Numerical Analysis B

Byron Jacobs
SCHOOL OF MATHEMATICS AND APPLIED MATHEMATICS
UNIVERSITY OF JOHANNESBURG

Contents

Prerequisites

This is the set of course notes for APM8X13, authored, compiled and prepared by Professor Byron Jacobs. The notes are intended to complement the lectures and other course material, and are by no means intended to be complete. Students should consult the various references that have been given to find additional material, and different views on the subject matter.

This material is under development. Please would you report any errors or problems (no matter how big or small). Any suggestions would be gratefully appreciated.

A special thanks to Dr. Matthew Woolway for the enormous effort put into authoring, compiling and preparing these notes.

School of Mathematics and Applied Mathematics, University of the Johanesburg, Johannesburg, South Africa

1

Introduction

Differential equation based mathematical modelling often aims to capture/describe complex phenomenological behaviours. The equations that arise from such a task may be nonlinear in nature or difficult/impossible to solve using analytic methods. As such, a reliable framework for the construction of a numerical scheme for such an equation is paramount. This course explores an approach to constructing, analysing and simulating numerical methods for ordinary and partial differential equations (ODEs and PDEs).

1.1 Course Outline

This course will cover

- Taylor Series (Revisted)
- Numerical Solution Methods and Analysis for Ordinary Differential Equations (ODEs)
- Numerical Solution Methods and Analysis for Partial Differential Equations (PDEs)

1.2 Learning outcomes

On completion of this course, a student should be able to:

- Expand a function around a give point using the Taylor series
- Construct and analyse a variety numerical methods for ODEs
- Numerically simulate/solve a system of ODEs
- Construct and analyse a variety numerical methods for parabolic PDEs
- Numerically simulate/solve parabolic PDEs
- Interpret a numerical method given in plain mathematics and implement that method in code

1.3 Module Descriptor

- NQF-level: 8
- MQF-credits: 16
- Module type: Semester module

1.4 Lecture/Consultation Schedule

Lectures: Tuesday 9:40 - 11:15
Consultation: Via Email or by appointment

1.5 Lecturer Contact Details

Prof Byron Jacobs
Mathematics and Applied Mathematics
Email: `byronj@uj.ac.za`
Office: B Ring 529, Auckland Park Campus

1.6 Additional Resources

There are many textbooks on the subject of numerical analysis amenable to this course. Many of these textbooks have far greater scope than this course and as such select chapters from the textbooks below should be consulted for enrichment.

- Numerical Analysis - *Burden and Faires*
- Numerical Analysis - *T. Sauer*
- Scientific Computin - *M. Heath*

1.6.1 Python/Google Colab

All aspects of coding and implementation for this course will be done in Python. Specifically, using the Jupyter Notebook framework. Google Colab (`https://colab.research.google.com/`) is a cloud hosted framework that allows instant access to a Jupyter Notebook. You are welcome to install a lo-

cal instance of Jupyter Notebook, instructions for how to do so can be found online.

1.7 Assessments

In all assessments, both formative and summative, questions are posed that require the application of acquired principles to solve well-defined problems in mathematics.

- Self-assessment: The student is expected to perform self-assessment by completing the activities (exercises in text book) at the end of each learning unit.
- Continuous assessment: During and/or after each learning unit the student will be assessed (class tests and/or assignments) and these assessments may form part of the semester mark.
- Formative assessment: Two major tests and a series of tutorial tests will be written during the semester (see Assessment Schedule). These tests will form part of the semester mark.
- Summative assessment: An examination paper at the end of the semester testing all the content covered in class. The exam will 50% towards the final mark.

The semester mark together with the examination mark will constitute the final pass mark. The student will be pronounced competent/ not yet component, based on the final pass mark obtained.

1.7.1 Assessment schedule

The usual examination rules and regulations are applicable and all assessments are treated as exams (see subsections below). Requests for reconsideration of an assessment mark will only be granted at the time when an assessment is handed out and discussed.

- Test 1: Tuesday, 31 August 2021
- Test 2: Tuesday, 19 October 2021
- Aegrotats (sick tests): Sick tests will be scheduled within 7 to 10 days of the corresponding semester test.
- Exam: TBC

1.7.2 Pass Requirements and Compilation of Marks

The following assessment guidelines will be implemented for this module:

1. Weighting of assessments:

- Semester mark: Test 1 (40%) + Test 2 (40%) + Assignments (20%)= 100%
- Examination mark: Main Exam = 100%
- Final mark: 50% Semester + Exam 50% = 100%

2. Exam entrance requirements:

- A student must obtain a semester mark of at least 40% to gain entrance to the exam.
- A student must obtain at least 40% in his/her final summative assessment (exam or supplementary exam) to pass the module.
- A student must obtain a final mark of 40% to 49% to qualify for the supplementary exam.

3. Pass requirements:

- All assessments are compulsory.
- The pass mark for any assessment (test, tutorial, assignment, project, etc.) is 50%. A distinction is a mark 75%.
- When a traditional examination or summative assessment is used as a last assessment, the student passes a module subject to obtaining a final calculated mark of at least 50%.
- The final mark for a supplementary summative assessment is capped at 50%.

1.8 Rules and Procedure for Absence from an Assessment

A student may be granted a special assessment opportunity if he/she applies within 7 working days subsequent to the original date of assessment missed. A student must submit a completed online application form for the aegrotat test together with a written & valid reason(s). The following cases will be considered and a supplementary assessment may be granted if the documentation as specified is delivered to your lecturer as stated above.

Reason for absence	Documentation as proof
Illness	The application form completed by student and medical
Compassionate reasons (Immediate family only)	Relevant documentation such as death certificate etc. and
Legitimate reasons	An affidavit signed and stamped by a commissioner of oa

- The discretion of the lecturer will play a major role in the making of the final decision on the supplementary assessment opportunity.
- Students must apply to write the supplementary exam or special supplemen-

tary exam only from CAA at the following URL: https://www.uj.ac.za/studyatUJ/Pages/Sick-notesubmission-form.aspx

2

Taylor Series (Revisted)

Taylor's theorem and the Taylor series of a function provide an analysis framework for a broad class of numerical methods. Primarily "finite difference" methods.

Theorem 2.1. Taylor's thoerem for functions of one[1] real variable *Assume that $f(x)$ has $n+1$ continuous derivatives for $a \leq x \leq b$, and let $x_0 \in [a,b]$ then*

$$f(x) = p_n(x) + R_n(x), \quad a \leq x \leq b,$$

where

$$p_n(x) = f(x_0) + (x-x_0)f'(x_0) + \frac{(x-x_0)^2}{2!}f''(x_0) + ... + \frac{(x-x_0)^n}{n!}f^{(n)}(x_0),$$

is the Taylor polynomial of degree n for the function $f(x)$ and the point of approximation x_0, and $R_n(x)$ is the remainder in approximating $f(x)$ by $p_n(x)$. We have

$$R_n(x) = \frac{1}{n!}\int_{x_0}^{x}(x-\xi)^n f^{(n+1)}(\xi)d\xi \tag{2.1}$$

$$= \frac{(x-x_0)^{n+1}}{(n+1)!}f^{(n+1)}(c_x), \tag{2.2}$$

where c_x is an unknown point between x_0 and x.

This result may be interpreted in a number of ways. The first illustrates that, if a function (and all necessary derivatives) are known at a **point** then the function can be reconstructed anywhere in the domain. Consider,

$$f(x) = f(x_0) + (x-x_0)f'(x_0) + \frac{(x-x_0)^2}{2!}f''(x_0) + \mathcal{O}(x-x_0)^3. \tag{2.3}$$

if $f(x_0), f'(x_0), ...$ are given $f(x)$ can be found for any value of x. Obviously, in practice, we can't know arbitrarily many function (and derivative) values.

[1] Taylor's theorem can be extended for function in multiple variables, this is left as an exercise to the reader.

As such, we will incur an error in our approximation which is proportional to the distance between x and x_0, shown explicitly by the $\mathcal{O}$ term.

The second interpretation of Taylor's series is allowing one to define the relationship between a function's derivatives and discrete sampling of the function. This will form the basis of constructing finite difference methods, by allowing us to **discretise** differential operators. With appropriate substitution we have

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + \frac{\Delta x^2}{2!} f''(x) + ... + \mathcal{O}(\Delta x^n), \tag{2.4}$$

rearranging we find,

$$f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} + \mathcal{O}(\Delta x). \tag{2.5}$$

Example 2.1. Consider $f(x) = e^x$, then $f^{(n)}(x) = e^x \quad \forall n$. Find the series approximation to $f(0.25)$ using four terms of Taylor's series, recalling that

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}.$$

Solution: Take $x = 0$ and $\Delta x = 0.25$, then from Taylor's series we have

$$f(x+\Delta x) \approx f(x) + \Delta x f'(x) + \frac{\Delta x^2}{2!} f''(x) + \frac{\Delta x^3}{3!} f'''(x) \tag{2.6}$$

$$f(0.25) \approx f(0) + 0.25 f'(0) + \frac{0.25^2}{2!} f''(0) + \frac{0.25^3}{3!} f'''(0) \tag{2.7}$$

$$= 1.28385 \tag{2.8}$$

Where the exact value is $e^{0.25} = 1.28403$.

```
import numpy as np
def taylorExp(n,dx):
  ns = np.arange(n)
  num = (dx)**(ns)
  den = np.array([ np.math.factorial(n) for n in ns])
  return np.sum(num/den)

n = 4
dx = 0.25
res = taylorExp(n,dx)
print(f"for n = {n} and dx = {dx} we get, {res}.")
```

```
## for n = 4 and dx = 0.25 we get, 1.2838541666666667.
```

Exercise 2.1. Verify the order accuracy of the approximation to the first derivative in the notes above.

Exercise 2.2. Using the Taylor expansion of $f(x + \Delta x)$ and $f(x - \Delta x)$ construct an approximation of $f'(x)$ that is $\mathcal{O}(\Delta x^2)$.

Exercise 2.3. Write a python function called `taylorSin(n,dx)` which computes the value of $\sin(dx)$ using n terms of Taylor's series.

(Hint: make use of the cyclic nature of the derivatives of sin.)

```
## for n = 5 and dx = 0.4 we get, 0.3894183423097002.
```

Exercise 2.4. Using the Taylor expansion of $f(x + \Delta x)$ and $f(x - \Delta x)$ construct an approximation of $f''(x)$ that is $\mathcal{O}(\Delta x^2)$.

2.1 Higher order and more accurate approximations

We now consider how one might use the Taylor series to construct an approximation to the second derivative of a function, $f''(x)$. Making $f''(x)$ the subject of the formula in (2.3) we find that

$$f''(x) = \frac{2}{\Delta x^2}\left(f(x + \Delta x) - f(x) - \Delta x f'(x)\right) + \mathcal{O}(\Delta x), \tag{2.9}$$

which raises the issue of the approximation to the second derivative being dependent on the first. To overcome this issue consider the expansions

$$f(x + \Delta x) = f(x) + \Delta x f'(x) + \frac{\Delta x^2}{2!} f''(x) + ... + \mathcal{O}(\Delta x^n), \tag{2.10}$$

$$f(x - \Delta x) = f(x) - \Delta x f'(x) + \frac{\Delta x^2}{2!} f''(x) - ... + \mathcal{O}(\Delta x^n). \tag{2.11}$$

Adding the above two sequences, we can see that the first derivative terms cancel.

$$f(x + \Delta x) + f(x - \Delta x) = 2f(x) + 2\frac{\Delta x^2}{2!} f''(x) + \mathcal{O}(\Delta x^4), \tag{2.12}$$

paying **careful attention** to the order of leading error term. Now making $f''(x)$ the subject of the formula, we arrive at the following approximation,

$$f''(x) = \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2). \tag{2.13}$$

This same trick can be applied to derive approximation to **both** higher order derivatives ($f^{(n)}(x)$) as well as approximations to derivatives with increased **accuracy**.

Exercise 2.5. Using the Taylor expansion of $f(x+2\Delta x)$, $f(x+\Delta x)$, $f(x-\Delta x)$ and $f(x - 2\Delta x)$ construct an approximation of $f'(x)$ that is $\mathcal{O}(\Delta x^4)$.

Exercise 2.6. Using the Taylor expansion of $f(x + 2\Delta x)$, $f(x + \Delta x)$, \$f(x), $f(x-\Delta x)$ and $f(x-2\Delta x)$ construct an approximation of $f''(x)$ that is $\mathcal{O}(\Delta x^4)$.

Exercise 2.7. Using the Taylor expansion of $f(x)$, $f(x + \Delta x)$, $f(x + 2\Delta x)$ and $f(x + 3\Delta x)$ what is the best order accuracy one can achieve for the approximation of $f''(x)$?

Notice that asymmetric stencils are unable to achieve the same order accuracy as their symmetric counterparts. Fornberg (Fornberg, 1998) presents a formula for deriving the coefficients for an arbitrary order derivative to arbitrary accuracy for a given stencil. The paper can be found here.

3

Ordinary Differentiable Equations (ODEs)

Ordinary differential equations govern a great number of many important physical processes and phenomena. Not all differential equations can be solved using analytic techniques. Consequently, numerical solutions have become an alternative method of solution, and these have become a very large area of study.

Importantly, we note the following:

- By itself $y' = f(t, y)$ does not determine a unique solution.
- This simply tells us the slope $y'(t)$ of the solution function at each point, but not the actual value $y(t)$ at any point.
- There are an infinite family of functions satisfying an ODE.
- To single out a particular solution, a value y_0 of the solution function must be specified at some point t_0. These are called initial value problems.

3.1 Initial Value Problems

The general first order equation can be written as:

$$\frac{dy}{dt} = f(t, y), \tag{3.1}$$

with $f(t, y)$ given. Together with this may be given an initial condition, say $y(t_0) = y_0$, in which case (3.1) and this condition form an initial value problem. Its general solution contains a single arbitrary constant of integration which can be determined from the given initial condition.

3.1.1 Stability of ODEs

Should members of the solution family of an ODE move away from each other over time, then the equation is said to be **unstable**. If the family members move closer to one another with time then the equation is said to be **stable**. Finally, if the solution curves do not approach or diverge from one another with time, then the equation is said to be **neutrally stable**. So small perturbations to a solution of a stable equation will be damped out with time since the

solution curves are converging. Conversely, an unstable equation would see the perturbation grow with time as the solution curves diverge.

To give physical meaning to the above, consider a 3D cone. If the cone is stood on its circular base, then applying a perturbation to the cone will see it return to its original position standing up, implying a stable position. If the cone was balanced on its tip, then a small perturbation would see the cone fall, there the position is unstable. Finally, consider the cone resting on its side, applying a perturbation will simply roll the cone to some new position and thus the position is neutrally stable.

3.1.2 Unstable ODE

An example of an unstable ODE is $y' = y$. Its family of solutions are given by the curves $y(t) = ce^t$. From the exponential growth of the solutions we can see that the solution curves move away from one another as time increases implying that the equations is unstable. We can see this is the plot below.

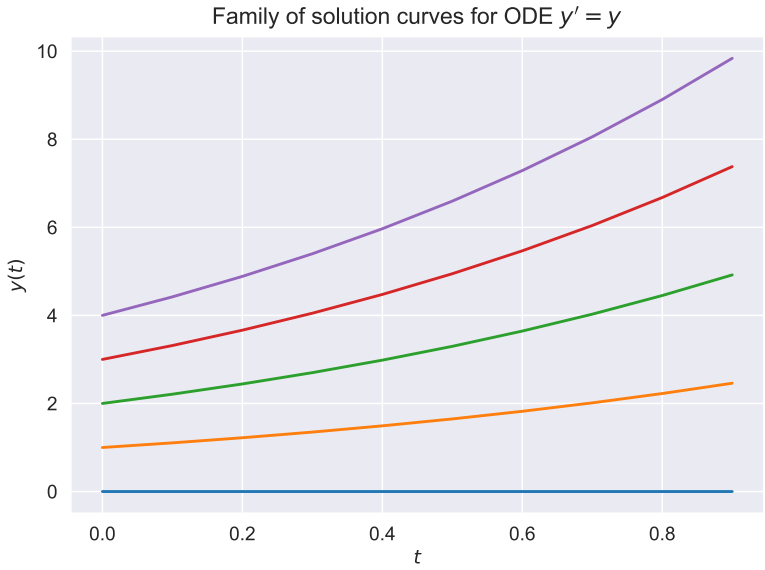
```
## <Figure size 1300x900 with 0 Axes>
```

```
## [<matplotlib.lines.Line2D object at 0x0000000033130F70>]
## [<matplotlib.lines.Line2D object at 0x0000000033144310>]
## [<matplotlib.lines.Line2D object at 0x0000000033144670>]
## [<matplotlib.lines.Line2D object at 0x00000000331449D0>]
## [<matplotlib.lines.Line2D object at 0x0000000033144D30>]
```

```
## Text(0.5, 0, '$t$')
```

```
## Text(0, 0.5, '$y(t)$')
```

```
## Text(0.5, 1.0, 'Family of solution curves for ODE $y^\\prime = y$')
```



3.1.3 Stable ODE

Now consider the equation $y' = -y$. Here the family of solutions is given by $y(t) = ce^{-t}$. Since we have exponential decay of the solutions we can see that the equation is stable as seen in Figure below.

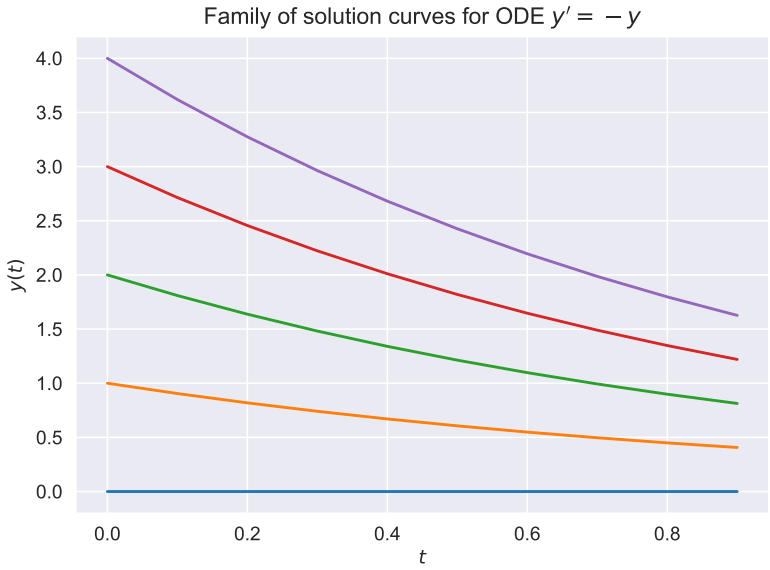
```
## <Figure size 1300x900 with 0 Axes>
```

```
## [<matplotlib.lines.Line2D object at 0x0000000033780430>]
## [<matplotlib.lines.Line2D object at 0x0000000033780790>]
## [<matplotlib.lines.Line2D object at 0x0000000033780AF0>]
## [<matplotlib.lines.Line2D object at 0x0000000033780E50>]
## [<matplotlib.lines.Line2D object at 0x000000003378B1F0>]
```

```
## Text(0.5, 0, '$t$')
```

```
## Text(0, 0.5, '$y(t)$')
```

```
## Text(0.5, 1.0, 'Family of solution curves for ODE $y^\\prime = -y$')
```



3.1.4 Neutrally Stable ODE

Finally, consider the ODE $y' = a$ for a given constant a. Here the family of solutions is given by $y(t) = at + c$, where c again is any real constant. Thus, in the example plotted below where $a = \frac{1}{2}$ the solutions are parallel straight lines which neither converge or diverge. Therefore, the equation is neutrally stable.

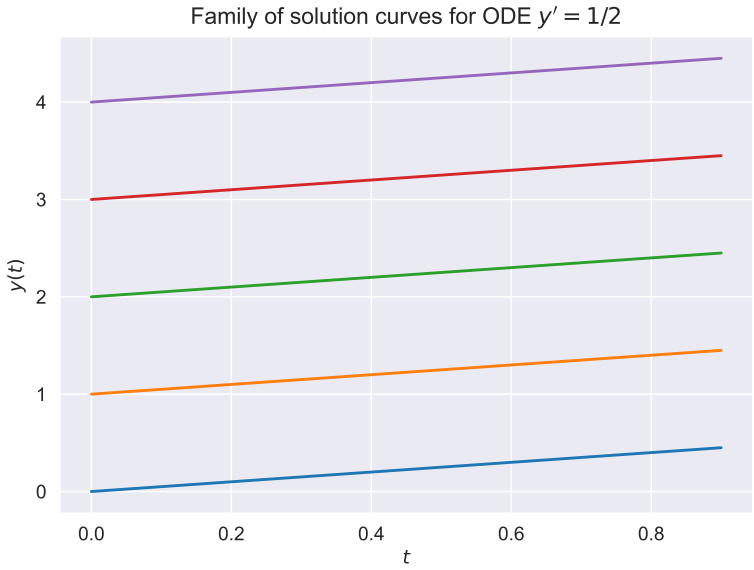
```
## <Figure size 1300x900 with 0 Axes>
```

```
## [<matplotlib.lines.Line2D object at 0x0000000033780A30>]
## [<matplotlib.lines.Line2D object at 0x0000000033C5F490>]
## [<matplotlib.lines.Line2D object at 0x0000000033C5F8B0>]
## [<matplotlib.lines.Line2D object at 0x0000000033C5FDC0>]
## [<matplotlib.lines.Line2D object at 0x0000000033C5F250>]
```

```
## Text(0.5, 0, '$t$')
```

```
## Text(0, 0.5, '$y(t)$')
```

```
## Text(0.5, 1.0, 'Family of solution curves for ODE $y^\\prime = 1/2$')
```



3.2 Euler's Method

The simplest numerical technique for solving differential equations is Euler's method. It involves choosing a suitable step size h and an initial value $y(t_0) = y_0$, which are then used to estimate $y(t_1),\ y(t_2), \ldots$ by a sequence of values $y_i,\ i = 1, 2, \ldots$. Here use the notation $t_i = t_0 + ih$.

A method of accomplishing this is suggested by the Taylor's expansion

$$y(t+h) = y(t) + hy'(t) + \frac{1}{2!}h^2y''(t) + \frac{1}{3!}h^3y'''(t) + \cdots$$

or, in terms of the notation introduced above:

$$y_{i+1} = y_i + hy_i' + \frac{1}{2!}h^2y_i'' + \frac{1}{3!}h^3y_i''' + \cdots \tag{3.2}$$

By the differential equation (3.2), we have:

$$y_i' = f(t_i, y_i)$$

which when substituted in (3.2) yields:

$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{1}{2!}h^2f'(t_i, y_i) + \frac{1}{3!}h^3f''(t_i, y_i) + \cdots \tag{3.3}$$

and so if we truncate the Taylor series (3.3) after the term in h, we have the approximate formula:

$$y_{i+1} = y_i + hf(t_i, y_i) \tag{3.4}$$

This is a difference formula which can be evaluated step by step. This is the formula for **Euler's (or Euler-Cauchy)** method. Thus given (t_0, y_0) we can calculate (t_i, y_i) for $i = 1, 2, \cdots, n$. Since the new value y_{i+1} can be calculated from known values of t_i and y_i, this method is said to be **explicit**.

3.2.1 Error in Euler's Method

Each time we apply an equation such as(3.4) we introduce two types of errors:

- Local truncation error introduced by ignoring the terms in h^2, $h^3, \cdots$ in equation (3.2). For Euler's method, this error is

$$E = \frac{h^2}{2!} y_i''(\xi), \quad \xi \in [t_i, t_{i+1}],$$

  i.e. $\epsilon_E = \mathcal{O}(h^2)$. Thus the local truncation error per step is $\mathcal{O}(h^2)$.
- A further error introduced in y_{i+1} because y_i is itself in error. The size of this error will depend on the function $f(t, y)$ and the step size h.

The above errors are introduced at each step of the calculation.

3.2.2 Example

Apply the Euler's method to solve the simple equation:

$$\frac{dy}{dt} = t + y, \qquad y(0) = 1$$

(Exercise: Solve the equation analytically and show that the analytic solution is $y = 2e^t - t - 1$.)

Solution:

Here $f(t_i, y_i) = t_i + y_i$. With $h = 0.1$, and $y_0 = 1$ we compute y_1 as:

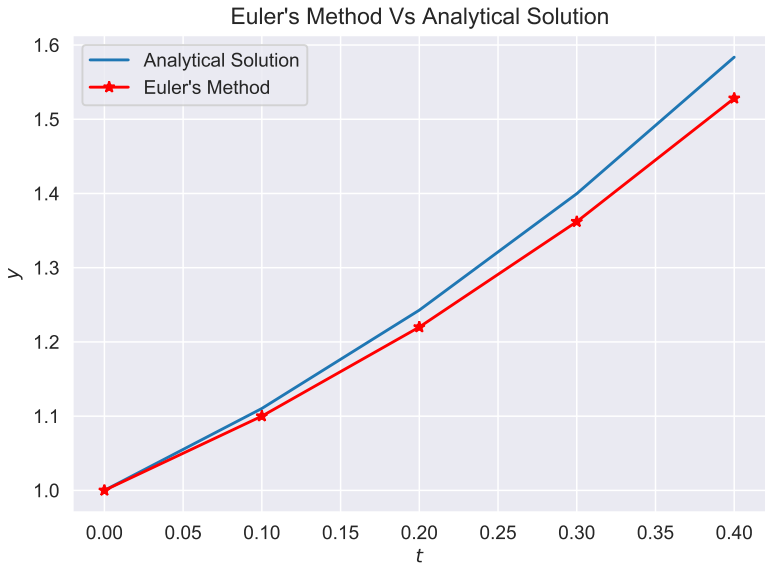
$$y_1 = y_0 + hf(t_0, y_0) = 1 + 0.1(0 + 1) = 1.1$$

The numerical results of approtimate solutions at subsequent points $t_1 = 0.2, ...$ can be computed in a similar way, rounded to 3 decimal, to obtain places.

t	y	$y' = f(t, y)$	$y'h$
0	1.000	1.000	0.100
0.1	1.100	1.200	0.120
0.2	1.220	1.420	0.142
0.3	1.362	1.662	0.166
0.4	1.528	1.928	0.193

The analytical solution at $t = 0.4$ is 1.584. The numerical value is 1.528 and hence the error is about 3.5%. The accuracy of the Euler's method can be improved by using a smaller step size h. Another alternative is to use a more accurate algorithm.

```
## <Figure size 1300x900 with 0 Axes>
## [<matplotlib.lines.Line2D object at 0x000000003318BF70>]
## [<matplotlib.lines.Line2D object at 0x00000000331B7D00>]
## Text(0.5, 0, '$t$')
## Text(0, 0.5, '$y$')
## <matplotlib.legend.Legend object at 0x0000000033264E50>
```



3.3 Modified Euler's Method

A fundamental source of error in Euler's method is that the derivative at the beginning of the interval is assumed to apply across the entire subinterval.

There are two ways we can modify the Euler method to produce better results. One method is due to Heun (**Heun's method**) and is well documented in numerical text books. The other method we consider here is called the **improved polygon** (or **modified Euler**) method.

The modified Euler technique uses Euler's method to predict the value of y at the midpoint of the interval $[t_i, t_{i+1}]$:

$$y_{i+\frac{1}{2}} = y_i + f(t_i, y_i)\frac{h}{2}. \tag{3.5}$$

Then this predicted value is used to estimate a slope at the midpoint:

$$y'_{i+\frac{1}{2}} = f(t_{i+1/2}, y_{i+1/2}), \tag{3.6}$$

which is assumed to represent a valid approximation of the average slope for the entire subinterval. This slope is then used to extrapolate linearly from t_i to x_{i+1} using Euler's method to obtain:

$$y_{i+1} = y_i + f(t_{i+1/2}, y_{i+1/2})h \tag{3.7}$$

For the modified Euler method, the truncation error can be shown to be:

$$\epsilon_E = -\frac{h^3}{12}y_i'''(\xi), \quad \xi \in [t_i, t_{i+1}] \tag{3.8}$$

Note: $t_{i+\frac{1}{2}} = t_i + \frac{1}{2}$

3.3.0.1 Example

Solve

$$\frac{dy}{dt} = t + y, \quad y(0) = 1, \quad h = 0.1$$

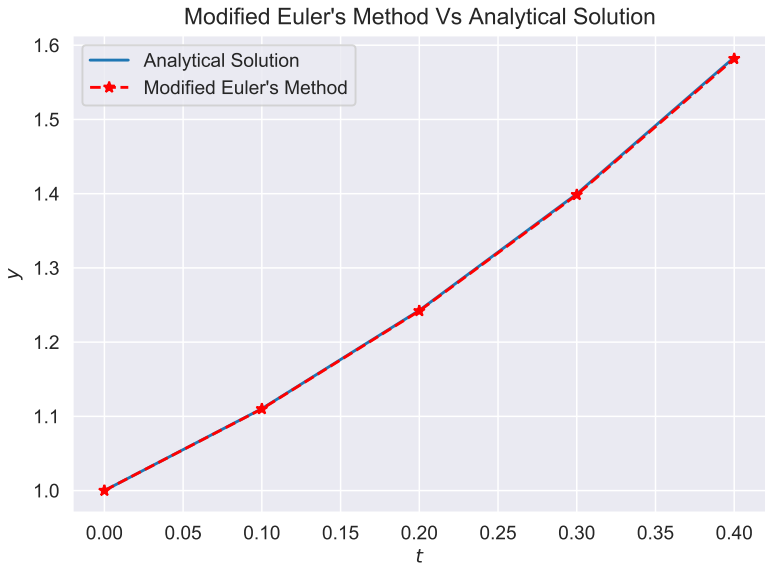
using the modified Euler's method described above.

Solution:

t_i	y_i	$y_{i+1/2}$	$y'_{i+1/2}$	$y'_{i+1/2}h$
0	1.000	1.050	1.100	0.110
0.1	1.110	1.1705	1.3205	0.13205
0.2	1.24205	1.1705	1.3205	0.13205
0.3	1.39847	1.31415	1.56415	0.15641
0.4	1.58180	1.48339	1.83339	0.18334

The numerical solution is now 1.5818 which much more accurate that the result obtained using Euler's method. In this case the error is about 0.14%.

```
## <Figure size 1300x900 with 0 Axes>
## [<matplotlib.lines.Line2D object at 0x0000000033C81AF0>]
## [<matplotlib.lines.Line2D object at 0x0000000033C81F40>]
## Text(0.5, 0, '$t$')
## Text(0, 0.5, '$y$')
## <matplotlib.legend.Legend object at 0x00000000337A39A0>
```



3.4 Runge-Kutta Methods

Runge and Kutta were German mathematicians. They suggested a group of methods for numerical solutions of ODEs.

The general form of the Runge-Kutta method is:

$$y_{i+1} = y_i + h\phi(t_i, y_i; h), \tag{3.9}$$

where $\phi(t_i, y_i; h)$ is called the increment function.

In Euler's method, $\phi(t_i, y_i; h) = f(t_i, y_i) = y_i'$, i.e we are using the slope at the point t_i to extrapolate y_i and obtain y_{i+1}. In the modified Euler's method:

$$\phi(t_i, y_i; h) = f(t_{i+\frac{1}{2}}, y_{i+\frac{1}{2}}) = y'_{i+\frac{1}{2}}$$

The increment function can be written in a general form as:

$$\phi = w_1 k_1 + w_2 k_2 + \cdots + w_n k_n \tag{3.10}$$

where the k's are constants and the w's are weights.

3.4.1 Second Order Runge-Kutta Method

The second order R-K method has the form:

$$y_{i+1} = y_i + (w_1 k_1 + w_2 k_2), \tag{3.11}$$

where

$$\begin{aligned} k_1 &= hf(t_i, y_i) && (3.12)\\ k_2 &= hf(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}), && (3.13) \end{aligned}$$

and the weights $w_1 + w_2 = 1$. If $w_1 = 1$, then $w_2 = 0$ and we have Euler's method. If $w_2 = 1$, then $w_1 = 0$ we have the Euler's improved polygon method:

$$\begin{aligned} y_{i+1} &= y_i + k_2 && (3.14)\\ &= y_i + hf(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}), && (3.15) \end{aligned}$$

If $w_1 = w_2 = \frac{1}{2}$, then we have:

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{2}(k_1 + k_2), && (3.16)\\ k_1 &= hf(t_i, y_i) && (3.17)\\ k_2 &= hf(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}), && (3.18) \end{aligned}$$

called Heun's method.

3.4.2 Fourth Order Runge-Kutta Method

The **classical fourth order R–K method** has the form:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{3.19}$$

where

$$k_1 = hf(t_i, y_i) \tag{3.20}$$
$$k_2 = hf(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}) \tag{3.21}$$
$$k_3 = hf(t_i + \frac{h}{2}, y_i + \frac{k_2}{2}) \tag{3.22}$$
$$k_4 = hf(t_i + h, y_i + k_3), \tag{3.23}$$

This is the most popular R–K method. It has a local truncation error $\mathcal{O}(h^4)$

3.4.2.1 Example

Solve the DE $y' = t + y$, $y(0) = 1$ using 4^{th} order Runge–Kutta method. Compare your results with those obtained from Euler's method, modified Euler's method and the actual value. Determine $y(0.1)$ and $y(0.2)$ only.

The solution using Runge-Kutta is obtained as follows:

For y_1 :

$$k_1 = 0.1(0 + 1) = 0.1 \tag{3.24}$$
$$k_2 = 0.1\left(\left(0 + \frac{0.1}{2}\right) + \left(1 + \frac{0.1}{2}\right)\right) = 0.01 \tag{3.25}$$
$$k_3 = 0.1\left(\left(0 + \frac{0.1}{2}\right) + \left(1 + \frac{0.11}{2}\right)\right) = 0.1105 \tag{3.26}$$
$$k_4 = 0.1((0 + 0.1) + (1 + 0.1105)) = 0.1211 \tag{3.27}$$

and therefore:

$$y_1 = y_0 + \frac{1}{6}(0.1 + 2(0.01) + 2(0.1105) + 0.1211) = 1.1103$$

A similar computation yields

$$y(0.2) = y_2 = 1.1103 + \frac{1}{6}(0.1210 + 2(0.1321) + 2(0.1326) + 0.1443 = 1.2428$$

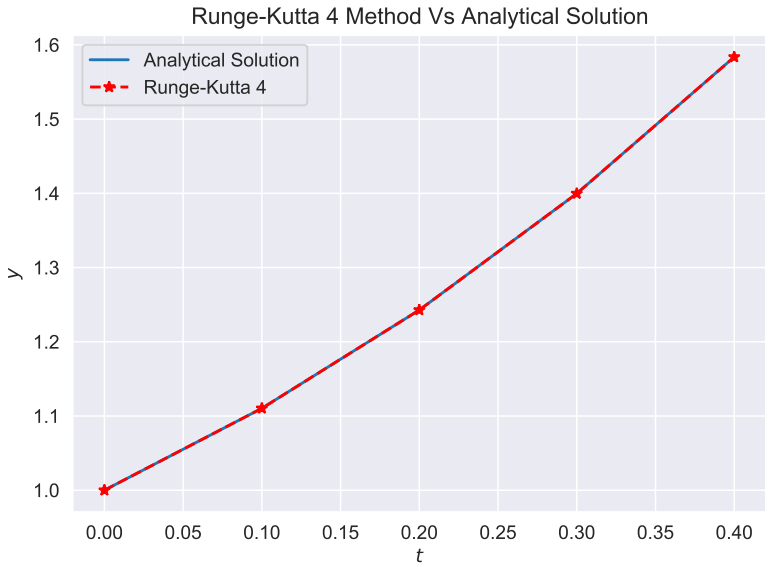
A table for all the approximate solutions using the required methods is:

t	Euler	Modified Euler	4^{th} Order RK	Actual value
0.1	1.1000000	1.1100000	1.1103417	1.1103418
0.2	1.2300000	1.2420500	1.2428052	1.2428055

```
## <Figure size 1300x900 with 0 Axes>
```

```
## [<matplotlib.lines.Line2D object at 0x0000000035EF17C0>]
```

```
## [<matplotlib.lines.Line2D object at 0x0000000035EF1C10>]
## Text(0.5, 0, '$t$')
## Text(0, 0.5, '$y$')
## <matplotlib.legend.Legend object at 0x0000000035EF1FD0>
```



3.5 Multistep Methods

As previously, Euler's method, Modified Euler's method and Runge-Kutta methods are single-step methods. They work by computing each successive value y_{i+1} only utilising information from the preceding value y_n. Another approach are *multistep methods*, where values from several computed previously computed steps are used to obtain y_{i+1}. There are numerous methods using this approach, however, for the purpose of this course we will only consider one - the **Adam Bashforth Method**.

3.5.1 Adam-Bashforth-MoultonMethod

This is a multistep method is similar to the Modified Euler's method in that it is a predictor-corrector method, i.e. uses one formula to predict a value y'_{i+1}, which is then used to obtain a corrected value y_{i+1}. The predictor in this

method is the Adams-Bashforth formula. Specifically,

$$\begin{aligned} y^*_{i+1} &= y_i + \frac{h}{24}(55y'_i - 59y'_{i-1} + 37y'_{i-2} - 9y'_{i-3}), \\ y'_i &= f(t_i, y_i), \\ y'_{i-1} &= f(t_{i-1}, y_{i-1}), \\ y'_{i-2} &= f(t_{i-2}, y_{i-2}), \\ y'_{i-3} &= f(t_{i-3}, y_{i-3}), \end{aligned}$$

for $i \geq 3$, which is then substituted into the Adams-Moulton corrector:

$$y_{i+1} = y_i + \frac{h}{24}(9y'_{i+1} + 19'_i - 5y'_{i-1} + y'_{i-2}) \tag{3.28}$$

$$y'_{i+1} = f(t_{i+1}, y^*_{i+1}). \tag{3.29}$$

Note that Equation (3.28) requires that we know the initial values of y_0, y_1, y_2 and y_3 in order to obtain y_4. The value y_0 is the initial condition. Since Adams-Bashforth method is of $\mathcal{O}(h^5)$, we need a high order accurate method to first obtain y_1, y_2 and y_3. Therefore, we compute these values using the RK-4 formula.

3.5.1.1 Example

Use the Adam-Bashforth method with $h = 0.2$ to obtain an approximation to $y(0.8)$ for the IVP:

$$y' = t + y - 1, \quad y(0) = 1.$$

Solution:

Using the RK-4 method to get started, we obtain the following:

$$y_1 = 1.0214, \quad y_2 = 1.09181796, \quad y_3 = 1.22210646.$$

Since $h = 0.2$, we know that $t_1 = 0.2, t_2 = 0.4, t_3 = 0.6$ and $f(t, y) = t + y - 1$. Now we can proceed:

$$\begin{aligned} y'_0 &= f(t_0, y_0) = 0 + 1 - 1 = 0, \\ y'_1 &= f(t_1, y_1) = 0.2 + 1.0214 - 1 = 0.2214, \\ y'_2 &= f(t_2, y_2) = 0.4 + 1.09181796 - 1 = 0.49181796, \\ y'_3 &= f(t_3, y_3) = 0.6 + 1.22210646 - 1 = 0.82210646. \end{aligned}$$

Now we can compute the predictor y^*_4:

$$y'_4 = y_3 + \frac{0.2}{24}(55y'_3 - 59y'_2 + 37y'_1 - 9y'_0) = 1.42535975.$$

Next, we need y_4':

$$y_4' = f(t_4, y_4^*) = 0.8 + 1.42535975 - 1 = 1.22535975.$$

Finally, this gives y_4 by:

$$y_4 = y_3 + \frac{0.2}{24}(9y_4' + 19_3' - 5y_2' + y_1') = 1.42552788.$$

The exact solution of this ODE at $y(0.8)$ is 1.42554093.

3.5.2 Advantages of Multistep Methods

There are a number of decisions to make when choosing a numerical method to solve a differential equation. While single step explicit methods such as RK-4 are often chosen due to their accuracy and easily programmable implementation, the right hand side of the equation needs to be evaluated many times. In the case of RK-4, the method is required to make four function evaluations at each step. On the Implicit side, if the function valuations in the previous step have been computed and stored, then a multistep method would require only one new function evaluation at each step - saving computational time.

In general the Adam-Bashforth method requires slightly more than one quarter of the number of function evaluations required for the RK-4 method.

3.6 Systems of First Order ODEs

A nth order system of first order initial value problems can be expressed in the form:

$$\begin{aligned}
\frac{dy_1}{dt} &= f_1(t, y_1, y_2, \cdots, y_n), \quad y_1(t_0) = \alpha_1 \\
\frac{dy_2}{dt} &= f_2(t, y_1, y_2, \cdots, y_n), \quad y_2(t_0) = \alpha_2 \\
&\vdots \\
\frac{dy_n}{dt} &= f_n(t, y_1, y_2, \cdots, y_n), \quad y_n(t_0) = \alpha_n,
\end{aligned}$$

for $t_0 \le t \le t_n$.

The methods we have seen so far were for a single first order equation, in

which we sought the solution $y(t)$. Methods to solve first order systems of IVP are simple generalization of methods for a single equations, bearing in mind that now we seek n solutions $y_1,\ y_2, \ldots, y_n$ each with an intial condition $y_k(t_0); k = 1, \ldots, n$ at the points $t_i,\ i = 1, 2. \ldots$.

3.6.1 R-K Method for Systems

Consider the system of two equations:

$$\frac{dy}{dt} = f(t, y, z), \quad y(0) = y_0 \tag{3.30}$$

$$\frac{dz}{dt} = g(t, y, z), \quad z(0) = z_0. \tag{3.31}$$

Let $y = y_1,\ z = y_2,\ f = f_1,$ and $g = f_2$. The fourth order R-K method would be applied as follows. For each $j = 1, 2$ corresponding to solutions $y_{j,i}$, compute

$$k_{1,j} = hf_j(t_i, y_{1,i}, y_{2,i}), \quad j = 1, 2 \tag{3.32}$$

$$k_{2,j} = hf_j(t_i + \frac{h}{2},\ y_{1,i} + \frac{k_{1,1}}{2},\ y_{2,i} + \frac{k_{1,2}}{2}) \quad j = 1, 2 \tag{3.33}$$

$$k_{3,j} = hf_j(t_i + \frac{h}{2},\ y_{1,i} + \frac{k_{2,1}}{2},\ y_{2,i} + \frac{k_{2,2}}{2}), \quad j = 1, 2 \tag{3.34}$$

$$k_{4,j} = hf_j(t_i + h,\ y_{1,i} + k_{3,1},\ y_{2,i} + k_{3,2}), \quad j = 1, 2 \tag{3.35}$$

and:

$$y_{i+1} = y_{1,i+1} = y_{1,i} + \frac{1}{6}(k_{1,1} + 2k_{2,1} + 2k_{3,1} + k_{4,1}) \tag{3.36}$$

$$z_{i+1} = y_{2,i+1} = z_i + \frac{1}{6}(k_{1,2} + 2k_{2,2} + 2k_{3,2} + k_{4,2}). \tag{3.37}$$

Note that we must calculate $k_{1,1},\ k_{1,2},\ k_{2,1},\ k_{2,2},\ k_{3,1},\ k_{3,2},\ k_{4,1},\ k_{4,2}$ in that order.

3.7 Converting an n^{th} Order ODE to a System of First Order ODEs

Consider the general second order initial value problem

$$y'' + ay' + by = 0, \quad y(0) = \alpha_1, \quad y'(0) = \alpha_2$$

If we let

$$z = y', \quad z' = y''$$

then the original ODE can now be written as

$$y' = z, \quad y(0) = \alpha_1 \tag{3.38}$$
$$z' = -az - by, \quad z(0) = \alpha_2 \tag{3.39}$$

Once transformed into a system of first order ODEs the methods for systems of equations apply.

3.7.0.1 Exercise

Solve the second order differential equation:

$$y'' + 3ty' + 2t^2y = 0, \quad y(0) = 3, \quad y'(0) = 1$$

(i) Second order R–K method (ii) 4th order R–K. Use $h = 0.1$. Do only two steps.

3.7.1 Exercises

Use (i) Euler's method (ii) modified Euler's formula to solve the following IVP;

- $y' = \sin(t + y), \quad y(0) = 0$
- $y' = yt^2 - y, \quad y(0) = 1$ for $h = 0.2$ and $h = 0.1$.
- Determine $y(0.4)$ for each of the above IVP.
- Use Richardson's extrapolation to get improved approximations to the solutions at $t = 0.4$
- If f is a function of t only, show that the fourth-order Runge-Kutta formula, applied to the differential equation $dy/dt = f(t)$ is equivalent to the use of Simpson's rule (over one interval) for evaluating $\int_0^t f(t)dt$.
- Use fourth order Runge–Kutta method to solve the following IVPs:
  - $y' = 2ty, \quad y(0) = 1$
  - $y' = 1 + y^2, \quad y(0) = 0$, Use $h = 0.2$ and determine the solutions at $t = 0.4$.
- Solve the following systems of IVPs:
  - $y' = yz, \quad z' = tz, \quad y(0) = 1, \quad z(0) = -1$
  - $y' = t - z^2, \quad z' = t + y, \quad y(0) = 1 \quad z(0) = 2$, using (i) Euler's method (ii) Second order Runge-Kutta with $h = 0.1$. Compute y and z, at $t = 0.2$.
- Use Euler's method to solve the differential equation:

  $$y' = 1 + y^2,$$

  on the domain $[0,\ 1]$ with the initial condition of (a) $y_0 = 0$ and (b) $y_0 = 1$. Plot these solutions along with the exact solution. Use step sizes of $h = 0.1$ and $h = 0.05$.

- Given the IVP $y' = (t+y-1)^2$, $y(0) = 2$. Using the Modified Euler's method with $h = 1$ and $h = 0.05$, obtain approximate solutions of the solution at $t = 0.5$. Compare these values with the analytical solution.

4

Partial Differential Equations

Partial differential equations are incredibly powerful modelling tools. They are used to mathematically describe phenomenological behaviour. For example, how heat and mass is transferred, how medicine traverses the human body and permeates organs/muscle tissue differently, the motion of fluid, general physics simulation in game design, chemical reaction and interactions, etc.

Partial differential equations (PDEs) form a class of equations which extend those of ordinary differential equations to a number of dimensions. In this course we will focus on PDEs with one spatial dimension and one temporal dimension, with the dependent variable typically denoted by $u(x,t)$, where x is the spatial variable and t represents time. We shall also restrict our analysis to PDEs on a regular domain, $x \in [a,b]$, however much research has been conducted to solving PDEs on arbitrary domains/geometries. A partial derivative of our dependent variable u towards x is written $\frac{\partial u}{\partial x}$ or by making use of the short hand u_x

4.1 Classification of Partial Differential Equations

We may classify a PDE given the general form of a two dimensional, second-order PDE

$$au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0, \tag{4.1}$$

by the discriminant

$b^2 - 4ac > 0$	: hyperbolic,
$b^2 - 4ac = 0$	: parabolic,
$b^2 - 4ac < 0$	: elliptic.

Unfortunately, not all PDEs fit nicely into one of these categories, if the coefficients are variable then the PDE may exhibit properties from different classifications under varying conditions. Each of these categories are associated with typical behaviour, summarised

1. Hyperbolic: Time dependent PDEs which exhibit propagational behaviour, like wave motion, and *do not* tend toward a steady state. (Wave Equation, Advection Equation, etc)

$$u_{tt} = \omega^2 u_{xx}$$

2. Parabolic: Time dependent PDES which exhibit dissipative behaviour, like heat diffusion, and *do* tend toward a steady state. (Heat, diffusion Equation, Reaction-Diffusion Equation, etc)

$$u_t = D u_{xx}$$

3. Elliptic: Steady PDEs (not time dependent) that are in a state of equilibrium. (Laplace's equation, Poisson's equation, etc)

$$u_{xx} + u_{yy} = 0$$

This course primarily focuses on the parabolic type of equations and are typically the easiest to solve and analyse.

4.2 Time-Dependent Problems

Solution approaches to time-dependent problems are often hinged on the domain of the prescribed problem. If the problem is prescribed on a infinite (or semi-infinite) domain then transform and analytic methods are appropriate, or the infinite domain needs to be approximated by a large finite domain to render the problem amenable to numerical methods. As suggested by the domain truncation; problems on bounded domains are well suited to be solved by numerical methods.

Figure 4.1 below illustrates the region in which we want to solve our problem.

Consider the heat equation on this domain. The problem is prescribed by

$$u_t = D u_{xx}, \tag{4.2}$$

subject to the initial condition

$$u(x, 0) = f(x), \tag{4.3}$$

and Dirichlet boundary conditions,

$$u(a, t) = \alpha, \qquad u(b, t) = \beta, \tag{4.4}$$

or the Neumann boundary conditions,

$$u_x(a, t) = \alpha, \qquad u_x(b, t) = \beta. \tag{4.5}$$

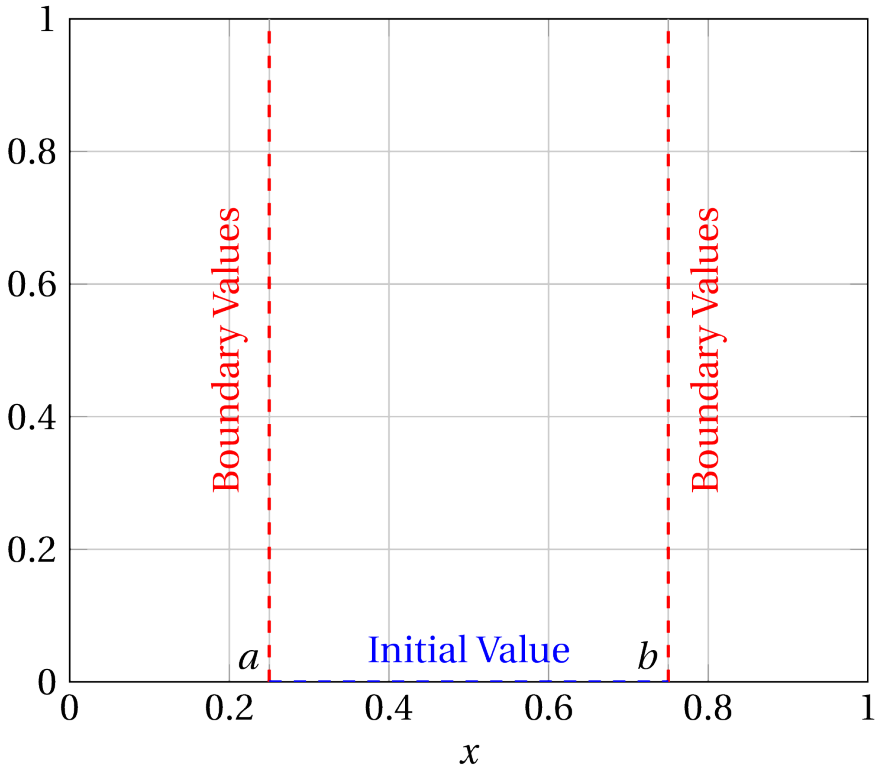


FIGURE 4.1 Initial-boundary value problem domain for a time-dependent PDE in one spatial dimension.

Our approach to numerically solving this equation is to begin by discretising the problem region into a discrete grid, initialising our solution from the initial condition, and then computing the solution at the next time step (row) through an update rule prescribed by the equation and boundary conditions. The boundary conditions (Dirichlet vs Neumann) will be discussed in depth in Section 4.3.2.

4.3 Fully Discrete Methods

Analytic solutions for nonlinear partial differential equations are often difficult or impossible to compute. Numerical methods, such as the Finite Difference Method, provide a robust framework for the approximation of the solution of a PDE. We leverage analysis of the resulting numerical schemes to confirm the properties of the obtained approximate solution; such as convergence rate (accuracy), consistency and stability.

4.3.1 Explicit Finite Difference Method

The explicit finite difference method is a quick and easy approach to solving parabolic PDEs. It is usually a good starting point for getting an idea of the behaviour of a problem, however it suffers from strict stability criteria and can sometimes be unconditionally unstable (meaning no parameter choice will lead to a stable scheme).

Example 4.1. For the moment we consider the heat equation (4.2). The domain $[x,t] = [a,b] \times [0,\infty]$ is discretised into $N+1$ spatial points,

$$x = [x_0 = a, x_1 = a + \Delta x, \dots, x_{N-1} = a + (N-1)\Delta x, x_N = b], \tag{4.6}$$

and temporal discretisation

$$t = [t_0 = 0, t_1 = \Delta t, \dots, t_{n-1} = (n-1)\Delta t, t_n = n\Delta t, \dots]. \tag{4.7}$$

The spatial discretisation for $N = 4$ is illustrated in Figure 4.2 below.

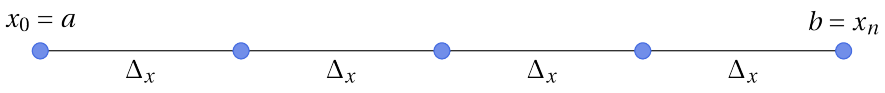


FIGURE 4.2 Schematic Spatial Discretisation for $N = 4$.

The entire discrete problem may be visualised as a grid, as in Figure 4.3

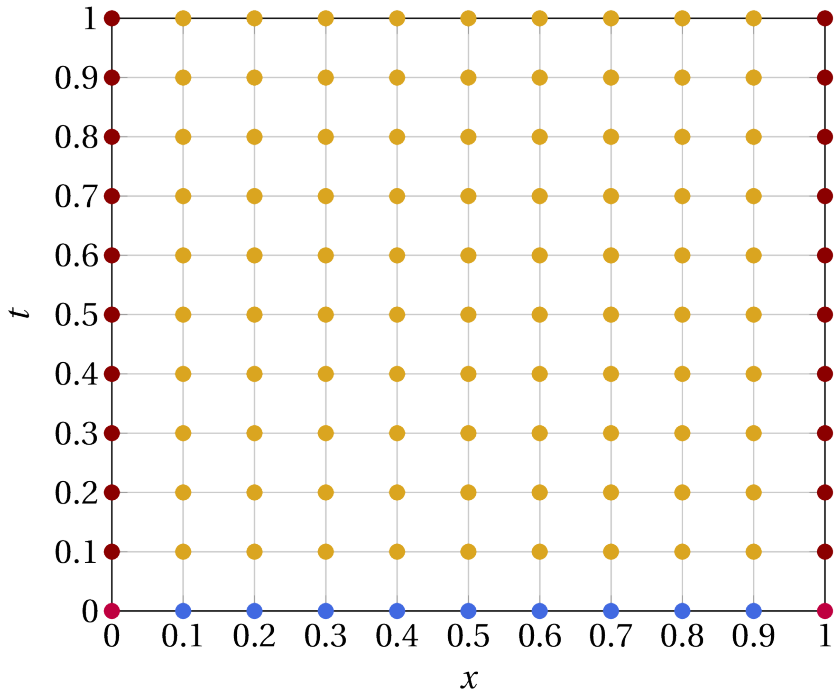


FIGURE 4.3 Schematic Discretisation for $N = 8$. Blue nodes indicate values known from the initial condition. Red nodes indicate values known from boundary values. Purple nodes indicate values known from both initial and boundary conditions (and should be in agreement with one another).

In order to discretise the differential operators we consider the Taylor expansion

$$u(x, t + \Delta t) = u(x, t) + \frac{\Delta t}{2} u_t(x, t) + \mathcal{O}(\Delta t^2), \tag{4.8}$$

As a shorthand we denote the discrete point of the solution $u(x_i, t_n)$ by u_i^n and note that i represents a spatial index and n a temporal index. Hence the temporal derivative is approximated by

$$u_t(x_i, t_n) = \frac{u(x_i, t_{n+1}) - u(x_i, t_n)}{\Delta t} + \mathcal{O}(\Delta t) = \frac{u_i^{n+1} - u_i^n}{\Delta t} + \mathcal{O}(\Delta t). \tag{4.9}$$

Similarly the spatial derivative may be approximated by

$$u_{xx}(x_i, t_n) = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \mathcal{O}(\Delta x^2). \tag{4.10}$$

Discretising equation (4.2) via equations (4.9) and (4.10) we obtain the following recursive relation (difference equation),

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = D \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2). \tag{4.11}$$

Rearranging and dropping the truncation errors, we have,

$$u_i^{n+1} = u_i^n + \frac{D\Delta t}{\Delta x^2} \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n \right), \quad i = 1, \ldots, n-1. \tag{4.12}$$

From the above we note that, in order to compute the value at the next time step, $n + 1$, for spatial location i we require points $i - 1$, i and $i + 1$ all from the known time step n. Figures 4.4 and 4.5 below illustrate different stencils for three different schemes. The explicit stencil requires only explicitly known values to iterate the recursive relation defined above. The implicit and Crank-Nicolson stencil requires three unknown points to calculate one value at the next time step, essentially we have one equation and three unknowns. How we deal with this is discussed later in Section 4.3.3.

The local truncation error, derived from Taylor's series, for this scheme is $\mathcal{O}(\Delta t)+\mathcal{O}(\Delta x^2)$, meaning the scheme is first order accurate in time and second order accurate in space.

We note that equation (4.12) is only valid for $i = 1, \ldots, n-1$, what happens when $i = 0$ or $i = n$? Boundary conditions from the prescribed problem factor in and allow for the computation at each boundary.

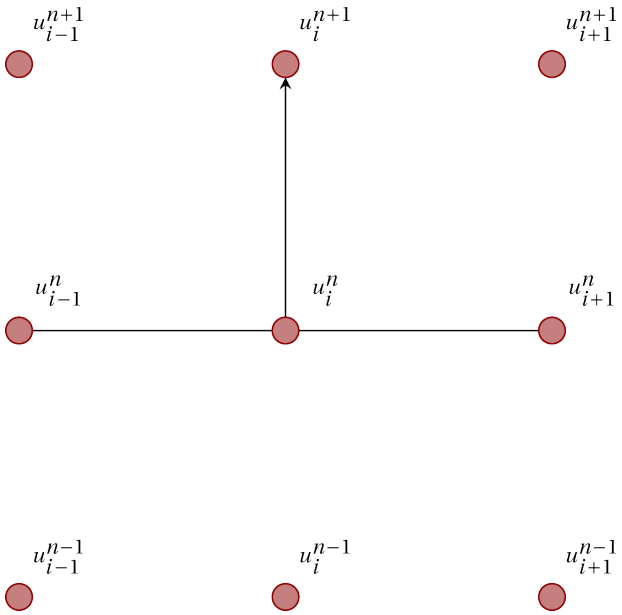


FIGURE 4.4 Explicit Finite Difference Stencil.

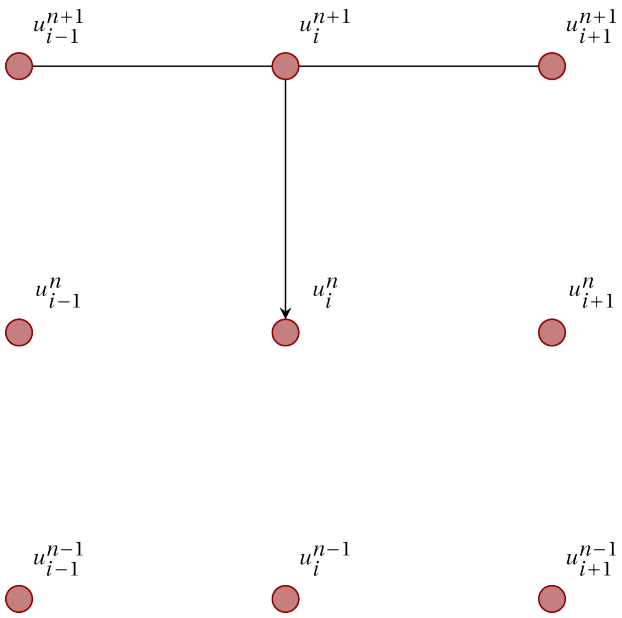


FIGURE 4.5 Implicit Finite Difference Stencil.

4.3.2 Boundary Conditions

Boundary conditions form an important part of the problem description, not only to render the problem numerically tractable, but also as a modelling device. We consider two classes of boundary conditions in this section.

4.3.2.1 Dirichlet Boundary Conditions

Dirichlet boundary conditions, sometimes referred to as clamped boundary conditions, prescribe a constant value or time dependent function at the boundary. This means that the value of our solution at $x = a$ or $x = b$ is known for all values of time, t. Continuing on from the explicit formulation in the previous section, we now see how to incorporate Dirichlet boundary conditions into our numerical scheme.

Regardless of the domain $[a, b]$, the boundary conditions are imposed on the numerical scheme at x_0 and x_N (red nodes in Figure 4.3. According to equation (4.4) we have,

$$u(a, t) = \alpha, \qquad u(b, t) = \beta, \tag{4.13}$$

which is discretised as

$$u_0 = \alpha, \qquad u_N = \beta. \tag{4.14}$$

These values may now be incorporated into the update scheme, equation (4.12), when $i = 1$ and $i = n - 1$ due to the width of the stencil.

Now that we have a mechanism to update all values in our discretisation, we can formulate the scheme in vector form. Let

$$\underline{u}^n = \begin{pmatrix} u_0^n \\ u_1^n \\ \vdots \\ u_N^n \end{pmatrix}. \tag{4.15}$$

By grouping like terms in equation (4.12) we may rewrite that scheme in matrix form,

$$\underline{u}^{n+1} = \mathbf{A}\underline{u}^n, \tag{4.16}$$

where

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & ... & 0 \\ \Lambda_1 & \Lambda_2 & \Lambda_1 & 0 & ... & 0 \\ 0 & \Lambda_1 & \Lambda_2 & \Lambda_1 & ... & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & ... & \Lambda_1 & \Lambda_2 & \Lambda_1 & 0 \\ 0 & ... & 0 & \Lambda_1 & \Lambda_2 & \Lambda_1 \\ 0 & ... & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{4.17}$$

and

$$\Lambda_1 = \frac{D\Delta t}{\Delta x^2}, \tag{4.18}$$

$$\Lambda_2 = 1 - \frac{2D\Delta t}{\Delta x^2}. \tag{4.19}$$

Notice that the top and bottom rows enforce the boundary values.

Exercise 4.1. Verify by hand for $N = 4$ that the above matrix equation indeed recovers the correct equations.

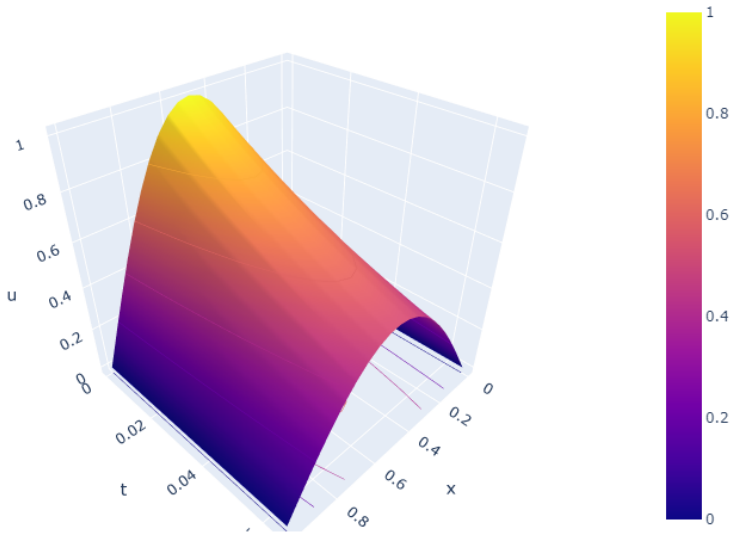
Example 4.2. Using a forward time central space finite difference scheme on the heat equation given by:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2},$$

where:

$$u(x,0) = \sin(\pi x), \quad u(0,t) = 0, \quad u(1,t) = 0,$$
$$\Delta x = 0.05, \quad \Delta t = 0.0013 \quad \text{and} \quad N = 50.$$

we obtain the following numerical solution plotted below:



4.3.2.2 Neumann Boundary Conditions

Neumann boundary conditions, prescribe a constant value or time dependent function for the **derivative** of the dependent variable at the boundaries. Often a modelling choice is to require that no heat/mass leave the system and so heat/mass at the boundary will bounce back into the problem domain, this is known as zero-flux boundary conditions, mathematically represented by the derivative of the function at the boundaries being 0 for all time, t.

Implementation of derivative boundary conditions is slightly more complicated. Consider the boundary conditions (equation (4.5)),

$$u_x(a,t) = \alpha, \qquad u_x(b,t) = \beta. \tag{4.20}$$

We may discretise the boundary conditions using a central finite difference formula

$$u_x(x_i,t_n) \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x}, \tag{4.21}$$

evaluating the above at the boundary $x = a$ or $i = 0$, we have,

$$\frac{u_1 - u_{-1}}{2\Delta x} = \alpha. \tag{4.22}$$

We have introduced a 'ghost' point at $i = -1$,

$$u_{-1} = u_1 - 2\Delta x\alpha. \tag{4.23}$$

Similarly for $x = b$ or $i = N$,

$$\frac{u_{N+1} - u_{N-1}}{2\Delta x} = \beta. \tag{4.24}$$

Again we have a 'ghost' point at $i = N + 1$,

$$u_{N+1} = u_{N-1} + 2\Delta x\beta. \tag{4.25}$$

Using these ghost points we may now evaluate our update scheme (equation (4.12)) at points $i = 0$ (which requires u^n_{-1}) and $i = N$ (which requires u^n_{N+1}).

We may again formulate the problem in matrix form, taking into account the boundary conditions. In the case of Neumann boundary conditions we have,

$$\underline{u}^{n+1} = \mathbf{B}\underline{u}^n + \underline{c}, \tag{4.26}$$

where

$$\mathbf{B} = \begin{pmatrix} \Lambda_2 & 2\Lambda_1 & 0 & 0 & ... & 0 \\ \Lambda_1 & \Lambda_2 & \Lambda_1 & 0 & ... & 0 \\ 0 & \Lambda_1 & \Lambda_2 & \Lambda_1 & ... & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & ... & \Lambda_1 & \Lambda_2 & \Lambda_1 & 0 \\ 0 & ... & 0 & \Lambda_1 & \Lambda_2 & \Lambda_1 \\ 0 & ... & 0 & 0 & 2\Lambda_1 & \Lambda_2 \end{pmatrix}, \tag{4.27}$$

and

$$\underline{c} = \begin{pmatrix} -2\Lambda_1 \Delta x \alpha \\ 0 \\ \vdots \\ 0 \\ 2\Lambda_1 \Delta x \beta \end{pmatrix}, \tag{4.28}$$

and

$$\Lambda_1 = \frac{D\Delta t}{\Delta x^2}, \tag{4.29}$$

$$\Lambda_2 = 1 - \frac{2D\Delta t}{\Delta x^2}. \tag{4.30}$$

Exercise 4.2. Again, verify by hand for $N = 4$ that the above matrix equation indeed recovers the correct equations.

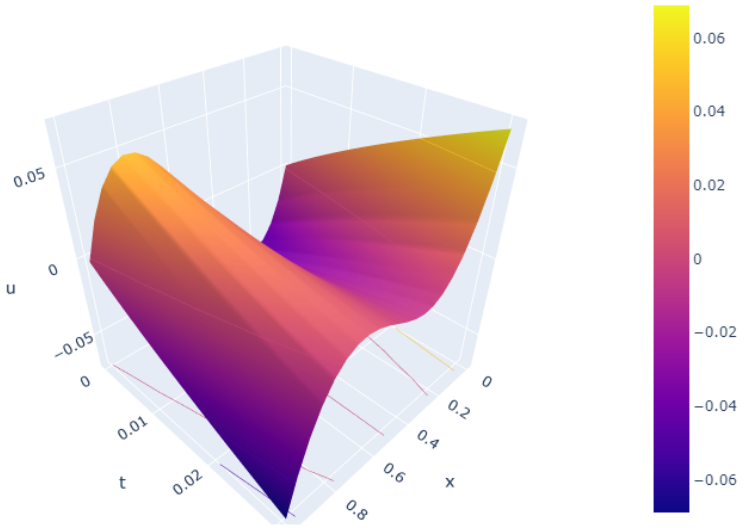
Example 4.3. Using a forward time central space finite difference scheme on the heat equation given by:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2},$$

where:

$$u(x,0) = x(1-x)(x-\tfrac{1}{2}), \quad u_x(0,t) = -\tfrac{1}{2},$$
$$u_x(1,t) = -\tfrac{1}{2}, \ \Delta x = 0.05, \quad \Delta t = 0.0013 \ \text{ and } \ N = 50.$$

we obtain the following numerical solution plotted below:



4.3.3 Implicit Finite Difference Method

As alluded to briefly in the previous section, explicit methods often suffer from strict stability conditions. This usually means that Δx and Δt need to be smaller than we might like them to be, leading to slow computation of our numerical simulation. Implicit methods are computationally more expensive, since we have to solve matrix inverses, but generally have much more appealing stability properties. Unconditionally stable implicit methods allow us to choose Δx and Δt however we like, which means we can dictate the spatial resolution we seek (Δx) while taking as large a time step as we like (Δt).

We construct an implicit finite difference scheme by simply evaluating u_{xx} at the implicit time step. Concretely we have,

$$u_t(x_i, t_n) = \frac{u_i^{n+1} - u_i^n}{\Delta t} + \mathcal{O}(\Delta t), \tag{4.31}$$

as before. However the spatial derivative is now approximated by

$$u_{xx}(x_i, t_n) = \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} + \mathcal{O}(\Delta x^2). \tag{4.32}$$

As previously mentioned and implied by the implicit stencil, we now have three

unknown variables (u_{i+1}^{n+1}, u_i^{n+1} and u_{i-1}^{n+1}) for each spatial point. However, once the boundary values are incorporated the entire row, $n+1$, can be computed at once. Using the above discretisation we can write our numerical scheme as

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = D\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2}. \tag{4.33}$$

Rearranging and collecting terms we have

$$-\frac{D\Delta t}{\Delta x^2}u_{i+1}^{n+1} + (1 + \frac{2D\Delta t}{\Delta x^2})u_i^{n+1} - \frac{D\Delta t}{\Delta x^2}u_{i-1}^{n+1} = u_i^n, \tag{4.34}$$

which, in matrix form, is given by

$$\mathbf{C}\underline{u}^{n+1} = \underline{u}^n, \tag{4.35}$$

where (for Dirichlet boundary conditions)

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 0 & 0 & ... & 0 \\ -\Lambda_1 & \Lambda_3 & -\Lambda_1 & 0 & ... & 0 \\ 0 & -\Lambda_1 & \Lambda_3 & -\Lambda_1 & ... & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & ... & -\Lambda_1 & \Lambda_3 & -\Lambda_1 & 0 \\ 0 & ... & 0 & -\Lambda_1 & \Lambda_3 & -\Lambda_1 \\ 0 & ... & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{4.36}$$

and

$$\Lambda_1 = \frac{D\Delta t}{\Delta x^2}, \tag{4.37}$$

$$\Lambda_3 = 1 + \frac{2D\Delta t}{\Delta x^2}. \tag{4.38}$$

Note that equation (4.35) is in the form of $\mathbf{A}\underline{x} = \underline{b}$, which we can solve using previously learned techniques.

Exercise 4.3. Verify (by hand with $N = 4$) that the above matrix equation correctly recovers the implicit finite difference scheme.

Exercise 4.4. Construct the matrix equation for Neumann boundary conditions for the implicit finite difference, and verify (by hand with $N = 4$) that the matrix equations correctly recover the numerical scheme.

4.4 Consistency, Stability and Convergence

This section briefly discusses the three main pillars of numerical analysis. More formal descriptions of these concepts will be given but we outline

the ideas here. Consistency is the idea that the numerical scheme that we construct will recover the governing {*equations*} *as our discretisation parameters (Δx and Δt) are limited to 0, in essence confirming that our numerical scheme is approximating the correct equation. In a similar vein, but more difficult to prove, convergence is the idea that as our discretisation parameters tend to 0, the {solution} obtained by our numerical approximation tends to the true (potentially unknown) solution. Finally, stability of a numerical scheme entails an analysis of how errors introduced by our approximation grow/shrink/change as our simulation is computed. Although these are three distinct ideas, they are related.*

4.4.1 Consistency

In order to prove consistency of a proposed numerical scheme, we are required to show that as our discretisation parameters tend to 0, we recover the original equation. We typically make use of the Taylor series (in reverse) to show this. Consistency is illustrate by way of example.

Example 4.4. Show that,

$$u_i^{n+1} = \frac{D\Delta t}{\Delta x^2} u_{i+1}^n + u_i^n \left(1 - \frac{2D\Delta t}{\Delta x^2} + \Delta t - \Delta t u_i^n\right) + \frac{D\Delta t}{\Delta x^2} u_{i-1}^n, \tag{4.39}$$

is consistent with

$$u_t = Du_{xx} + u(1-u). \tag{4.40}$$

Rearranging our numerical scheme we have,

$$u_i^{n+1} = u_i^n + \frac{D\Delta t}{\Delta x^2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) + \Delta t u_i^n \left(1 - u_i^n\right) \tag{4.41}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = D\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + u_i^n \left(1 - u_i^n\right). \tag{4.42}$$

Applying limits,

$$\lim_{\Delta x, \Delta t \to 0} \frac{u_i^{n+1} - u_i^n}{\Delta t} = \lim_{\Delta x, \Delta t \to 0} D\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \lim_{\Delta x, \Delta t \to 0} u_i^n \left(1 - u_i^n\right), \tag{4.43}$$

which recovers,

$$u_t = Du_{xx} + u(1-u), \tag{4.44}$$

from the Taylor series.

Exercise 4.5. Repeat the above example with an implicit finite difference approximation.

4.4.2 von Neumann Stability Analysis

In order to conduct a von Neumann stability analysis, we insert a 'trial' solution into our numerical scheme and assess how that trial solution behaves

with respect to the discretisation parameters. In truth what we are doing, is constructing a single (general) Fourier mode and proving that if any general Fourier mode is bounded above then they all are and the numerical scheme is stable.

Example 4.5. We again use the explicit finite difference approximation to the heat equation as an example of how to conduct a von Neumann Stability analysis. Consider

$$u_i^{n+1} = u_i^n + \frac{D\Delta t}{\Delta x^2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right). \tag{4.45}$$

We assume an Ansatz in the form of a Fourier mode

$$u_i^n = \zeta^n e^{Iqi\Delta x}, \tag{4.46}$$

where q is a real spatial wave number of the Fourier series, $I = \sqrt{-1}$ and i and n are the usual indices. Substituting this Ansatz into our numerical scheme we get

$$\zeta^{n+1}e^{Iqi\Delta x} = \zeta^n e^{Iqi\Delta x} + \frac{D\Delta t}{\Delta x^2}\zeta^n\left(e^{Iqi\Delta x}e^{Iq\Delta x} - 2e^{Iqi\Delta x} + e^{Iqi\Delta x}e^{-Iq\Delta x}\right), \tag{4.47}$$

dividing through by $\zeta^n e^{Iqi\Delta x}$ we get

$$\zeta = 1 + \frac{2D\Delta t}{\Delta x^2}\left(\frac{e^{Iq\Delta x} + e^{-Iq\Delta x}}{2} - 1\right). \tag{4.48}$$

Using Euler's formula and the double angle formula for $\cos(\theta)$ we get,

$$\zeta = 1 - \frac{4D\Delta t}{\Delta x^2}\sin^2\left(\frac{q\Delta x}{2}\right). \tag{4.49}$$

For our scheme to be stable, we require the amplification factor $|\zeta| < 1$. We should rather phrase this as, "What conditions need to be placed on Δx and Δt so that $|\zeta| < 1$?".

$$|\zeta| < 1 \Rightarrow |1 - \frac{4D\Delta t}{\Delta x^2}\sin^2\left(\frac{q\Delta x}{2}\right)| < 1, \tag{4.50}$$

$$\Rightarrow -1 < 1 - \frac{4D\Delta t}{\Delta x^2}\sin^2\left(\frac{q\Delta x}{2}\right) < 1, \tag{4.51}$$

$$\Rightarrow -2 < -\frac{4D\Delta t}{\Delta x^2}\sin^2\left(\frac{q\Delta x}{2}\right) < 0, \tag{4.52}$$

$$\Rightarrow 0 < \frac{4D\Delta t}{\Delta x^2}\sin^2\left(\frac{q\Delta x}{2}\right) < 2. \tag{4.53}$$

Since Δt, Δx and $\sin^2((\frac{q\Delta x}{2}))$ are always positive the left hand inequality is always satisfied. Hence, our stability is conditioned on

$$\frac{4D\Delta t}{\Delta x^2}\sin^2\left(\frac{q\Delta x}{2}\right) < 2,$$

but since $0 \leq \sin^2\left(\frac{q\Delta x}{2}\right) \leq 1$, then

$$\frac{4D\Delta t}{\Delta x^2} < 2, \tag{4.54}$$

$$\Rightarrow \frac{D\Delta t}{\Delta x^2} < \frac{1}{2}. \tag{4.55}$$

or in terms of Δt,

$$\Delta t < \frac{\Delta x^2}{2D} \tag{4.56}$$

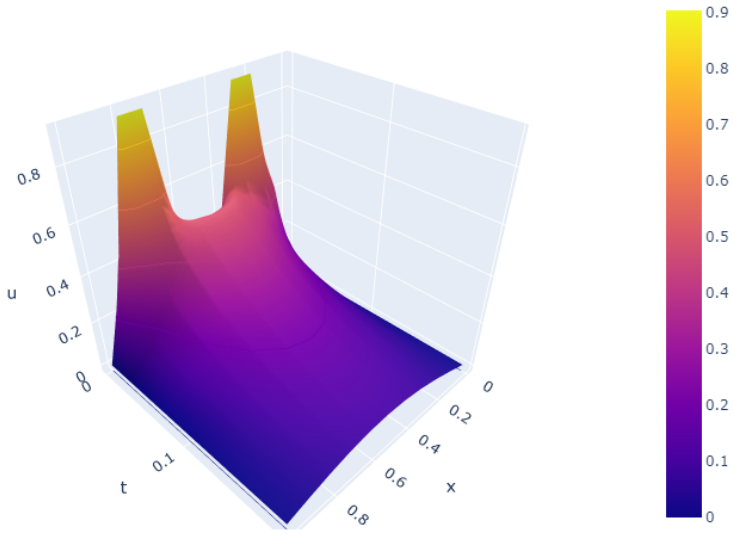
Example 4.6. In order to illustrate the stabilty concerns with the explicit finite difference scheme, consider the following example:

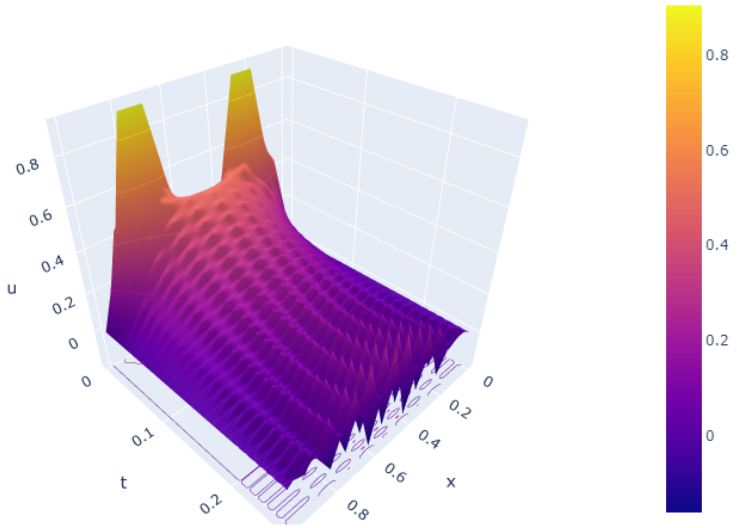
$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2},$$
$$u(0,t) = u(1,t) = 0, \quad u(x,0) = \sin^2(2\pi x), \quad \Delta x = 0.1.$$

Now applying our finite difference scheme, we consider two sets of parameters. In the first plot, we see a stable solution where $\Delta t = 0.004$. We can see this stability from the von Neumann analysis above, since $\Delta t < \frac{\Delta x^2}{2D} = \frac{0.1^2}{2} = 0.005$, which is true.

In our second plot, we have $\Delta t = 0.0053$ and resultant unstable solution since how choice of Δt exceeds the maximum stability above obtained above.

This shows that controlling the error magnification for practical step sizes is a crucial aspect of obtaining an efficient solution.


u
0.8
0.6
0.4
0.2
0
t
0.1
0
0.2
0.4
0.6
0.8
x
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0



4.4.3 Lax Equivalence Theorem

Theorem 4.1 (Lax Equivalence Theorem)**.** *If we have a consistent approximation to a well-posed linear initial-value problem, then the approximation is convergent if and only if it is stable.*

4.4.3.1 Tutorial

1. Prove that the functions (a) $u(x,t) = e^{2t+x} + e^{2t-x}$, (b) $u(x,t) = e^{2t+x}$ are solutions of the heat equation $u_t = 2u_{xx}$ with the specified boundary conditions:

$$\begin{cases} u(x,0) = e^x \text{ for } 0 \le x \le 1 \\ u(0,t) = e^{2t} \text{ for } 0 \le t \le 1 \\ u(1,t) = e^{2t+1} \text{ for } 0 \le t \le 1 \end{cases}$$

2. Prove that if $f(x)$ is a degree 3 polynomial, then $u(x,t) = f(x) + ctf''(x)$ is a solution of the initial value problem $u_t = cu_{xx}$, $u(x,0) = f(x)$.
3. True or False: For solving a time-dependent partial differential equation, a finite difference method that is both consistent and stable converges to the true solution as the step sizes in time and space go to zero.

4. Use step sizes of (a) $\Delta x = 0.1$ and $\Delta t = 0.0005$ and (b) $\Delta x = 0.1$ and $\Delta t = 0.01$ to approximate the solution to the heat equation:

$$u_t = u_{xx},\ 0 < x < 1,\ t \geq 0,$$

with boundary conditions:

$$u(0,t) = u(1,t) = 0,\ t > 0,$$

and initial conditions:

$$u(x,0) = \sin(\pi x), \quad 0 \leq x \leq 1.$$

Compare the results at $t = 0.5$ to the exact solution:

$$u(x,t) = e^{-\pi^2 t}\sin(\pi x).$$

5. Approximate the solution to the following partial differential equation using the backward difference method:

$$u_t = u_{xx},\ 0 < x < 2,\ t \geq 0,$$

$$u(0,t) = u(2,t) = 0,\ t > 0,$$

$$u(x,0) = \sin(\frac{\pi}{2}x), \quad 0 \leq x \leq 2.$$

Use $m = 4, T = 0.1$, and $N = 2$ and compare your results to the actual solution $u(x,t) = e^{-(\pi/4)t}\sin\frac{\pi}{2}x$.

6. Approximate the solution to the following partial differential equation using the backward difference method:

$$u_t = \frac{1}{16}u_{xx},\ 0 < x < 1,\ t \geq 0,$$

$$u(0,t) = u(1,t) = 0,\ t > 0,$$

$$u(x,0) = 2\sin(2\pi x), \quad 0 \leq x \leq 1.$$

Use $m = 3, T = 0.1$, and $N = 2$ and compare your results to the actual solution $u(x,t) = 2e^{-(\pi^2/4)t}\sin 2x$.

7. Using the code you have written in Lab 4, apply your functions on the PDEs given in questions 5 and 6. Explore various step sizes. Compare your results against the actual solutions.

References

Bibliography

Fornberg, B. (1998). Classroom note: Calculation of weights in finite difference formulas. *SIAM review*, 40(3):685–691.