrt{c}}{\sqrt{\pi}erf(\sqrt{c})}e^{-c(1-\eta)^2} \tag{7.6}$$

We can then easily calculate $q'(x)$ and $\frac{dq'}{d\eta}$:

$$q'(x) = \frac{1}{g'(\eta)} = \frac{\sqrt{\pi}erf(\sqrt{c})}{2\sqrt{c}}e^{c(1-\eta)^2} \tag{7.7}$$

which gives us

$$\frac{dq'}{d\eta} = -\sqrt{c\pi}erf(\sqrt{c})(1-\eta)e^{c(1-\eta)^2} \tag{7.8}$$

If we plug in all these derivatives into (2.16), we get our final transformed ODE:

$$\begin{aligned}\varepsilon\frac{\pi}{4c}(erf(\sqrt{c}))^2 e^{2c(1-\eta)^2}\frac{d^2u}{d\eta^2} \\ + \frac{\sqrt{\pi}erf(\sqrt{c})}{2\sqrt{c}}e^{c(1-\eta)^2}(1-\varepsilon\sqrt{c\pi}erf(\sqrt{c})(1-\eta)e^{c(1-\eta)^2})\frac{du}{d\eta} = 3 \\ - 2\frac{erf(\sqrt{c}(1-\eta))}{erf(\sqrt{c})}\end{aligned} \tag{7.9}$$

with restrictions

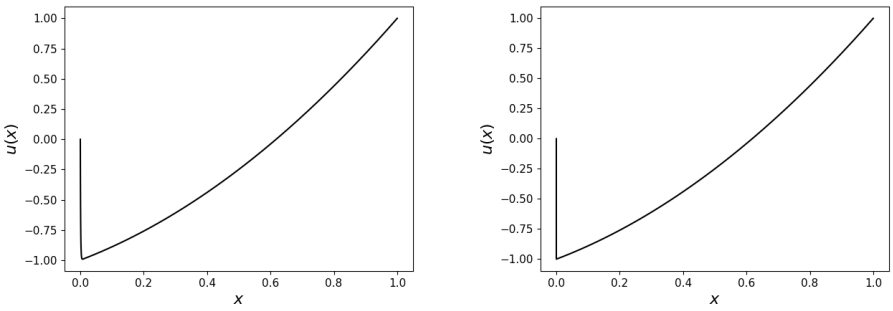
$$0 < \eta < 1, \quad \varepsilon \ll 1 \tag{7.10}$$

and boundary conditions

$$u(0) = 0, \quad u(1) = 1 \tag{7.11}$$

Now that we have an ODE, we can use python to solve the equation numerically with uniform grid spacing using finite differences. This code can be found in Appendix A.

For simplicity, we have chosen $c = 8$ as our constant as this value seems to give an appropriate balance between the behaviour of the mapping inside and outside the boundary layer. Figures 7.3 and 7.4 show the obtained solutions for number of points $N = 100$ and order $q = 10$. If we compare the numerical solutions obtained using the Hermanns-Hernández spacing for $N = 100$ in Figure 6.5, we can see that there is significantly less instability within the boundary layer. When $N = 100$ and $\varepsilon = 10^{-5}$, the wiggles are no longer noticeable, a big improvement considering we have only used $N = 100$.



(a) (b)

Figure 7.3: (a) Numerical solution for the ODE with $\epsilon = 10^{-3}$ obtained by grid stretching. (b) Numerical solution for the ODE with $\epsilon = 10^{-5}$ obtained by grid stretching.

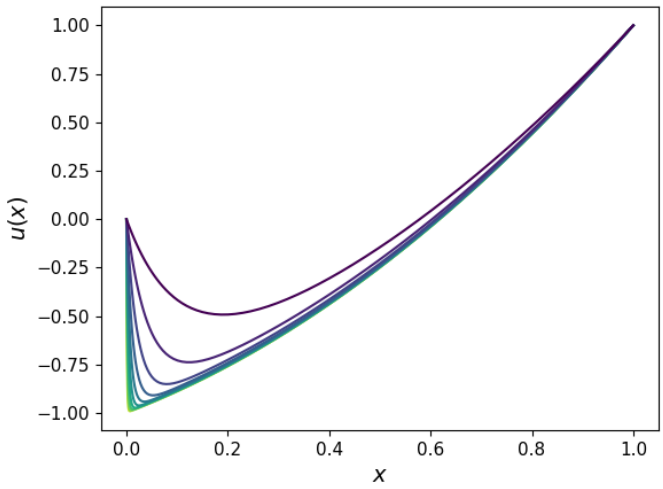


Figure 7.4: Numerical solution for the ODE for different values of ϵ varying from 10^{-1} to 10^{-3} with decreasing ϵ starting from the purple to green lines. We have taken N = 100 and q = 10 here.

We can compare the solutions obtained by Hermanns-Hernández spacing, uniform spacing and grid stretching by looking at their errors side by side as in Figure 7.5. We can see that there is a significant decrease in error for smaller values of x using the grid-stretching compared to the other two methods. The grid-stretching error in this boundary layer region is roughly 10^6 times smaller than the Hermanns-Hernández spacing. We may also try to analyze the error for larger values of x, but it is important to note that for these values, the error is within machine epsilon, so they are not really of significance.

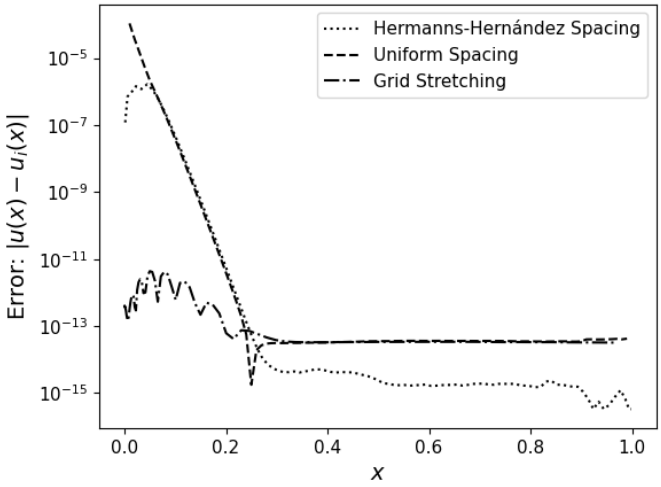


Figure 7.5: Plot showing the errors of the uniform, Hermanns-Hernández spacing and grid-stretched spacing compared to the exact analytical solution with $\epsilon = 10^{-3}$, $N = 100$ and $q = 12$. Note that the y-axis is log-scaled.

Chapter 8

Conclusions

William Taylor, Niall Oswald, Moe Okawara, Yaofeng Lan, and Nicholas Matheou

We have shown the utility of finite difference approximations to provide solutions to differential equations to a variety of accuracies - dependent on grid choice. Additionally, understanding of the efficacy of grid choice has been extended to include non-uniform grid spacing, with techniques evolving beyond the uniform grid spacings offered by finite differences calculated with Fornberg's algorithm. Our work also displays how vital iterative code is for numerical analysis, as evidenced by the extensive repository found in Appendix A.

Also, we've shown how the finite difference methods on a uniform grid can be used to solve challenging problems, like the Bratu problem. A non-standard finite difference method has been shown to give highly accurate results together with the standard finite difference method.

Having investigated Runge's phenomenon for a specific function, we have laid out an explanation for its occurrence in high order polynomial interpolation. Giving reasoning and a method to compute the rate of convergence and divergence, and their corresponding domains.

In order to minimise the effect of Runge phenomenon, we apply Chebyshev roots as grids to minimise the maximal error of finite difference methods. Both proved theoretically and demonstrated by approximating ODE (2.16), Chebyshev roots decrease significantly the errors of finite difference methods. And we can notice from applying spectral collocation method with Chebyshev polynomials of first kind that Chebyshev polynomials do possess great properties when it comes to numerical analysis. As we demonstrated through approximating ODE 2.16, spectral collocation method using Chebyshev polynomials of first kind obtains much smaller errors than the finite difference method does.

Despite the benefits of using Chebyshev polynomials, we find that at times the inability to separate the accuracy of the scheme from the grid size to be restrictive, resulting in slow computations with larger grids. However, having

witnessed the affects of Runge's phenomenon and how it can affect our numerical solutions, we still aim to minimise the interpolation errors even when $q < N$. We outline the Hermanns-Hernández grid spacing, which minimises the error in this case.

We also introduce an algorithm for computing the grid spacing for arbitrary grid sizes and degrees. However, although this algorithm is found to be accurate, at times it can be rather slow, with more work needed both theoretically and experimentally to make efficiency improvements and guarantees on convergence.

When attempting to test the Hermanns-Hernández scheme on an especially difficult ODE problem (6.7), we discover drawbacks in our choice of grid spacing, with large errors resulting from steep slopes in the solution when ε is small. To overcome this we investigate the use of grid stretching to seek more accurate solutions.

The implementation of grid-stretching using an error function based mapping, significantly decreased the error near $x = 0$, improving on the Hermanns-Hernández scheme. However, it is important to note that the choice of mapping is not necessarily unique and therefore it may also be interesting to explore other mappings such as a piecewise mapping which optimises grid-stretching according to different regions of x.

Overall, we have provided a range of grid spacings that can be used with a multitude of finite difference schemes. We have experimented in using these schemes to solve a challenging ODE problem, which helped to motivate and guide our search though the possible approaches, with the aim of achieving accurate numerical solutions in an efficient manner.

Appendix A

Code and Algorithms

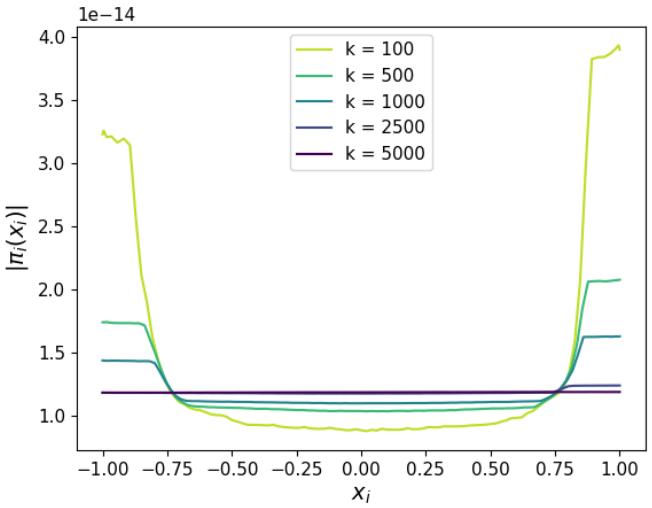


Figure A.1: $|\pi_i(x_i)|$ evaluated using the k-th iteration of the algorithm outlined in Section 6.3 for $N = 100$ and $q = 10$.

The code and notebooks used to generate the plots found throughout this paper are given in the GitHub repository at: https://github.com/NiallOswald/high-order-finite-differences.

With regards to the algorithm mentioned in Section 6.3: in Figure A.1, we can see the amplitudes $\pi_i(x_i)$ plotted for various numbers of iterations ranging from $k = 100$ to $k = 5000$. A relative error of 0.47% was found after 5000 iterations, which was returned separately. This is representative of other tests, with errors converging to zero with increasing iterations.

Appendix B

Analytic Solution to the ODE Problem

Consider the boundary value ODE problem:

$$\varepsilon \frac{d^2u}{dx^2} + \frac{du}{dx} = 1 + 2x, \quad 0 < x < 1, \quad \varepsilon \ll 1 \tag{B.1}$$

with boundary conditions,

$$u(0) = 0, \quad u(1) = 1 \tag{B.2}$$

This is a second-order linear non-homogeneous ODE with constant coefficients, and so can be easily solved using its auxiliary equation [25]. Consider the characteristic equation of the ODE,

$$\varepsilon\lambda^2 + \lambda = 0$$

which has roots $\lambda = 0, -\varepsilon^{-1}$, giving the complementary function,

$$u_{CF} = A + Be^{-x\varepsilon^{-1}} \tag{B.3}$$

where A and B are constants of integration. Now consider the ansatz $u = ax^2 + bx$. Substituting this into (B.1), we obtain,

$$2\varepsilon a + b + 2ax = 1 + 2x$$

Comparing coefficients, we have $a = 1$ and $b = 1 - 2\varepsilon$, giving the particular integral,

$$u_{PI} = x^2 + (1 - 2\varepsilon)x \tag{B.4}$$

Combining (B.3) and (B.4) gives the general solution,

$$u = A + Be^{-x\varepsilon^{-1}} + x^2 + (1 - 2\varepsilon)x \tag{B.5}$$

To find the constants of integration, it a simple exercise to substitute the boundary conditions (B.2) into (B.5), yielding the particular solution,

$$u = \frac{2\varepsilon - 1}{1 - e^{-\varepsilon^{-1}}}(1 - e^{-x\varepsilon^{-1}}) + x^2 + (1 - 2\varepsilon)x \tag{B.6}$$

Appendix C

Proof of Certain Equations of Chebyshev Polynomials

We use (5.6) to prove (5.9):

$$\begin{aligned} T_{n+1}(x) &= \cos((n+1)\cos^{-1}(x)) \\ &= x\cos(n\cos^{-1}(x)) - \sin(n\cos^{-1}(x))\sin(\cos^{-1}(x)) \\ &= x\cos(n\cos^{-1}(x)) - \frac{1}{2}(\cos((n-1)\cos^{-1}(x)) - \cos((n+1)\cos^{-1}(x))) \\ &= xT_n(x) - \frac{1}{2}(T_{n-1}(x) - T_{n+1}(x)) \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \end{aligned} \tag{C.1}$$

Proof of (5.18e) is the following:

$$\begin{aligned} T_n^{(k)} &\overset{(5.7)}{=} \frac{d^k}{dx^k}(2xT_{n-1}(x) - T_{n-2}(x)) \\ &= \frac{d^k}{dx^k}(2xT_{n-1}(x)) - T_{n-2}^k(x) \\ &= 2xT_{n-1}^k(x) + 2kT_{n-1}^{k-1}(x) - T_{n-2}^k(x) \end{aligned} \tag{C.2}$$

Bibliography

[1] Olver S. Differentiation. Imperial College London; 2022. Lecture Notes. Available from: https://github.com/Imperial-MATH50003/MATH50003NumericalAnalysis/blob/main/notebooks/Differentiation.ipynb [cited 15 June 2022].

[2] LeVeque RJ. Finite Difference Approximations. In: Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems. 3600 Market Street, Floor 6, Philadelphia, PA 19104: Society for Industrial and Applied Mathematics SIAM; 2007. p. 3-11.

[3] Sadiq B, Viswanath D. Finite Difference Weights, Spectral Differentiation, and Superconvergence. Mathematics of Computation. 2014;83:2412-3.

[4] Fornberg B. Generation of Finite Difference Formulas on Arbitrarily Spaced Grids. Mathematics of Computation. 1988;51:699-706.

[5] Brodtkorb PA, D'Errico J. numdifftools.fornberg; 2009. Python Module. Available from: https://numdifftools.readthedocs.io/en/latest/_modules/numdifftools/fornberg.html [cited 21 June 2022].

[6] LeVeque RJ. Steady States and Boundary Value Problems. In: Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems. 3600 Market Street, Floor 6, Philadelphia, PA 19104: Society for Industrial and Applied Mathematics SIAM; 2007. p. 13-6.

[7] Kong Q, Siauw T, Bayen AM. Chapter 23 - Boundary-Value Problems for Ordinary Differential Equations (ODEs). In: Kong Q, Siauw T, Bayen AM, editors. Python Programming and Numerical Methods. Academic Press; 2021. p. 399-414. Available from: https://www.sciencedirect.com/science/article/pii/B9780128195499000336.

[8] Buckmire R. On exact and numerical solutions of the one-dimensional planar Bratu problem. Science Direct Working Paper No S1574-0358(04)70978-8; 2003. Available from: https://ssrn.com/abstract=3181448.

[9] Temini H, Ben-Romdhane M. An iterative finite difference method for solving Bratu's problem. Journal of Computational and Applied mathematics. 2016;292:76-82.

[10] Mohson A. A simple solution of the Bratu problem. Computers and Mathematics with Applications. 2014;67:26-33.

[11] Eagan N, Hauser G, Flaherty DT. Newton's Method on a system of Non-Linear Equations; 2014.

[12] Fornberg B. Introduction to PS methods via finite differences. In: A Practical Guide to Pseudospectral Methods. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press; 1996. p. 14–35.

[13] Runge C. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. Zeitschrift fur Mathematik und Physik. 1901;46:224-43.

[14] Hermanns M, Hernández JA. Stable high-order finite-difference methods based on non-uniform grid point distributions. International Journal for Numerical Methods in Fluids. 2008;56(3):233-55.

[15] Hundley DR. Chebyshev's Theorem. Whitman College; 2006. Lecture Notes. Available from: http://people.whitman.edu/~hundledr/courses/M467F06/ChebInterp.pdf.

[16] Cai Q, Song L. The Chebyshev interpolation polynomial algorithm error analysis. In: 2012 IEEE International Conference on Information Science and Technology; 2012. p. 745-8.

[17] Hussaini MY, Kopriva DA, Patera AT. Spectral collocation methods. Applied Numerical Mathematics. 1989;5(3):177-208. Available from: https://www.sciencedirect.com/science/article/pii/0168927489900330.

[18] Storm JL. Construction of Chebyshev Polynomials. Lubbock, TX: Texas Tech University; 1994. Available from: https://ttu-ir.tdl.org/bitstream/handle/2346/60966/31295008378936.pdf?sequence=1.

[19] Gresho PM, Lee RL. Don't suppress the wiggles—They're telling you something! Computers & Fluids. 1981;9(2):223-53.

[20] Olver S. Decompositions and Least Squares. Imperial College London; 2022. Lecture Notes. Available from: https://github.com/Imperial-MATH50003/MATH50003NumericalAnalysis/blob/main/notebooks/Decompositions.ipynb [cited 15 June 2022].

[21] LeVeque RJ. Boundary Value Problems. In: Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems. 3600 Market Street, Floor 6, Philadelphia, PA 19104: Society for Industrial and Applied Mathematics SIAM; 2007. p. 31-4.

[22] Mughal MS. Grids and Clustering. Imperial College London; 2022. Lecture Notes.

[23] Thompson JF, Mastin CW. Order of Difference Expressions in Curvilinear Coordinate Systems. Journal of Fluids Engineering. 1985:5-7.

[24] Nitsche LC. One-Dimensional Stretching Functions For Cˆn Patched Grids, and Associated Truncation Errors in Finite Difference Calculations. Communications in Numerical Methods in Engineering. 1996;12:303-16.

[25] Shahrezaei V. Calculus and Applications - Part II. Imperial College London; 2021. Lecture Notes. Available from: https://bookdown.org/vshahrez/lecture-notes/ [cited 15 June 2022].