Imperial College
London

M2R Project

Group 14

Imperial College London

Department of Mathematics

Higher order finite differences and Runge phenomenon

Author:
Mehul Odedra
Ruxandra Iliescu
Shuaipeng Zhang
Stelios Attipa
Hermine Tranié

Supervisor:
Dr. Shahid Mughal

June 2020

Abstract

In this project, we will be using different methods to solve ordinary differential equations. We are showing the pros and cons of the higher order finite difference method. We are also looking at methods to reduce the Runge phenomenon. We look at where Chebyshev nodes work and don't work, for example they work at range boundaries while decreasing the number of total operations but at internal boundaries they yield more error and should be avoided. At these internal boundaries, an interior layer can occur (depending on the example). We use matching techniques to construct asymptotic expansions. We are also comparing the accuracy of these numerical and asymptotic methods.

Contents

1 Introduction

1.1 Background and motivation

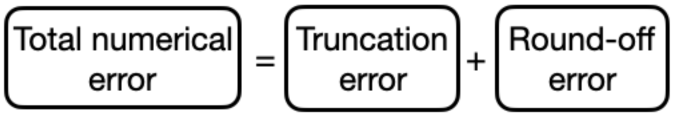
During the academic year, we followed different modules including differential equations and introduction to numerical analysis. The latter introduced us to solving odes numerically. Which is why we wanted to combine the both (numerical and analytical methods) in order to tackle more challenging problems.

1.2 Notation

1.2.1 Order Notation

The order notation is used to describe how closely a finite series approximates a given function.
As it is defined in "Introduction to Perturbation Methods", this is what order notation means:
Let f(z) and g(z) be two functions of a complex number z, defined on a domain D containing a neighborhood of z_0.
If there exist constants k, $\gamma > 0$ such that $|f(z)| \leq k|g(z)|$ for any $0 < |z - z_0| < \gamma$, then we write $f(z) = O(g(z))$ as $z \to z_0$.
If for any $k > 0$, there exists $\gamma > 0$ such that $|f(z)| \leq k|g(z)|$ for any $0 < |z - z_0| < \gamma$, then we write $f(z) = o(g(z))$ as $z \to z_0$.

1.2.2 Numerical Accuracy



Local Truncation Error (LTE): method dependant
- result of an approximation in a method (eg. Taylor series truncation)
Round-off error : machine dependant
- result from an approximation to represent an exact number (eg. $\pi = 3.14$)

2 Higher order finite differences

A finite difference is an approximation of the derivative of a function. This is very useful for numerical solution of boundary value problems. Going to higher order order means that the approximation becomes more accurate, as more points are taken into consideration.

2.1 Deriving the finite difference formulas

There are 3 main methods to derive formulas for finite difference [1]: method of undetermined coefficients, interpolating polynomials and Taylor series. In this project, we will go through the explanation for the latter in details.

2.1.1 Central Difference

Recall the Taylor series expansion about a point x noted i, assuming f is continuously differentiable [2] and points x satisfying:

$$x_{i\pm1} = x_i \pm \Delta x$$

$$f(x_{i+1}) = f(x_i+\Delta x) = f(x_i)+\Delta x f'(x_i)+\frac{\Delta x^2}{2!}f''(x_i)+\frac{\Delta x^3}{3!}f'''(x_i)+O(\Delta x^4) \tag{1}$$

$$f(x_{i-1}) = f(x_i-\Delta x) = f(x_i)-\Delta x f'(x_i)+\frac{\Delta x^2}{2!}f''(x_i)-\frac{\Delta x^3}{3!}f'''(x_i)+O(\Delta x^4) \tag{2}$$

(1) + (2) yields:

$$\begin{aligned} f(x_{i+1}) + f(x_{i-1}) &= 2f(x_i) + \Delta x^2 f''(x_i) + O(\Delta x^4) \\ \implies f''(x_i) &= \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{\Delta x^2} + O(\Delta x^2) \\ \implies f_i'' &= \frac{f_{i-1} - 2f_i + f_{i+1}}{\Delta x^2} \end{aligned}$$

(1) - (2) yields :

$$\begin{aligned} f(x_{i+1}) - f(x_{i-1}) &= 2\Delta x f'(x_i) + O(\Delta x^3) \\ \implies f'(x_i) &= \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} + O(\Delta x^2) \\ \implies f'_i &= \frac{-f_{i-1} + f_{i+1}}{2\Delta x} + O(\Delta x^2) \end{aligned}$$

These formulae are accurate to O(Δx^2). Therefore they are called : second order central finite difference for f.

2.1.2 Forward Difference

This is the 2nd order approximation of the first forward derivative [3]:

$$f'_i = \frac{\alpha_1 f_i + \alpha_2 f_{i+1} + \alpha_3 f_{i+2}}{\Delta x} + O(\Delta x^2) \tag{3}$$

As seen before, the Taylor series expansion around x_i are:

$$\begin{aligned} f_i &= f_i \\ f_{i+1} &= f_i + \Delta x f'_i + \frac{\Delta x^2}{2!} f''_i + \frac{\Delta x^3}{3!} f'''_i + O(\Delta x^4) \\ f_{i+2} &= f_{i+1} + \Delta x f'_{i+1} + \frac{\Delta x^2}{2!} f''_i + \frac{\Delta x^3}{3!} f'''_i + O(\Delta x^4) \\ &= f_i + 2\Delta x f'_i + 2\Delta x^2 f''_i + \frac{4}{3}\Delta x^3 f'''_i + O(\Delta x^4) \end{aligned}$$

Now substitute these Taylor expansion in (3) and re-arrange:

$$\begin{aligned} \frac{\alpha_1 f_i + \alpha_2 f_{i+1} + \alpha_3 f_{i+2}}{\Delta x} =& \frac{\alpha_1 + \alpha_2 + \alpha_3}{\Delta x} f_i \\ &+ (\alpha_2 + 2\alpha_3) f'_i \\ &+ (\frac{\alpha_2}{2} + 2\alpha_3)\Delta x f''_i \\ &+ (\frac{\alpha_2}{6} + \frac{4\alpha_3}{3})\Delta x^2 f'''_i + O(\Delta x^3) \end{aligned}$$

$$\implies \begin{cases} \frac{\alpha_1+\alpha_2+\alpha_3}{\Delta x} = 0 \\ \alpha_2 + 2\alpha_3 = 1 \\ (\frac{\alpha_2}{2} + 2\alpha_3)\Delta x = 0 \end{cases}$$

$$\implies \begin{cases} \alpha_1 = -\frac{3}{2} \\ \alpha_2 = 2 \\ \alpha_3 = -\frac{1}{2} \end{cases}$$

We get the following forward second order finite difference:

$$f_i' = \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2\Delta x}$$
$$f_i'' = \frac{2f_i - 5f_{i+1} + 4f_{i+2} - f_{i+3}}{\Delta x^2}$$

2.1.3 Backwards Difference

Similarly, we get the following backward second order finite difference:

$$f_i' = \frac{f_{i-2} - 4f_{i-1} + 3f_i}{2\Delta x}$$
$$f_i'' = \frac{-f_{i-3} + 4f_{i-2} - 5f_{i-1} + 2f_i}{\Delta x^2}$$

The point of having forward and backward finite difference is so that in case you reach a boundary point for example, you won't be able to go over it, hence that's where the sided difference comes in, allowing to evaluate new coefficients at appropriate nodes.

2.2 Fornberg Algorithm

Clearly the computation these coefficients quickly becomes very tedious when increasing the order of the finite differences. This is why, we use an algorithm that was developed by Fornberg in 1998 (see Appendix) [4]. This algorithm is based on the way weights are derived from the interpolating polynomial method, as follows. One can also note that there are alternatives to derive the weights, however it has been shown that Fornberg's method is the most efficient for Matlab [5].

Consider interpolating polynomial of degree 2

$$L(x) = \sum_{j=-1}^{1} l_j(x) f_j$$

for points

$$(x_{-1}, f_{-1}), (x_0, f_0), (x_1, f_1)$$

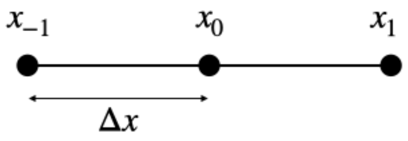
Also by numerical analysis lecture notes,

$$l_j(x) = \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m}$$
$$l_{-1}(x) = \frac{(x - x_0)(x - x_1)}{(x_{-1} - x_0)(x_{-1} - x_1)}$$
$$l_0(x) = \frac{(x - x_{-1})(x - x_1)}{(x_0 - x_{-1})(x_0 - x_1)}$$
$$l_1(x) = \frac{(x - x_{-1})(x - x_0)}{(x_1 - x_{-1})(x_1 - x_0)}$$

Now, evaluating at x_0 our twice differentiated interpolating polynomial $L(x)$, we get:

$$f''(x_0) \approx L''(x_0) = \sum_{j=-1}^{1} l_j''(x_0) f_j$$

Also, note that our points are equidistant, as below:



Hence,

$$\begin{cases} l_{-1}(x) & = \frac{(x-x_0)(x-x_1)}{2\Delta x^2} \\ l_0(x) & = \frac{(x-x_{-1})(x-x_1)}{\Delta x^2} \\ l_1(x) & = \frac{(x-x_{-1})(x-x_0)}{2\Delta x^2} \end{cases}$$

$$\implies \begin{cases} l''_{-1}(x_0) & = \frac{1}{2\Delta x^2} \\ l''_0(x_0) & = -\frac{2}{\Delta x^2} \\ l''_1(x_0) & = \frac{1}{2\Delta x^2} \end{cases} \implies f''_0 = \frac{f_{-1} - 2f_0 + f_1}{\Delta x^2}$$

As expected.

2.3 Benefits of higher order finite differences

[15]

2.3.1 More accurate than the second order method with same number of nodes

We can use Taylor's Expansion for arbitrary function $f(x)$ to derive this. For some h, compute the Taylor' Expansion at $x + h$:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + ...$$

Rearranging the equation, we have

$$\left|\frac{f(x+h)-f(x)}{h} - f'(x)\right| = \left|\frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^2 + ...\right|$$

For the second order finite difference, we have the error O(h) for small h. While the higher order finite difference has the error $O(h^n)$ for some order n. Hence this implies that if we take the same number of nodes in the calculation, as n increases, the error will relatively decreases.

2.3.2 Less computationally demanding

More specific: In our algorithm, the last matrix we solve is tridiagonal. Thus we can make the use of Gaussian elimination to reduce the computational procedure.
Assume the last equation to solve the tridiagonal matrix is in the form of :

$$\begin{pmatrix} a_1 & 0 & 0 & & ... & ... & 0 \\ b_2 & a_2 & c_2 & 0 & ... & ... & 0 \\ 0 & b_3 & a_3 & c_3 & 0 & ... & 0 \\ ... & & ... & ... & ... & & ... \\ ... & & ... & ... & b_{n-1} & a_{n-1} & c_{n-1} \\ 0 & & ... & ... & & 0 & a_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ ... \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ 0 \\ 0 \\ ... \\ 0 \\ y_n \end{pmatrix}$$

We can rewrite this into the equation form with index i, that is, $a_ix_i + b_ix_{i-1} + c_ix_{i+1} = y_i$, where $b_1 = c_n = 0$ for $i \in [1, n]$. Thus we get this system of equations :

$$\begin{cases} a_1x_1 + c_1x_2 = y_1 \\ b_ix_{i-1} + a_ix_i + c_ix_{i+1} = y_i, i \in [2, n-1] \\ b_nx_{n-1} + a_nx_n = y_n \end{cases}$$

Next we can get rid of every x step by step in the way of multiplying proper coefficients on the equations with increasing value of i and making subtractions from the top down.
For example, in the first step we just consider equation 1 and equation 2 with taking $i = 2$. Then we can compute by multiplying equation 1 and 2 by b_2 and a_1 respectively, the result is $a_1 (b_2x_1 + a_2x_2 + c_2x_3) - b_2 (a_1x_1 + c_1x_2) = a_1y_2 - b_2y_1$, a equation with x_1 cancelled.
In this way we can continue to cancel an x_i in the ith step until we comes to the last row. We build an equation with x_n being the only unknown. If the equation can be solved, we will be able to get all the values of x_i, thus solve the equation.
The whole technique just consists of multiplications and additions between equations with only three terms because every row of the matrix is made of at most three non-zero terms. Apparently it is much easier for the computer to get the solution than calculating the inverse of a large matrix and then computing the matrix multiplication.

2.3.3 Can solve complex problems faster

We are able to prove this easily by using the 'tic toc' timing function in MATLAB to compare the time taken for higher order finite difference with that of second order FDM while solving the same problem.

2.4 Costs of higher order finite differences

Using the higher order FDM, there may be numerical instability generated by the strong oscillation closed to the area of boundary conditions, which leads to the topic of Runge Phenomenon.

3 Runge Phenomenon

3.0.1 Weierstrass Approximation Theorem

For every continuous function $f(x)$ defined on a closed interval $[a, b]$, there exists a set of polynomials $P_N(x)$ for $N = 0, 1, 2 \ldots$, each of degree at most N, that approximates $u(x)$ with uniform convergence over the interval as N tends to infinity.

$$\lim_{N\to\infty} (\max_{a\leq x\leq b} |f(x) - P_N(x)|) = 0$$

3.0.2 Global Polynomial Interpolation Theorem

Define a function $u(x)$ on an interval $[a, b]$ and $N+1$ unique nodes: $x_0, x_1, \ldots x_N$. Assume the existence of an Nth order derivative in $[a, b]$ and $(N+1)th$ order derivative in the open interval (a, b). Define the polynomial interpolant $I(x)$ of degree N as follows,

$$I(x) = \sum_{j=0}^{N} l_j(x) u_j$$

$$l_j(x) = \prod_{m=0, m\neq j}^{N} \frac{x - x_m}{x_j - x_m}$$

Then for all x in $[a, b]$, we have the corresponding pointwise interpolation error $\varepsilon(x)$,

$$\varepsilon(x) = u(x) - P_n(x) = \pi(x) \frac{u^{(N+1)}(c)}{(N+1)!}$$

$$\pi(x) = \prod_{m=0}^{N} (x - x_m)$$

where c is in (a, b). In particular the pointwise interpolation error depends on the node distribution (the number N as well as the points chosen) and the regularity of the function being interpolated. [18]

3.0.3 What is the Runge Phenomenon?

In the field of numerical analysis, polynomial interpolation can be used as a method of estimating values between a known data set. This has many applications within mathematics, such as approximating complicated curves and in numerical quadrature. However, the use of high degree polynomials over a set of equispaced nodes is problematic because it can lead to strong oscillations of $I(x)$ at the edges of the interval. These oscillations can easily result in divergent approximations and shows that going to large degrees does not always improve accuracy. This is known as the Runge Phenomenon and was discovered by Carl David Tolmé Runge in 1901. The phenomenon shares similarities with the Gibbs Phenomenon in Fourier series approximation; which states there is always an overshoot by a constant factor that occurs closer and closer to any discontinuities as we increase the number of terms in the Fourier series. For example the function $\frac{\sin(x)}{x}$ has an overshoot by a factor of 1.179 close to $x = 0$. [9]

3.0.4 Example of the Runge Phenomenon

Consider the function $g(x)$,

$$g(x) = \frac{1}{1 + 25x^2}$$

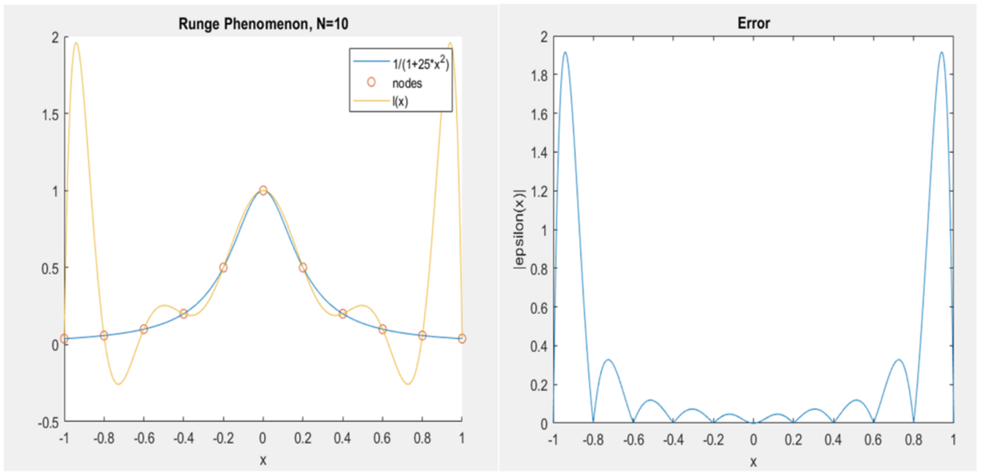
Define an equispaced grid consisting of 11 (N=10) nodes in the interval (-1,1):

$$x_i = \frac{i}{5} - 1$$
$$i = 0, 1, ..., N$$

Global interpolation using the above equispaced grid presents large oscillations of $I(x)$ close to the boundaries [18]. In particular the interpolation error diverges to infinity when the degree of the polynomial is increased.

$$\lim_{N\to\infty} (\max_{-1\le x\le 1} |g(x) - P_N(x)|) = +\infty$$

Note this result contradicts the Weierstrass Approximation Theorem which discusses the existence of *some* sequence of polynomials where as now we are discussing a *specific* method for generating a sequence of polynomials (simply put the P_N in both situations are different). Below we can see a graphical representation [17] of why going to higher degrees at equidistant points can be troublesome.



3.0.5 Why does it arise?

The Runge Phenomenon is caused by two properties; the high order derivative of the function rises very fast and the fact that the nodes are equally spaced. Let us again consider the function $g(x)$. Then we have

$$\varepsilon(x) = \frac{g^{(N+1)}(c)}{(N+1)!}\prod_{m=0}^{N}(x-x_m)$$

for some $c\epsilon(-1,1)$. This implies that

$$\max_{-1\leq x\leq 1}|\varepsilon(x)| \leq \max_{-1\leq x\leq 1}\frac{\left|g^{(N+1)}(x)\right|}{(N+1)!}\max_{-1\leq x\leq 1}\prod_{m=0}^{N}|x-x_m|$$

It is easy to prove that with equally spaced nodes

$$\max_{-1\leq x\leq 1}\prod_{m=0}^{N}|x-x_m| \leq N!h^{N+1}$$

where $h=\frac{2}{N}$ is the step size. The magnitude of the $(N+1)th$ derivative of $g(x)$ grows rapidly as we increase the order. It can be calculated [18] that

$$\max_{-1\leq x\leq 1} g^{(11)}(x) = 1.77219\times 10^{15}$$

In particular we can bound the high order derivative of $g(x)$ by a constant which depends on N,

$$\max_{-1\leq x\leq 1}\left|g^{(N+1)}(x)\right| \leq (N+1)!5^{N+1}$$

This gives an upper bound for $\varepsilon(x)$,

$$\max_{-1\leq x\leq 1}|\varepsilon(x)| \leq \frac{(N+1)!5^{N+1}}{(N+1)!}N!(\frac{2}{N})^{N+1} = (\frac{10}{N})^{N+1}N! \to \infty$$

as $N \to \infty$. Note this is merely an explanation, and not a formal proof, since we've only shown that the upper bound of the error diverges (and this does not imply that the error itself tends to infinity). The property of the high order derivative is important. For example suppose for an arbitrary function $u(x)$ the following holds for all $N > 0$,

$$\max_{a\leq x\leq b}\left|u^{(N+1)}(x)\right| \leq M$$

where M is a constant that does not depend on N (the derivative does not diverge like it did for $g(x)$). Then

$$\max_{a\leq x\leq b}|\varepsilon(x)| \leq \frac{M}{(N+1)!}(b-a)^{N+1} \to 0$$

as $N \to \infty$. This of course contradicts the Runge Phenomenon. [18]

3.1 Method to eliminate this phenomenon

There are three main ways to reduce the Runge phenomena. The first being the trivial solution to use a lower degree polynomial since the higher order is the issue. However, although this increases the accuracy at the boundaries, we want to increase the accuracy everywhere which we can't do with 2nd order methods. The second solution is using piecewise polynomials. Instead of increasing the degree of of our interpolating polynomial, we can increase the number of polynomial pieces and this decreases the error term. The third option is to pick non-uniformly spaced nodes such as Chebyshev nodes. Increasing the nodes at the boundaries stabilizes the solution.

3.1.1 Piecewise Polynomial Interpolation

Define interpolating polynomials of degree $q < N$:

$$I_i(x) = \sum_{j=s_i}^{s_i+q} l_{ij}(x) u_j$$

$$l_{ij}(x) = \prod_{m=0, s_i+m \neq j}^{q} \frac{x - x_{s_i+m}}{x_j - x_{s_i+m}}$$

where $x \epsilon \Omega_i$ the domain of validity which is given by $\Omega_i : [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}})$ for $i = 0, ..., N$. Every polynomial interpolant $I_i(x)$ is associated with a specific subset of nodes and has its own pointwise interpolation error $\varepsilon_i(x)$:

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

$$\pi_i(x) = \prod_{m=0}^{q} (x - x_{s_i+m})$$

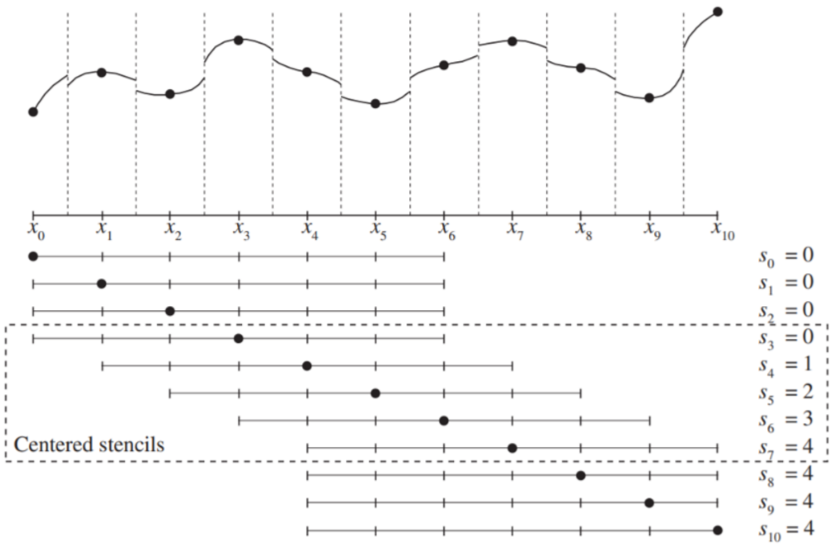
With piecewise polynomials, we avoid the use of high-order polynomials and this prevents the error term from exploding. It is possible to derive classical finite differences on non-uniform grids by differentiation of the piecewise polynomial interpolant [17]:

$$[\frac{du}{dx}]_{x_i} = [\frac{dI_i}{dx}]_{x_i} = \sum_{j=s_i}^{s_i+q} [\frac{dl_{ij}}{dx}]_{x_i} u_j = \sum_{j=s_i}^{s_i+q} a_{ij} u_j$$

For central difference formulas, it is necessary for q to be even,

$$\{s_i\} = \{0, ..., 0, 1, ..., N - q, N - q, ..., N - q\}$$

The subsets $\{0, ..., 0\}$ and $\{N - q, ..., N - q\}$ each have size $\frac{q}{2}$. Therefore the resulting finite-difference will be centred for points x_i, far away from the boundaries of the interval $[x_0, x_N]$. This is apparent in the figure below [15], which displays stencils of $I_i(x)$ for the case $q = 6$ and $N = 10$.



On the other hand let q be odd, then

$$\{s_i\} = \{0, ..., 0, 0, 1, ..., N - q, N - q, ..., N - q\}$$

However now the subsets $\{0, ..., 0\}$ and $\{N - q, ..., N - q\}$ have sizes $\frac{q-1}{2}$ and $\frac{q+1}{2}$ respectively. So the finite-difference expression will be slightly shifted towards the right. [15]

3.1.2 Chebyshev Nodes

Marcinkiewicz's theorem [19] guarantee's that we can pick the interpolation points such that $\varepsilon(x)$ tends to 0 as N increases. Sadly, these may be quite hard to find but Chebyshev nodes are a good alternative to use. It will not be the best interpolation but will certainly be better than equispaced nodes. [22]

Part of the error expression $\varepsilon(x)$ is the polynomial $\pi(x) = (x - x_0)(x - x_1)...(x-x_N)$. Therefore if we can reduce the size of $\pi(x)$ we can decrease the error of the interpolate. So we want to minimize $\max_{-1\leq x\leq 1} |\pi(x)|$ and we'll treat $\frac{u^{(N+1)}(c)}{(N+1)!}$ as a constant. Therefore tackling the Runge phenomenon and creating an accurate interpolation using this approach is particularly useful if you don't know much about the function itself. [21].

For this report we are only interested in Chebyshev polynomials of the first kind. These are defined by:

$$T_n(x) = \cos(n\theta),\ \theta = \arccos(x)$$

However it can also be written as the recurrence relationship:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \end{aligned}$$

due to the trigonometric equation:

$$cos((n+1)\theta) = 2\cos(\theta)\cos(n\theta) - \cos((n-1)\theta)$$

Therefore:

$$\begin{aligned} T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \end{aligned}$$

and so on. So we get $T_n(x)$ is a polynomial with leading coefficient 2^{n-1}.

The roots are:

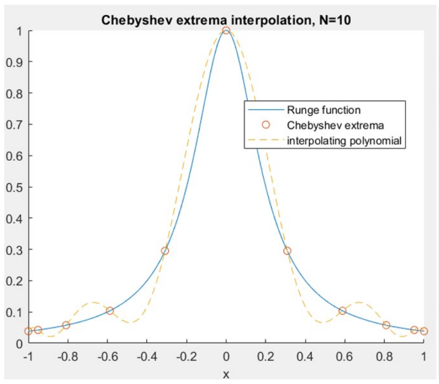
$$\begin{aligned} T_n(x) = \cos(n\theta) = 0, n\theta = (2k-1)\frac{\pi}{2} \quad & k = 1, ..., n \\ \implies a_k = \cos(\frac{(2k-1)\pi}{2n}) \quad & k = 1, ..., n \end{aligned}$$

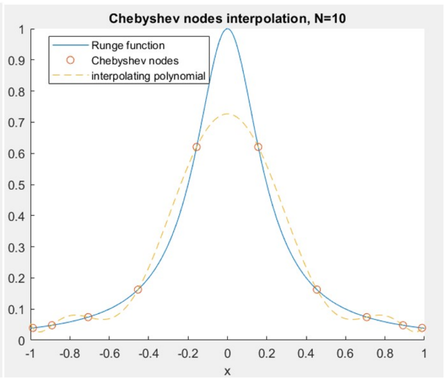
And the extrema are such that $T_n(x) = (-1)^n$ or when $T_(x) = 1$. This happens when $x = 1$ or -1 which implies:

$$b_k = \cos(\frac{(k-1)\pi}{n}) \quad k = 1, ..., n+1$$

Both the roots and the extrema are distributed in $[-1, 1]$ with the same density.D $\frac{n}{\pi\sqrt{1-x^2}}$. meaning there are more nodes on the edges than the centre. [23] This means if the singularities are on the edges of the range both the roots and extrema will reduce the Runge phenomena if used as nodes.

The plots below show both are better alternatives than equispaced nodes and reduce the Runge phenomena for a degree 10 polynomial.





The roots also have the incredible property that the maximum error in determining the Runge phenomena is guaranteed to diminish with increasing polynomial order.

If we pick the nodes to be $x_k = \cos(\frac{(2k-1)\pi}{2n})$ for $k = 1, ...n$ then [22]:

$$
\begin{aligned}
\frac{T_n(x)}{2^{n-1}} &= \prod_{m=0}^{n}(x - x_k) \\
\|\varepsilon_i(x)\| &= \left\| \left(\pi_i(x) \frac{u^{(n+1)}(c)}{(n+1)!} \right) \right\|, \quad \pi_i(x) = \prod_{m=0}^{n}(x - x_k) \\
&= \left\| \left(\frac{T_n(x)}{2^{n-1}} \frac{u^{(n+1)}(c)}{(n+1)!} \right) \right\| \\
&\leq \left\| \left(\frac{1}{2^{n-1}} \frac{u^{(n+1)}(c)}{(n+1)!} \right) \right\|
\end{aligned}
$$

Therefore the error term should decrease as n is increased ie as we use higher order polynomials the error will decrease.

Despite this, they certainly have their flaws. Divergence is still possible if $u^{(N+1)}(c)$ grows too rapidly [19]. It also fails to be the most efficient method of interpolation when the boundaries are not on the edges but in the center as this means the number of nodes required for an acceptable solution becomes very large. We will see cases of both of these happening in the next section.

4 Application: methods to solve an ODE of the form:

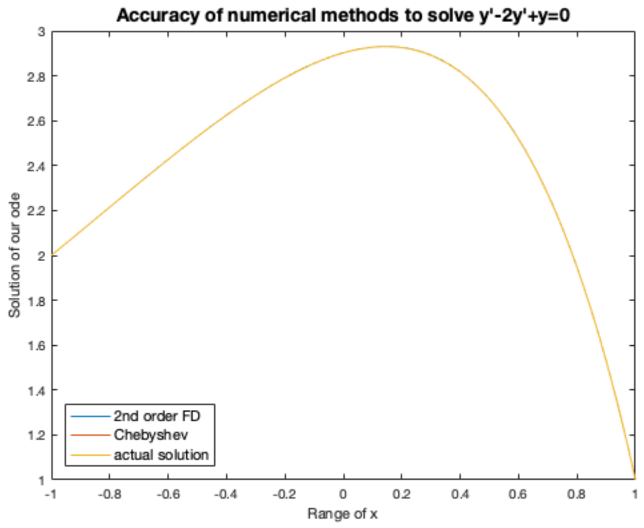
$$\epsilon y'' + a(x)y' + b(x)y = 0, y(x_1) = A, y(x_2) = B, -1 \leq x \leq 1$$

We are studying the simplest family of odes which exhibit isolated layers.

To comfort our code validity, we first checked our 2 different numerical methods against the actual solution of an analytically solvable ode:

$$y'' - 2y' + y = 0$$

As we can observe below, the numerical methods are very accurate as all three curves are on top of each other. We only start to differentiate them at $\sim 10^{-4}$.



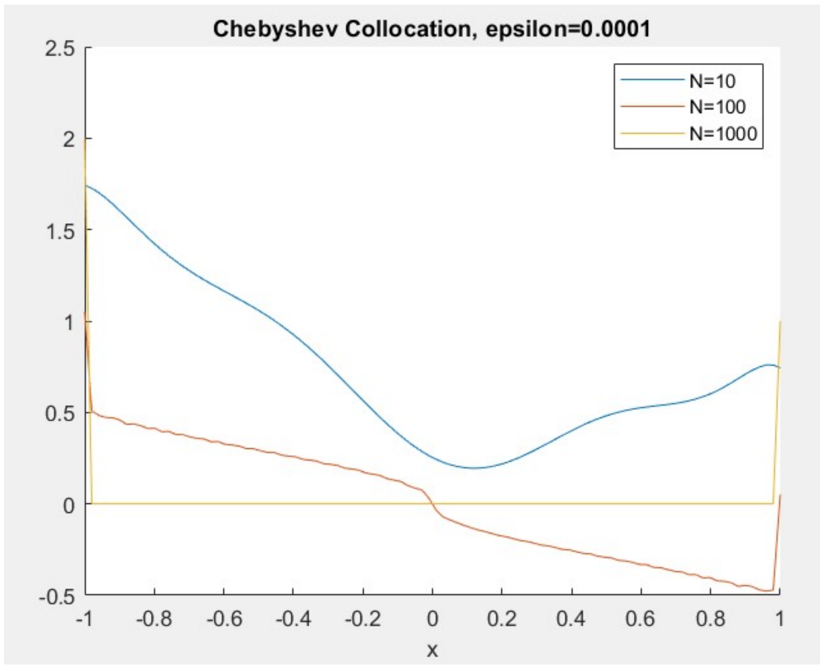
4.1 Collocation method

Within the field of spectral methods, Chebyshev extrema (also known as Gause-Lobatto points) have been increasingly dominant in the last thirty years; they're well surveyed in all the popular spectral books, unlike the Chebyshev nodes. [23] So we will be using the Gause-Lobatto nodes for our collocation method.

The code we wrote to apply the collocation method is in the appendix. Essentially it finds N coefficients for our N Chebyshev polynomials which we can then use to plot our solutions.

$$\epsilon y'' - 2xy' + (1 + x^2)y = 0, y(1) = 1, y(-1) = 2, -1 \leq x \leq 1$$

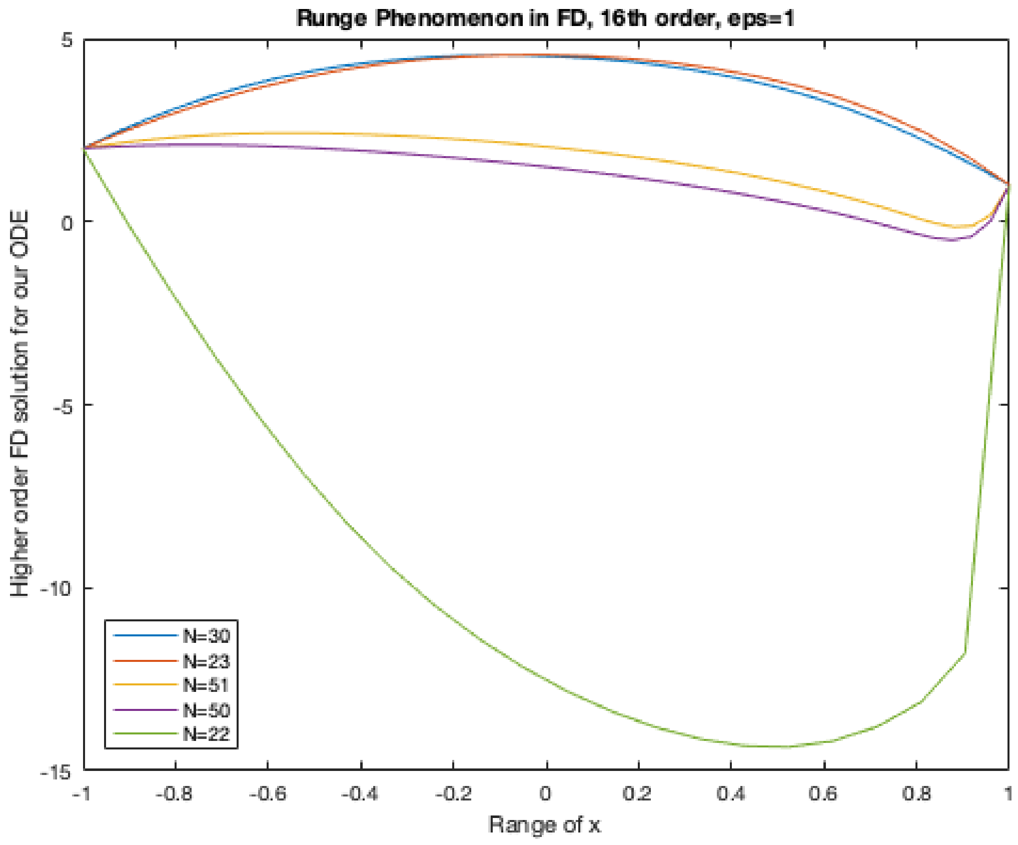
We choose to plot this. It shows the strengths and weaknesses of the Chebyshev method pretty clearly. Below is a plot of using 10th order, 100th order and 1000th order Chebyshev polynomials for epsilon equal to 0.0001.



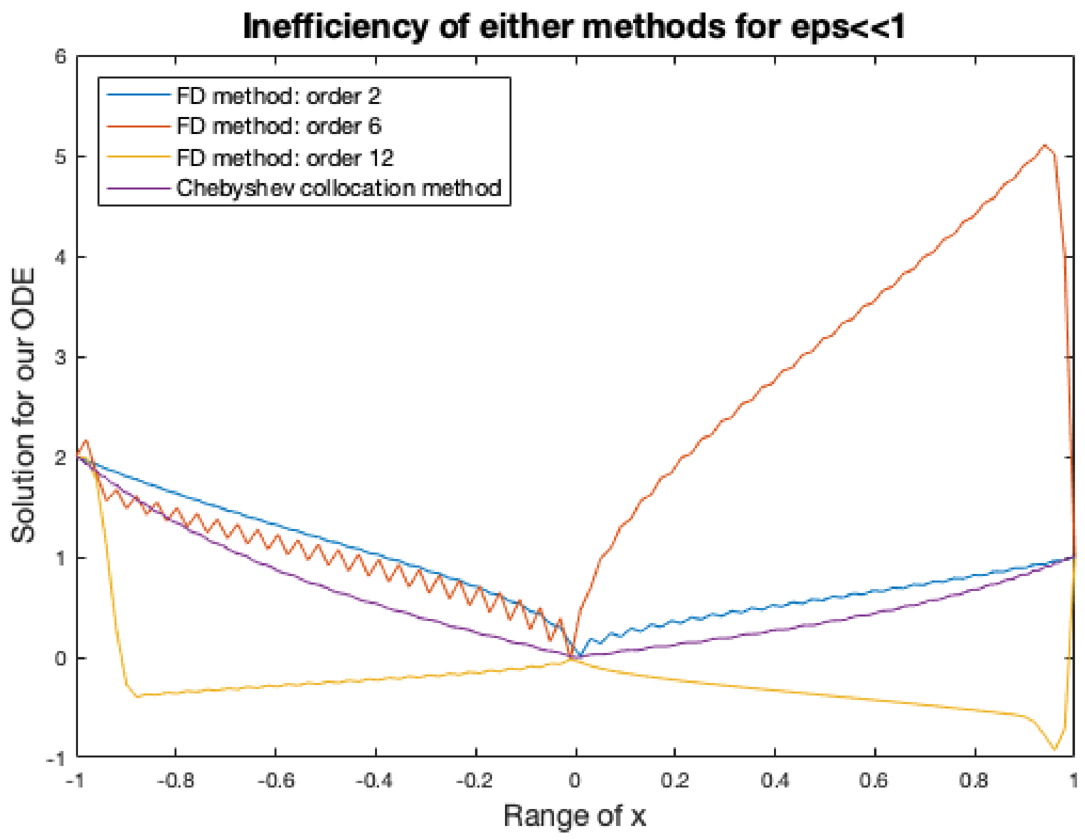
We can see it becomes more accurate as N increases but it takes a 1000 degree polynomial and 1000 nodes to achieve an acceptable solution! This highlights that it beats the Runge phenomenon due to higher order polynomials but since the boundary for this ode is in the centre at x=0 where there are fewer nodes, it takes more nodes than desired.

4.2 Higher order finite differences

Runge phenomenon arises in higher order FD as below:



When $\epsilon << 1$, we observe the following graph, which clearly shows that either numerical methods we've used so far aren't useful. This is why we shall study an analytical method which allows to separate the case with an inner and an outer solution from the boundary layer.



4.3 Perturbation method

4.3.1 Singular Perturbation Problem

The solution to the problem depends on the perturbation parameter ϵ, multiplied by the highest-order derivative. When it is very close to zero, one will use the solution for $\epsilon = 0$ to approximate the asymptotic solution, that satisfies the equation up to a small error. In that case, the problem is no longer second order. The solution qualitatively changes and it is a so-called singular perturbation problem. What is obtained is:

$$a(x)y' + b(x)y = 0$$

Using the separation of variables technique, the solution to this first-order ODE is:

$$y(x) = exp\left[- \int \frac{b(x)}{a(x)} dx \right]$$

As we solve an equation which is of order one, we only find one of the solutions to the original problem. This makes it impossible to satisfy both boundary conditions. The following procedure outlines how the unperturbed problem is different from the perturbed problem.

4.3.2 Method of Matched Asymptotic Expansions

We will apply the Method of Matched Asymptotic Expansions to obtain a solution up to leading-order. The procedure is as follows: the location of the layer (a region where a rapid change in the solution occurs) is obtained; then, an asymptotic approximation of the solution is derived for each of the distinguished regions: the outer region, where the solution is slowly varying, and the inner region, where a rapid variation of the solution occurs. These are then matched to form a uniform solution, valid throughout the whole region.
The procedure relies on the assumption that the inner and the outer solution overlap in an identifiable region, which in principle can occur anywhere in the domain. We will localize it using boundary-layer techniques.

4.3.3 First Example

$$\epsilon y'' - xy' + xy^2 = 0, \quad y(0) = \alpha < 0, \quad y(1) = 0, \quad 0 \leq x \leq 1[10]$$

$a(x) = -x < 0$ for any x in [0,1], thus a boundary layer is at x=1.
No boundary layer is at x=0. If it were, the solution would become exponentially large as $x \to 0$.
We seek an outer solution in terms of a perturbation series in powers of ϵ:

$$y_{out} \sim y_0 + \epsilon y_1 + \epsilon^2 y_2 + O(\epsilon^3), \quad \epsilon \to 0^+$$

The outer problem must satisfy the boundary condition at x=0, $y_{out}(0) = \alpha$.
Substituting the asymptotic expansion into the ODE, we obtain:

$$\epsilon(y_0'' + \epsilon y_1'' + ...) - x(y_0' + \epsilon y_1' + ...) + x(y_0^2 + \epsilon y_0 y_1 + \epsilon^2 y_1^2 + ...) = 0$$

This can be split in terms of powers of ϵ into a sequence of first-order differential equations. The one corresponding to leading-order is:

$$-xy_0' + xy_0^2 = 0$$

This is solved by separation of variables:

$$\frac{dy_0}{dx} = y_0^2 => -y_0^{-1} = x + a_0 => y_0 = \frac{-1}{x+a_0}$$

Applying the boundary condition $y_0(1) = 1$ yields $a_0 = \frac{-1}{\alpha} => y_0(x) = \frac{-\alpha}{\alpha x - 1}$. To establish the inner expansion up to leading-order, we consider the following rescaled problem, aiming to reduce the $\epsilon -$ dependence of the highest derivative in the equation.:

$$X = \frac{x-1}{\delta}, \quad Y(X) \equiv y_{in}(x)$$

The problem then becomes

$$\frac{\epsilon}{\delta^2}\frac{d^2Y}{dX^2} - \frac{\delta X}{\delta}\frac{dY}{dX} + \delta XY^2 = 0$$

The orders of magnitude of the three terms in the equation are $O(\frac{\epsilon}{\delta^2}), O(\frac{1}{\delta}), O(1)$. Balancing the last two terms recovers the outer problem, so it is not the case that $\frac{1}{\delta} \sim 1$. It is also not the case that $\frac{\epsilon}{\delta^2} \sim 1$. The only distinguished limit is thus for $\frac{\epsilon}{\delta^2} \sim \frac{1}{\delta}$, for which $\delta = O(\epsilon)$. Working to leading-order, the equation of interest is:

$$Y_0'' - Y_0' = 0$$

with general solution

$$Y_0(X) = Ae^X + C$$

The boundary condition $Y_0(0) = 0$ leads to the inner solution in its most general form,

$$Y_0(X) = A(e^X - 1)$$

By the matching procedure the value of A is obtained, imposing the condition $lim_{X\to-\infty}Y_0 = lim_{x\to1}y_0$.

$$lim_{X\to-\infty}Y_0 = A = lim_{x\to1}y_0 = \frac{\alpha}{1-\alpha}$$

The asymptotic solution to leading - order is therefore:

$$Y_0(X) = \frac{\alpha}{1-\alpha}(1-e^X)$$

The final uniform solution to leading-order is

$$y(x) = y_0(x) + Y_0(x) - y_c(x)$$

where $y_c(x)$ is the expansion of the solution in the common region:

$$= -\frac{\alpha}{\alpha x - 1} + \frac{\alpha}{1-\alpha}(1-e^X) - \frac{\alpha}{1-\alpha} = -\frac{\alpha}{\alpha x - 1} - \frac{\alpha}{1-\alpha}e^{\frac{(x-1)}{\epsilon}}$$

4.3.4 Our Proposed ODE

$$\epsilon y'' - 2xy' + (1+x^2)y = 0, y(-1) = 2, y(1) = 1, -1 \le x \le 1$$

The coefficient of y' is $a(x) = -2x$. As $a(-1) = 2 > 0$ and $a(1) = -2 < 0$, $a(x)$ changes sign in the region and no boundary layer can occur at $x = \pm 1$. If a boundary layer is located at x=1, then $X = \frac{(x-1)}{\epsilon^2}$ and the inner expansion would be governed by the equation $Y''(X) - 2Y'(X) = 0$. The solution $Y = c_1 e^{2X} + c_2$ becomes exponentially large as $X \to \infty$(this is when $x \to 1$), therefore it is impossible to match the outer solution that remains finite.
Similarly, for $x = -1$, the rescaling is $X = \frac{(x+1)}{\epsilon^2}$ and the inner equation is $Y''(X) + 2Y'(X) = 0$. The solution $Y = k_1 e^{-2X} + k_2$ becomes exponentially large as $X \to -\infty$, hence by the same argument as above, no boundary layer occurs at either $x = \pm 1$. Instead, as $a(x) = 0$ at x=0, we deduce that there might be an interior layer at x=0.
We look for a solution in the form of a perturbation series

$$y_{out} \sim y_0 + \epsilon y_1 + \epsilon^2 y_2 + O(\epsilon^3), \quad \epsilon \to 0^+$$

and substitute this into the equation:

$$\epsilon(y_0'' + \epsilon y_1'' + \epsilon^2 y_2'' + ...) - 2x(y_0' + \epsilon y_1' + \epsilon^2 y_2' + ...) + (1+x^2)(y_0 + \epsilon y_1 + \epsilon^2 y_2 + ...) = 0$$

This can be split in terms of powers of ϵ into a sequence of first-order inhomogeneous differential equations:

$$-2xy_0' + (1+x^2)y_0 = 0$$

$$-2xy_1' + (1+x^2)y_1 = -y_0''$$

$$-2xy_2' + (1+x^2)y_2 = -y_1''$$

where the n^{th} order term is

$$-2xy_n' + (1+x^2)y_n = -y_{n-1}''$$

The leading-order equation

$$-2xy_0' + (1+x^2)y_0 = 0$$

gives solution

$$y_0(x) = a_0^{\pm} \cdot \sqrt{x} \cdot e^{x^2/4}$$

Applying the boundary condition at $x = -1$ would give $2 = a_0^- \cdot i \cdot e^{1/4} \Rightarrow a_0^- = -2 \cdot i \cdot e^{-1/4}$. For $x = -1$ one would get $1 = a_0^+ \cdot e^{1/4} \Rightarrow a_0^+ = e^{-1/4}$. Therefore, let us assume for the moment that the leading-order outer solution is

$$y_0(x) = \begin{cases} -2 \cdot i \cdot e^{-1/4} \cdot \sqrt{x} \cdot e^{x^2/4} & x < 0 \\ e^{-1/4} \cdot \sqrt{x} \cdot e^{x^2/4} & x > 0 \end{cases}$$

To investigate the inner problem, we rescale the problem as $\epsilon \to 0^+$.
Let $X = \frac{x}{\delta}$ and $Y_{in}(X,\epsilon) \equiv y(x,\epsilon)$.
Differentiating by the chain rule, we obtain $\frac{dY}{dX} = \frac{dy}{dx} \cdot \frac{dx}{dX} = \delta y'$ and $\frac{d^2Y}{dX^2} = \frac{d}{dX}\frac{dy}{dX} = \frac{d}{dx}\frac{dy}{dX} \cdot \frac{dx}{dX} = \delta y'' \cdot \delta = \delta^2 y''$.
Rewriting the equation yields

$$\frac{\epsilon}{\delta^2}\frac{d^2Y_{in}}{dX^2} - 2X\frac{dY_{in}}{dX} + (1+\delta^2X^2)Y_{in} = 0$$

In order to determine $\delta(\epsilon)$, one analyzes the three possibilities: $\frac{\epsilon}{\delta^2} \sim 1$, $\frac{\epsilon}{\delta^2} \sim \delta^2, \delta^2 \sim 1$. The only distinguished limit for which $\delta \ll 1$ is $\delta \sim \sqrt{\epsilon}$.
As it is presented in [11], in order to tackle this problem, one should first analyze the inner problem in the neighbourhood of $x = 0$. Substitute $a(x) = -2x$ by -2 and $b(x) = 1 + x^2$ by $b(0) = 1$. The differential equation becomes

$$\frac{d^2Y_{in}}{dX^2} - 2\frac{dY_{in}}{dX} + Y_{in} = 0$$

where $Y_{in} = Y_0+$ higher order terms. Let $Y_0 = e^{-(-2)x^2/4}W = e^{(2x^2)/4}W$. Substituting into the equation and differentiating by the chain rule, we get:

$$\frac{dY_0}{dX} = Xe^{X^2/2}W + e^{X^2/2}\frac{dW}{dX}$$

$$\frac{d^2Y_0}{dX^2} = e^{X^2/2}W + X^2e^{X^2/2}W + Xe^{X^2/2}\frac{dW}{dX} + Xe^{X^2/2}\frac{dW}{dX} + e^{X^2/2}\frac{d^2W}{dX^2}$$
$$\Rightarrow e^{X^2/2}\left(\frac{d^2W}{dX^2} + 2X\frac{dW}{dX} - 2X\frac{dW}{dX} - 2X^2W + X^2W + 2W\right) = 0$$
$$\Rightarrow \left(\frac{d^2W}{dX^2} - X^2W + 2W\right) = 0$$

The parabolic cylinder equation, whose solutions are parabolic functions, is introduced: $\frac{d^2f}{ds^2} + (as^2 + bs + c)f = 0$. Completing the square and rescaling s brings the equation into the form $\frac{d^2f}{ds^2} \mp (1/4s^2 \pm a)f = 0$.
In the ODE satisfied by $W(X)$, let $\sqrt{2}X = S$. What is obtained is

$$\frac{d^2W}{dS^2} + \left(\frac{\beta}{\alpha} - \frac{1}{2} - \frac{1}{4}S^2\right)W = 0 \iff \frac{d^2W}{dS^2} + \left(1 - \frac{1}{4}S^2\right)W = 0.$$

The equation above is a special case, and by the notation established by Whittaker and Watson, its solution is the function $D_v(s)$, as it will be defined in what follows. The function $W(S)$ is a solution to the parabolic cylinder equation above, in the form of a linear combination of parabolic cylinder functions:

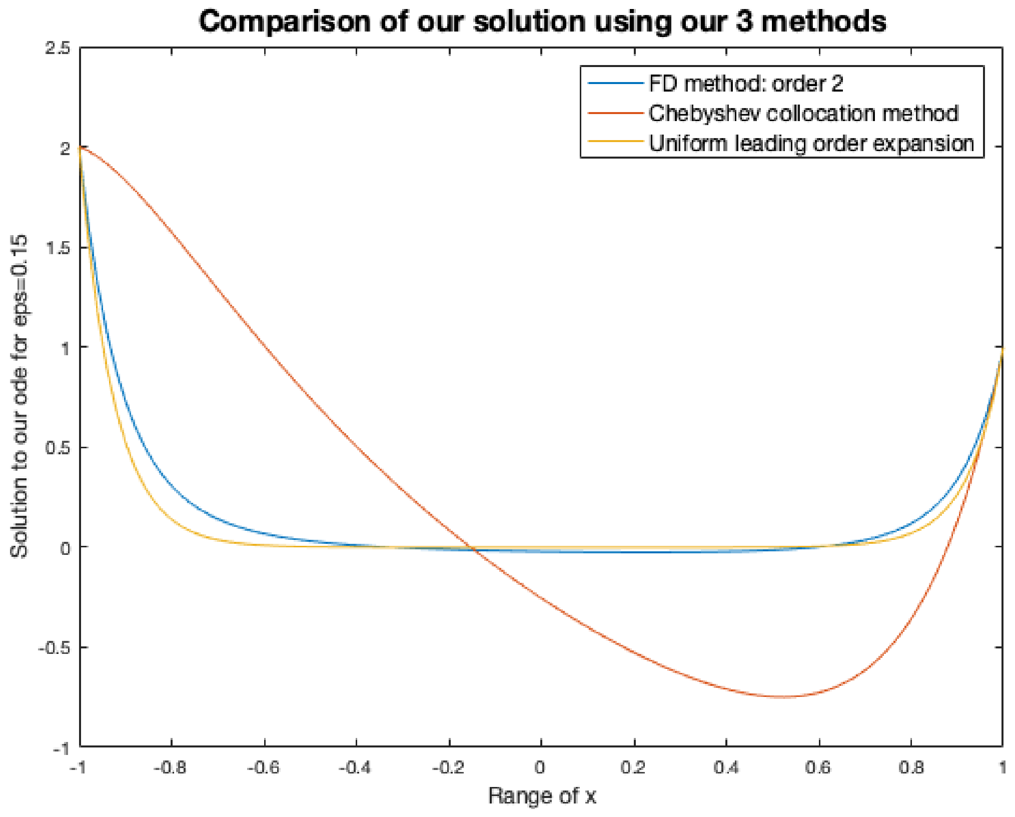
$$W(S) = C_1D_{1/2}(S) + C_2D_{1/2}(-S)$$

where $D_v(t) \sim t^v e^{-t^2/4}$ and $D_v(-t) \sim t^{-v-1}e^{t^2/4}\frac{\sqrt{2\pi}}{\Gamma(-v)}$. The solution is $Y_0(X) = e^{X^2/2}[C_1D_{1/2}(\sqrt{2}X) + C_2D_{1/2}(-\sqrt{2}X)]$. The term $e^{X^2/2}$ grows exponentially for both $X \to \pm\infty$.
For $X \to \infty$, the term $C_2D_{1/2}(-\sqrt{2}X)$ also grows exponentially, thus no asymptotic matching is possible at $x = 1$ unless $C_2 = 0$. Similarly, when $X \to -\infty$, the term $C_1D_{1/2}(\sqrt{2}X)$ grows exponentially, thus no asymptotic matching is possible at $x = -1$ unless $C_1 = 0$. Thus, $Y_0(X) = 0$.
As asymptotic matching imposes the condition that the inner and outer solutions agree at any X, we would require that the integration constants $a_0^{\pm}$ are both zero. As this is not the case, what happens is that the leading-order solution is zero everywhere, except for the points $x = \pm 1$. The leading-order boundary solutions are obtained, similar to the result in [11]:
at $x = -1$: $\quad Y_{0,l}(x) = Ae^{-a(-1)\frac{(x+1)}{\epsilon}}$
and at $x = 1$: $\quad Y_{0,r}(x) = Be^{a(1)\frac{(1-x)}{\epsilon}}$
The uniform asymptotic expansion is thus

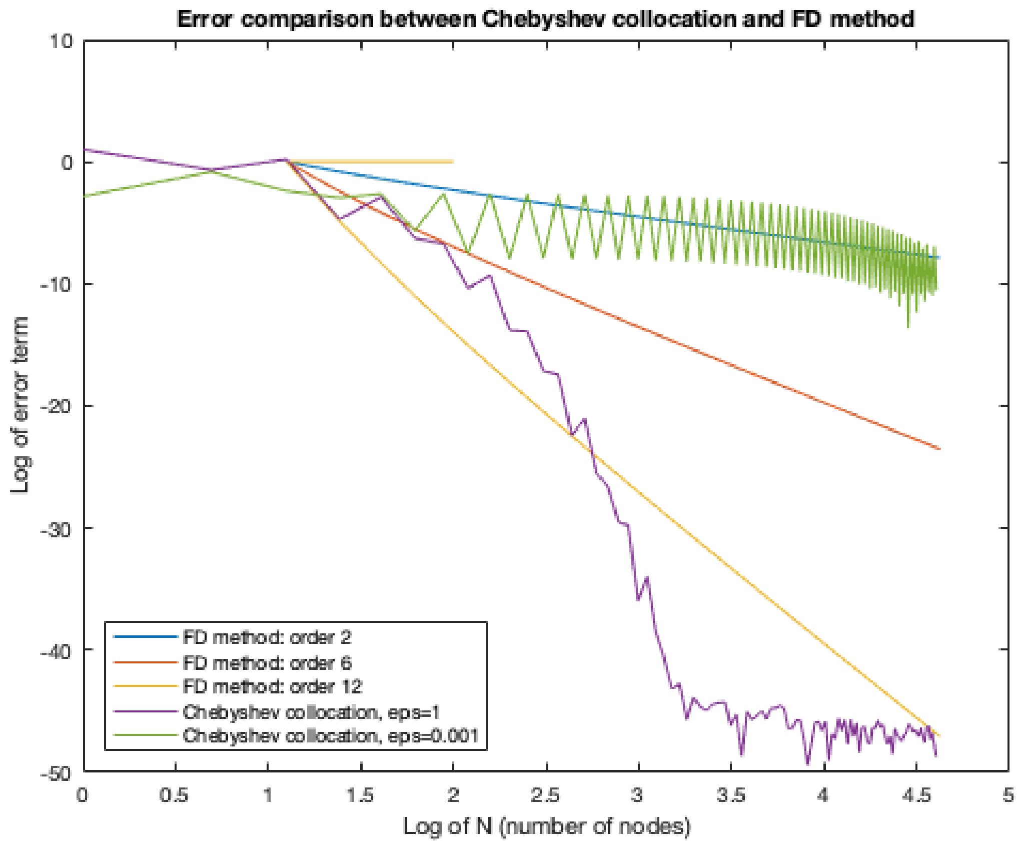
$$y(x) = Y_0(x) + y_0(x) - y_c(x) = Y_{0,l}(x) + Y_{0,r}(x) = 2e^{-2\frac{(x+1)}{\epsilon}} + e^{-2\frac{(1-x)}{\epsilon}}.$$

4.4 Accuracy analysis of these 3 methods

In the following graph, we plot the solution for our ode using our 3 stated methods: the second order finite difference, the Chebyshev collocation method and the uniform leading order expansion approximation.



In the graph below, we did an accuracy analysis of our method for different parameters. As shown previously, when our ϵ is very small, we must consider an inner and an outer solution. However here we see that as we go to higher order our error term decreases. Our Chebyshev collocation method oscillates in terms of accuracy around 12th order FD.



However, one must note that indeed truncation error is reduced when going to higher order (hence decreasing our step size), however, round-off error should be increasing.
Indeed, when you decrease the step size, error decreases as well as expected. However, when the round off error becomes the same magnitude as the truncation error, the global error becomes stable, this happens in decreasing the step size. When you do so very significantly, the round off error may take over and yield a great increase of the global error. This should be taken into account while performing high order finite differences. (Hernandez & Escoto Lopez, 2020)

5 Conclusion & Self evaluation

In conclusion, there is not one best way to solve the Runge phenomenon; depending on the nature of the problem (where the boundaries are, and which function coefficients the derivatives have), the best method to tackle the problem will change. Indeed, we have seen that our problem fails numerically when epsilon is small. The paper [15] discusses a 3rd numerical alternative which is the use of non-uniform grid. This would considerably help reduce the Runge phenomenon. Further analysis could be done in that way.

To gain more insights into real world application and the current methods used in applied mathematics to tackle such problems, we interacted with Hermanns & Hernandez, authors of "Stable high-order finite-difference methods based on non-uniform grid point distributions". We use numerical methods to obtain the leading order solutions to the outer and inner regions and perform asymptotic matching between them in a analytical way. They mention that "these kind of hybrid approaches are extremely useful and significantly extend the applicability of asymptotic methods to real world problems". This would be a great place to extend upon our knowledge and tackle real world problems.

6 References

References

[1] Shellie M Iseri. Department of Mathematics. Oregon State University. *High order finite difference methods*. 1996.

[2] Dmitri Kuzmin. Department of Mathematics. Technical University of Dortmund. *Lecture 4: Finite difference method*. 2011.
Accessed from: `www.mathematik.uni-dortmund.de`

[3] Johannes Westerink. Department of Mathematics. University of Notre Dame. *Lecture 7: Derivation of difference approximation using undetermined coefficients*. 2009.
Accessed from: `coast.nd.edu`

[4] Bengt Fornberg. *Classroom note: Calculation of weights in finite difference formulas*. 1998. Vol 40. Issue 3. Siam Rev. pp.685-691.

[5] Burhan Sadiq, Divakar Viswanath. *Finite difference weights, spectral differentiation, and superconvergence*. 2014. Vol 83. Issue 3289. Mathematics of computation. pp.2403-2427.

[6] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. 2007. Vol 98. Siam.

[7] Andreas Babucke, Markus J Kloker and U Rist. *Accuracy analysis of the fundamental finite-difference methods on non-uniform grids*. 2009. Journal of numerical analysis.

[8] Pascal Frey. Department of Mathematics. Université Pierre et Marie Curie. *The Finite Difference Method*.
Accessed from: `www.ljll.math.upmc.fr`

[9] Andrew Walton. Department of Mathematics. Imperial College London. *Multivariable Calculus (MATH95002)*. 2019.

[10] Ory Schnitzer. *Asymptotic Analysis [Lecture]*. Imperial College London, 20th October 2019.

[11] Steven Orszag, Carl Bender. *Boundary-layer Theory. In: Advanced Mathematical Methods for Scientists and Engineers*. McGraw-Hill Book Company. pp. 419-483. 1999.

[12] Mark Hayden Holmes. *Introduction to Perturbation Methods*. New York. 2013.

[13] John Hunter. *Asymptotic analysis and singular perturbation theory*. Department of Mathematics, University of California at Davis. 1-3. 2004.

[14] Edward John Hinch. *Perturbation Methods*. Cambridge University Press. 1-3, 5-7. 1991.

[15] Miguel Hermanns and Juan Antonio Hernandez. *Stable high-order finite-difference methods based on non-uniform grid point distributions*. 2007. Wiley InterScience.

[16] Juan Hernandez Ramos and Javier Escoto Lopez. Department of Applied Mathematics. School of Aeronautical and Space Engineering Technical University of Madrid (UPM). *How to learn Applied Mathematics through modern FORTRAN.*

[17] Miguel Hermanns. Technical University of Madrid *Stable high order finite difference methods capable of outperforming spectral methods.* 2014

[18] Bao Yan and Tianjian Xing. Department of Mathematics. SEU. *The analysis of Runge phenomenon.* 2011

[19] James F Epperson. (1987). *On the Runge example* The American Mathematical Monthly. Taylor & Francis. Volume 94. Pages 340. Available from: `shorturl.at/cuAO8`

[20] Riviere, B. (2013) *Lecture 18: Minimax Approximation, Optimal Interpolation, Chebyshev Polynomials.* Available from: `http://home.iitk.ac.in/ pranab/ESO208/rajesh/03-04/Chebyshev.pdf` [Accessed 15/06/2020].

[21] Ron, A. (2010) *Lecture 10: Introduction to Splines.* Available from: `http://pages.cs.wisc.edu/ amos/412/lecture-notes/lecture10.pdf` [Accessed 15/06/2020]

[22] Gil, A., Seugra, J. & Temme Nico. (2007) *Chebyshev Expansions.* pp. 62-63. Available from: `https://archive.siam.org/books/ot99/OT99SampleChapter.pdf`

[23] Xu, K. (2016) *The Chebyshev points of the first kind.* Available from: `http://www.sciencedirect.com/science/article/pii/S0168927416000039.`

7 Appendix

7.1 Fornberg Algorithm

```
function cwei = FDweights(z,x,m)

%--------------------------------
% finite-difference weights
% (Fornberg algorithm)
```

```
%
% z:  expansion point
% x:  vector of evaluation points
% m:  order of derivative
%
% Example: cwei = FDweights(0,[0 1 2],1);
% gives    cwei = [-3/2  2  -1/2]
%
% h f'_0 = -3/2 f_0 + 2 f_1 - 1/2 f_2
%
%---------------------------------

  n  = length(x)-1;
  c1 = 1;
  c4 = x(1)-z;
  c = zeros(n+1,m+1);
  c(1,1) = 1;
  for i=1:n
    mn = min(i,m);
    c2 = 1;
    c5 = c4;
    c4 = x(i+1)-z;
    for j=0:i-1
      c3 = x(i+1)-x(j+1);
      c2 = c2*c3;
      if (j == (i-1))
        for k=mn:-1:1
          c(i+1,k+1) = c1*(k*c(i,k)-c5*c(i,k+1))/c2;
        end
        c(i+1,1) = -c1*c5*c(i,1)/c2;
      end
      for k=mn:-1:1
        c(j+1,k+1) = (c4*c(j+1,k+1)-k*c(j+1,k))/c3;
      end
      c(j+1,1) = c4*c(j+1,1)/c3;
    end
    c1 = c2;
  end
  cwei = c(:,end);
%Application around 0, N=6, 2nd derivative
%FDweights(0,[-1,-3/5,-1/5,1/5,3/5,1],2)
%ans=[-0.65;5.07;-4.42;-4.42;5.07;-0.65]
```

```
function xvec = iorder(norder)
% for a given number norder, return the list of numbers from
  0 to norder-1
```

```
 for i=1:norder+1
  xvec(i)=i-1;
  end
%Application for norder=6
%iorder(6)
%ans=[0 1 2 3 4 5 6]
```

```
function cfv = fornberg_algo(ifdorder, ider)
% iforder is the accuracy of finite difference.
% ider is the derivative order, so ider= 1-st derivative,
    ider =2-nd derivative etc.
% inode is at which node the derivative is required.
% eg: for three point FD scheme, i.e 2nd-order accuracy:
    three point stencil, nodes will be  0, 1 & 2.
gap=' : ';
gapsemi=' ; ';
node=' node';
derivative=' deriv order : ';
acc=' accuracy : ';
fprintf('%s %d %s %s %d  \n',derivative, ider,gapsemi, acc,
    ifdorder );

xvec=iorder(ifdorder);
for inode=0:ifdorder
fprintf('%s %d  %s \n',node,inode, gap);
cfv=FDweights(inode,xvec,ider);
disp(cfv)
end
%Application for ifdorder=2, ider=2
%fornberg_algo(2,2)
%deriv order:  2 ; accuracy:  2 ; node 0: [1;-2;1] ; node 1:
    [1;-2;1] ;
%node 2: [1;-2;1]
```

7.2 ODE solved using collocation method

```
function a = solveode(N,e)
%Initialize D. The first row is 1 so I'm using the ones
    matrix to make the
%code a bit more efficient
D = ones(N+1,N+1);
%Replacing bottom row with the second boundary condition
for i=1:N+1
    D(N+1,i)=T(0,(i-1),-1);
end
```

```
%Replacing the rest of the rows with the ODE evaluated at the
     Gauss-Lobbato
%points
for i = 2:N
    for j = 1:N+1
        x = cos(((i-1)*pi)/N);
        D(i,j)=e*T(2,(j-1),x)- 2*x*T(1,(j-1),x)+(1+x^2)*T(0,(
    j-1),x);
    end
end
%Making the Matrix b
b=zeros(N+1,1);
b(0+1)=1;
b(N+1)=2;
%Doing D^-1 * b to find a
a=D\b;
%Displays D and b to make sure they are correct
disp(D);
disp(b);
end
```

```
function y = plotR(N,e)
%x ranges from -1 to 1. I have 1000 points to improve the
    resolution
x=linspace(-1,1,100);
%Solves 'a'
a = solveode(N,e);
%Initialise y
y=zeros(100,1);
%Do the sum to find y at each value of x
for i = 1:100
    for j=1:N+1
        y(i) = y(i)+a(j)*T(0,j-1,x(i));
    end
end
%Plots x and y
plot(x,y);
hold on
end
```

7.3 ODE solved using FDM

```
function y = ode2(ifdorder, eps, N)
% this function solves eps * y'' - 2x * y'+ (1+x^2) * y = 0,
    -1<=x<=1
% ifdorder is the accuracy of finite difference

% STEP 1 : Initialization of values
  % STEP 1.1: Initialize display of parameters
    xvec=iorder(ifdorder); % xvec is the vector of evaluation points
    ider=2; % ider is the derivative order
    ider1=1;
    gap=' : ';
    gapsemi=' ; ';
    node=' node';
    derivative=' deriv order : ';
    acc=' accuracy : ';
    fprintf('%s %d %s %s %d  \n',derivative, ider,gapsemi, acc, ifdorder);
   % STEP 1.2: Fornberg Algorithm to compute coefficients at each node
    for inode=0:ifdorder % inode is at which node the derivative is required
      fprintf('%s %d  %s \n',node,inode, gap);
      cfv=FDweights(inode,xvec,ider);
    disp(cfv)
    end

% STEP 2: Create coefficient matrix M
  % STEP 2.1: Initialize values
    h = 2/(N-1); % length between each node
    x=linspace(-1,1,N); % vector of nodes

  % STEP 2.2: Loop to update each row of M
    for np=1:N
    % STEP 2.2.1: Fill in first row
      if np == 1 % coeff of y_1; left boundary condition
        M(np,np)=1;
    % STEP 2.2.2: Fill in last row
      elseif np == N % coeff of y_N; right boundary condition
        M(np,np)=1;
    % STEP 2.2.3: Fill in rows where central stencil apply
      elseif np >= 1+ ifdorder/2 && np <= N - ifdorder/2
        inode=ifdorder/2; % define the node at which we should evaluate the coefficients
        cfv0=FDweights(inode,xvec,0); % set
```

```
coefficients for 0th derivative
        cfv=FDweights(inode,xvec,ider1); % set coefficients for 1st derivative
        cfv2d=FDweights(inode,xvec,ider); % set coefficients for 2nd derivative
          for k = 0: length(cfv2d)-1 % loop to shift entry in row corresponding to elseif statement, size of stencil
            M(np,np+k-ifdorder/2)=eps*cfv2d(k+1) - 2*x(np)*cfv(k+1)*h+ h*h*(1+x(np)*x(np))*cfv0(k+1);
          end
    % STEP 2.2.4: Fill in rows where forward stencil apply
      elseif np < 1 + ifdorder/2 %left sided difference
        inode = np - ifdorder/2 + 1;
        cfv0=FDweights(inode,xvec,0);
        cfv=FDweights(inode,xvec,ider1);
        cfv2d=FDweights(inode,xvec,ider);
          for k = 0: length(cfv2d)-1
            M(np,k+1)=eps*cfv2d(k+1) - 2*x(np)*cfv(k+1)*h+ h*h*(1+x(np)*x(np))*cfv0(k+1);
          end
    % STEP 2.2.5: Fill in rows where backward stencil apply
      else %right sided difference
        inode = np - ifdorder/2 +1;
        cfv0=FDweights(inode,xvec,0);
        cfv=FDweights(inode,xvec,ider1);
        cfv2d=FDweights(inode,xvec,ider);
          for k = 0: length(cfv2d)-1
            M(np, k+1 +N - length(cfv2d))=eps*cfv2d(k+1) - 2*x(np)*cfv(k+1)*h+ h*h*(1+x(np)*x(np))*cfv0(k+1);
          end
    end
    end
  % STEP 2.3: Print M
    disp(M)

% STEP 3: Solve My=b
  % STEP 3.1: Set the value of b with boundary conditions
    b=zeros(N,1);
    b(1)=2;
    b(N)=1;
  % STEP 3.2: Use Gauss elimination to solve My=b for y
```

```
        y = M\b;

% STEP 4: Plot numerical solution of ODE
    % STEP 4.1: Define x values
        x=linspace(-1,1,N);
    % STEP 4.2: Plot solution and label axes
        plot(x,y);
        xlabel('x');ylabel('y');
```

7.4 Accuracy analysis

```
function AbsOde(N,ifdorder)
x=2:N+2;
for i=3:N+2
    h = 2/(i-1);
    x(i-1)=log(x(i-1));
    y(i-1)=log(h^(ifdorder));
end
plot(x,y)
hold on
xlabel('N');ylabel('log(h^ifdorder)');
end
```

```
function LogAbsA(N,eps)
%Solves a for N=500
a=solveode(N,eps);
%Initilise x and y
x=1:100;
y=zeros(100,1);
%We arelogging the a_n's as this gives us abetter idea on
   when A_n gets
%arbitrarily small
for i=1:100
    x(i)=log(x(i));
    y(i)=log(abs(a(i)));
end
%Plots the graph
plot(x,y)
hold on
xlabel('Log of N (number of nodes)');ylabel('Log of error
   term');
end
```