IMPERIAL

Computational Partial Differential Equations (CPDEs) MATH60025/70025

Spring Term 2: 2024-2025

live document

(updated January 8, 2025)

MATH60025/70025: Computational Partial Differential Equations

Lecturer: Dr Shahid Mughal

The lecture notes are an adaptation of earlier notes (2018) on the course developed by Professor J. Mestel, Mathematics Department, Imperial College London.

Email : s.mughal@imperial.ac.uk
Office : Department of Mathematics, Huxley 734
Assessment : 100% by projects
Lectures : 26 + Surgery hours

Module Summary

This module will introduce a variety of computational approaches for solving partial differential equations, focusing mostly on finite difference methods. Students will gain experience implementing the methods and writing/modifying short programs in Matlab or another programming language of their choice. Applications will be drawn from problems arising in areas such as Mathematical Biology and Fluid Dynamics.

Prerequisites : None, other than the core modules from years 1 & 2, but Programming ability in Python is required – MATH50008 module *Partial Differential Equations in Action* will provide some background. You will be expected to produce and/or modify programs in (say) Python or any other language of your choice.

Module Content :
There is some flexibility as to the exact module content and ordering, but the following should be very close:

1. Introduction: How can we solve PDEs on a computer? Finite Difference Methods. Basic types and Classification of PDEs. Well-posedness and the importance of Boundary Conditions.
2. Parabolic equations: Explicit & Implicit Schemes. Maximum principle analysis.
3. Elliptic equations: Iterative methods: How can they be made faster? Jacobi, Gauss-Seidel, relaxation techniques. Multigrid methods and motivation and implementation.
4. Hyperbolic equations: characteristics, up-winding, Lex-Wendroff schemes. Non-reflecting boundary conditions, perfectly matched layers (PML).
5. Combinations, Extensions and Applications: e.g. advection/diffusion and Navier-Stokes equations. Magnetic Induction and heat transport equations. Domain decomposition, operator splitting.

Intended Learning Outcomes :

- Appreciate the physical and mathematical differences between different types of PDEs;
- Outline a theoretical approach to testing the stability of a given algorithm;
- Determine the order of convergence of a given algorithm;
- Demonstrate familiarity with the implementation and rationale of multigrid methods;
- Develop finite-difference based software for use on research level problems;
- Use numerical methods, as an alternative to analytical approaches, to solve mathematical problems with a physical underpinning.

Programming and Coding Proficiency : This course does **NOT** assess your programming/coding abilities. It is expected that you will already be proficient in a programming language like Python, Fortran, C or Matlab. If you have **NO experience in programming you should think wisely as to whether this course is for you** – it is possible to learn programming as you go along, but you may find this quite challenging particularly due to time constraints of submitting your assessed work.

Philosophy of the Module : Most of the universe is governed by Partial Differential Equations (PDEs), but the techniques for solving PDEs analytically are few. Yet nowadays we have incredible computer power, and we can obtain accurate approximations to solutions to most physically relevant problems. An understanding of analytical techniques really should be accompanied by the ability to find numerical solutions when required. This module is important for those considering a future in research, but also for those considering getting a job in a scientific & technical fields (including Finance) – these almost invariably involve solving some form of PDE or sets of PDEs with a computer.

It is moderately easy to write inefficient code which solves simple problems adequately. This skill should not be under-rated - many problems you will encounter can be dealt with effectively by a "quick and dirty" approach. Yet, if you want to be able to solve important problems quickly, or difficult problems at all, a good understanding of numerical techniques is vital, and is a well-valued talent. This is what this module aims to engender. At its conclusion, you should be able to make a decent stab at previously unsolved Research-level problems.

Assessment : The module will be assessed solely by projects. The plan is as follows: A relatively short project, worth 25%will be set early on, and returned to you with comments. You will be committed to completing the module on submission of 25%. A final project will be set in week 8, with an additional project released for the Mastery (MSc, MSci). These projects should be submitted electronically, via Blackboard.

1. **CW 1 :** 25% released 28 January, submission 1pm, 10 February.
2. **CW 2 :** 40% released 14 February, submission 1pm, 3 March.
3. **CW 3 :** 35% released 6 March, submission 1pm, 20 March.
4. **CW 4 :** 20% released 6 March, submission 1pm, 4 April **(Mastery)**.

The assessment for each CW is based upon you solving a given problem having written your code, and followed up by a **detailed technical report** which outlines your findings. You should not expect a high mark, if you only submit your code but fail to submit the written report.

During the module, various Matlab codes will be demonstrated in lectures and released on Blackboard. Most of you will choose to modify these codes for the problems set in the projects, but it is quite permissible to write your programs in any language which runs on the Imperial College machines, e.g. Python, C, Matlab, Fortran.

The projects will involve demonstrating that your codes work to the expected accuracy, obtaining solutions to set problems and discussing the results. The projects may build on one another slightly; you may be able to use your codes from the first two projects as part of the final project. The final project will be at a high level, the kind of problem which borders on Research level. At the end of the module you should be able to solve difficult problems using the various techniques covered.

More topics will be covered in lectures that will/may be of direct of use in the Projects, just as some topics do not get assessed in examinations.

General Comments on Project Modules :
Most Maths modules are assessed mainly by a summer exam. Some memory is required - in 2-3 hours you demonstrate what you have learned from the lectures and the many hours of revision. In project modules, memory is not a factor, and you can spend a very long time on them if you choose. As a result, average marks on project modules tend to be a little higher. Nevertheless, 100% is not achievable without deep understanding, and you should not expect perfection. In the past, some students have spent too long on their projects and neglected revision of other topics – I can only caution you against this. The Senior Tutor may advise you not to attend too many project modules.

Collaboration :
We do not, of course, wish to discourage discussion between you on any of the mathematical and computational issues underlying this module. However, plagiarism considerations are very important in project modules. The projects really must be produced independently and any help you received MUST be acknowledged in your submissions. You must adhere strictly to the plagiarism guidelines you have been given – if you are in any doubt about what is permitted you should seek clarification from the Senior Tutor, Dr Ford. The College penalties are exceptionally severe for breaches and this can jeopardise your entire degree. Please do be sensible about this.

Support Classes :
Every now and again, problem sessions will occur in lectures. There will also be surgery sessions in office hours. The timings of these will be discussed in lectures.

Recommended Books :

See the complete Reading list on Blackboard, but the following are pretty good.

1. LeVeque, Randall J., *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. **(Core)**
2. Tannehill, John C., *Computational fluid mechanics and heat transfer*.
3. Smith, G. D., *Numerical solution of partial differential equations : finite difference methods*.
4. LeVeque, Randall J., *Finite Volume Methods for Hyperbolic Problems*.
5. Toro, Eleuterio F., *Riemann Solvers and Numerical Methods for Fluid Dynamics*, 3rd edition.

MATH60025/70025: Computational Partial Differential Equations

Contents

1 Brief Motivation

The entire universe and phenomenon we come across on a daily basis ranging from physical, chemical, biological, mechanical, economical to weather forecasting can be described by some form of systems of Partial Differential Equations (PDEs). Only a limited set of PDEs can be solved exactly. For example, Laplace equation widely arises in physics problems ranging from electrical, magnetic and gravitational potentials, steady-state temperatures, and in hydrodynamics (to name a few). You may recall from earlier courses how to solve the Laplace equation for $u(x, y)$ in a rectangle,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \in [\, 0 < x < a, \ \ 0 < y < b \,], \tag{1.1}$$

with the boundary conditions

$$u(a, y) = 1 \ \text{ for } \ y \neq (0, b); \quad u(0, y) = u(x, 0) = u(x, b) = 0. \tag{1.2}$$

Using separation of variables and Fourier series, you can show that

$$u = \frac{2}{\pi} \sum_{n=1}^{\infty} \frac{1 - (-1)^n}{n \sinh(n\pi a/b)} \sinh\left(\frac{n\pi x}{b}\right) \sin\left(\frac{n\pi y}{b}\right). \tag{1.3}$$

That's fine, but what does the solution look like? You'll almost certainly want to contour it on a computer. To do this you have to truncate the series to a finite number of terms, and so only plot an approximation to the exact solution, as shown in Figs. 1.1 and 1.2. How many terms in the series are adequate to obtain a desired accuracy? Why is this better than obtaining an approximation to the entire problem on a computer from the start?

The above equation (1.1) may be solvable exactly on square, rectangular or circular boundaries as shown in Fig. 1.3a,b, however if the boundaries or boundary conditions become complex (Fig. 1.3c,d) either the exact analytic solution is not possible (in most cases) or some considerable work has to be done to find the exact analytical solution – which even then may well require some form of numerical computation for the solution to be make sense.

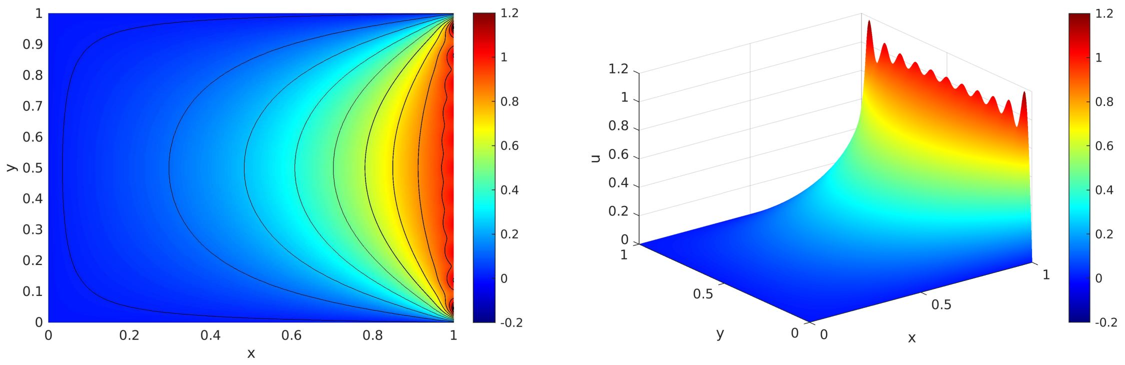

Figure 1.1: Typical solution of Eqn. 1.1 – two-dimensional plot, on truncation of Eqn. 1.3 to a finite number ($n = 21$) of terms.

Numerical solutions can be found for a much wider variety of problems, including systems of **nonlinear** PDEs. Numerical computation is commonplace today in virtually every aspect

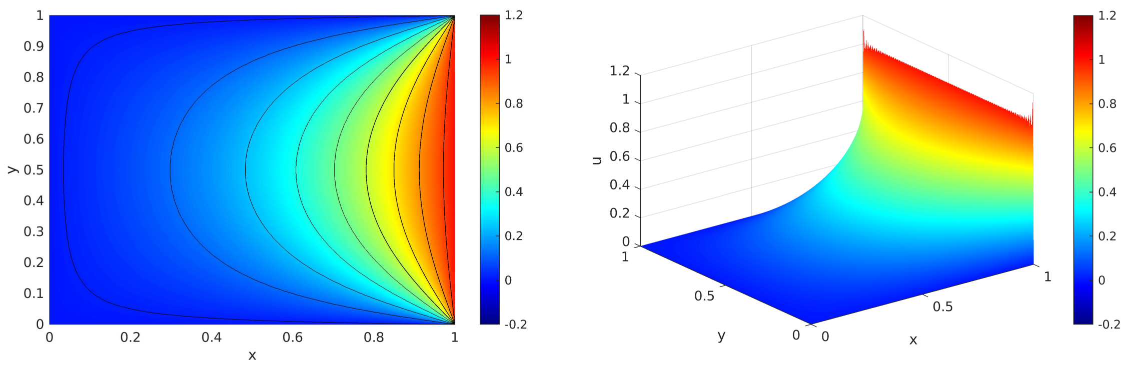

Figure 1.2: Solution of Eqn. 1.1 – surface plot, on truncation of Eqn. 1.3 to a larger but still finite $(n = 161)$ terms.

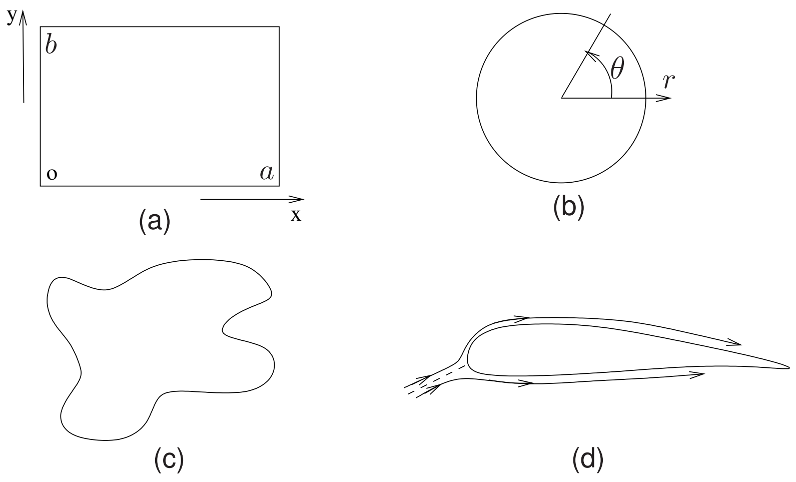

Figure 1.3: Examples of Boundary Value problems (BVPs).

of the world we live in, from engineering, finance, astronomy, physics, artificial intelligence, data analysis, weather prediction and modelling spread of infections - the list is endless. Computers have made possible solutions of scientific and engineering problems of great complexity.

This module aims to take you to the point where you will have the background knowledge to solve a range of PDEs with the aid of a computer. The course objective is to give some of the mathematical and numerical background which allows you to assess and wisely choose the most appropriate numerical technique for the PDE that you may be interested in solving. As you will see, in most cases the numerical technique devised, has to honour the mathematical and physical character of the PDE. There is no single numerical technique which will solve all PDEs that you may come across, rather the skill of a practitioner in the field is to have the necessary background in mathematical and numerical analysis to develop the technique which works best (or optimally) for the problem at hand – in terms of accuracy of the solution to the PDE which satisfies all applied boundary-conditions imposed on the PDE.

2 PDEs - Primer and Definitions

A partial differential equation (PDE) is a functional relation between an unknown function, say u, and its derivatives. We consider systems where there is some sort of interaction between independent variables, which we denote vectorially as $\mathbf{x} = (x, y, z, t)$ say, and the dependent variable u; or a set of dependent variables $\mathbf{u} = (u, v, w, \ldots)$. Here the x, y, z independent variables play the role of spatial variables in a three-dimensional (3D) world, say, and in time-dependent cases we reserve the variable t to play the role of time.

A partial derivative of u with respect to the x independent variable is denoted by

$$\frac{\partial u}{\partial x}, \quad \text{or} \quad u_x.$$

Similarly a partial derivative to second-order is denoted by

$$\frac{\partial^2 u}{\partial x^2}, \quad \text{or} \quad u_{xx}.$$

Hence a *mixed*-derivative u_{xy} implies

$$\frac{\partial^2 u}{\partial x \partial y} \equiv u_{xy}, \quad \text{and} \quad \frac{\partial^3 u}{\partial x^2 \partial y} \equiv u_{xxy} \quad \text{etc.}$$

Order of a PDE

One way to classify PDEs is according to the **order** of the equation. The order is defined to be the order of the highest derivative appearing in the equation. For example

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = f(x, t),$$

is a **second-order** equation, while

$$\frac{\partial u}{\partial t} + \frac{\partial^4 u}{\partial x^4} = 0$$

is a **fourth-order** equation.

More formally, a PDE is a functional relation of the form

$$\mathcal{F}(\mathbf{x}, u, Du, D^2u, \ldots, D^m u) = 0, \ \mathbf{x} \in \Omega \subset \mathbb{R}^n,$$

where the unknown is the function $u : \Omega \subset \mathbb{R}^n \mapsto \mathbb{R}$ and

$$D^m u = \left. \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \ldots \partial x_n^{\alpha_n}} \right|, \quad \alpha \in \mathbb{N}^n, |\alpha| = m$$

is the set of all derivatives of order m.

Linearity and Nonlinearity of a PDE

PDEs are classified in two groups: **linear** PDEs and **nonlinear** PDEs. For example, the equation

$$\mathcal{F} = x^7 \frac{\partial u}{\partial x} + e^{xy} \frac{\partial u}{\partial y} + \sin^2(x^2 + y^2) u - x^3 = 0$$

is a *first-order* linear equation. The heat or diffusion equation is a *second-order* linear PDE

$$\mathcal{F} = \frac{\partial u}{\partial t} - K \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) = 0,$$

with K a constant parameter representing either diffusion or heat conductivity. Other examples are

- **Schrodinger equation** in physics (second order, linear)

$$-\frac{h^2}{2m} \nabla^2 u + V(\mathbf{r}) u = ih \frac{\partial u}{\partial t}.$$

- **Black-Scholes equation** in finance (second order, linear)

$$\frac{\partial u}{\partial t} + \frac{\sigma^2 x^2}{2} \frac{\partial^2 u}{\partial x^2} + rx \frac{\partial u}{\partial x} - ru = 0.$$

A simple example of a nonlinear PDE is

$$\mathcal{F} = \left(\frac{\partial^2 u}{\partial t^2} \right)^2 + \left(\frac{\partial^2 u}{\partial x^2} \right)^2 = 0.$$

We further classify nonlinear equations in sub-classes according to the type of nonlinearity. The nonlinearity of an equation is more pronounced when it appears on a higher derivative. We commonly call PDEs :

1. **Semi-linear** – where $\mathcal{F}$ is nonlinear only with respect to u but is linear with respect to all its derivatives, for example

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = u^3.$$

Another example is the 2D *Fisher-Kolmogorov-Petrovsky-Piskunov's* second-order reaction-diffusion type equation

$$\frac{\partial u}{\partial t} - K \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = ru(M - u).$$

or the slightly more complicated *Fokker-Plank* equation[*] in 1-D for $u(x,t)$

$$\frac{\partial u}{\partial t} = -\frac{\partial}{\partial x}(b(x)\ u) + \frac{\sigma^2(x)}{2} \frac{\partial^2 u}{\partial x^2}.$$

[*]an interesting feature here is the competing effects of diffusion or *stochastic diffusion* given by the σ^2 parameter and so-called drift or "*stochastic drift*" arising from the $b(x)$ term and their relative magnitudes – termed convection-diffusion in fluid dynamics.

Fokker-Plank type equations often arise in modelling stochastic phenomenon in ecology, genetics, circuit theory, solid-state and chemical physics, finance and fusion modelling among others.

More generally, suppose a PDE has the form

$$\sum_{|\alpha|=k} \big(a_\alpha(\mathbf{x}) D^\alpha u(\mathbf{x}) + a_o(D^{k-1}u(\mathbf{x}), \ldots, u(\mathbf{x}), \mathbf{x})\big) = 0,$$

the PDE is semi-linear if the coefficient $a_\alpha(\mathbf{x})$ depends on $\mathbf{x}$ only, while a_o can comprise nonlinear combinations of derivatives of u in lower-order.

2. **Quasilinear** – where $\mathcal{F}$ is linear with respect to the highest order derivatives of u only, for example

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = |\nabla u|^2 u;$$

or the *first-order* inviscid Burger's equation

$$\frac{\partial u}{\partial t} + cu\frac{\partial u}{\partial x} = 0 \text{ with } c \text{ a constant.}$$

More generally, the PDE

$$\sum_{|\alpha|=k} \big(a_\alpha(D^{k-1}u(\mathbf{x}), \ldots, u(\mathbf{x}), \mathbf{x}) D^\alpha u(\mathbf{x}) + a_o(D^{k-1}u(\mathbf{x}), \ldots, u(\mathbf{x}), \mathbf{x})\big) = 0.$$

is quasi-linear if the coefficients a_α, a_o depend on lower-order derivatives of $u(\mathbf{x})$.

3. **Fully nonlinear** – where $\mathcal{F}$ is nonlinear with respect to the highest order derivatives of u, for example

$$\mathcal{F} = \left(\frac{\partial^2 u}{\partial x^2}\right)^2 + \left(\frac{\partial^2 u}{\partial x^2}\right)^2 = 0,$$

or the Monge-Ampere (MA) equation (in 2D)

$$\mathcal{F} = \frac{\partial^2 u}{\partial x^2}\frac{\partial^2 u}{\partial y^2} - \left(\frac{\partial^2 u}{\partial x \partial y}\right)^2 - f(x, y).$$

The MA equation is a fully nonlinear PDE with applications in physics and engineering ranging from meteorology, elasticity, geometric optics, image processing and others.

3 Finite Difference Methods (FDMs)

How might one solve a PDE numerically? We can only calculate a finite number of **discrete** values, so we begin by defining a **grid of points** on which we will seek the solution. For now, consider a one-dimensional grid, and seek to approximate the function $u(x)$ on (a, b). We shall consider a uniform grid almost always, so we define a step-length,

$$h = (b-a)/(N-1),$$

for some large N, as shown in Fig. (3.1). We consider the points

$$x_n \equiv a + (n-1)h \text{ for } n = 1 \dots N.$$

We shall seek an approximation U_n to the exact solution on the grid points, $u_n \equiv u(x_n)$.

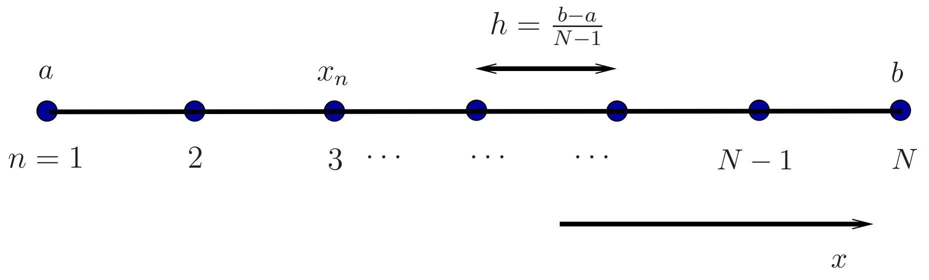

Figure 3.1: Discretised Grid.

Using suitable finite difference **(F-D)** approximations, we can represent any system of ODEs or PDEs in continuous variables as a set of algebraic equations for a finite list of unknown (*discrete*) values. We can then seek to solve this algebraic system on a computer. If $u(x)$ is a solution to a (partial) differential equation we need to represent the derivatives, $\partial u/\partial x \equiv u_x$, $\partial^2 u/\partial x^2 \equiv u_{xx}$, $\partial^2 u/\partial x \partial y \equiv u_{xy}$, $\dots$, etc. with finite difference approximations. The F-D method replaces continuous derivatives in the governing equations by finite-difference approximations. The underlying task then is to approximate the derivative $\partial u/\partial x \equiv u_x = u'(x)$ of the continuous function $u(x)$ evaluated at a specified grid point x_n by a linear combination of **discrete** function values U_n. For the case of a first derivative this may lead to

$$\frac{\partial u}{\partial x}\Big|_n \approx \frac{u(x_n+h) - u(x_n)}{h} = \frac{U_{n+1} - U_n}{h}, \tag{3.1}$$

or we may equally well approximate the derivative as follows:

$$\frac{\partial u}{\partial x}\Big|_n \approx \frac{u(x_n) - u(x_n-h)}{h} = \frac{U_n - U_{n-1}}{h}, \tag{3.2}$$

a third choice, namely centred about x_n is given by

$$\frac{\partial u}{\partial x}\Big|_n \approx \frac{u(x_n+h) - u(x_n-h)}{2h} = \frac{U_{n+1} - U_{n-1}}{2h}. \tag{3.3}$$

Whats the difference between these 3 expressions? Can one arbitrarily pick one at random and expect the results to be correct? We will see later on in the course, with especially PDEs, that one or the other may or may not be appropriate. Can you think why?

In general we have to specify the number and location of discrete function values U_n which are used in the approximation, as well as the location at which the derivative is to be evaluated. For example, taking into account three function values $U_{n-1},\ U_n,\ U_{n+1}$ and evaluating the first derivative at the central point x_n (see Fig. 3.2), we obtain the general setup[*]

$$\frac{du}{dx}\Big|_n \approx aU_{n-1} + bU_n + cU_{n+1} \tag{3.4}$$

with some yet unknown coefficients $a,\ b,\ c$. Similar setups for higher derivatives are imaginable. Once the unknown coefficients or **weights** are determined, the derivatives in the governing equations can then be replaced by linear combinations of the function values.

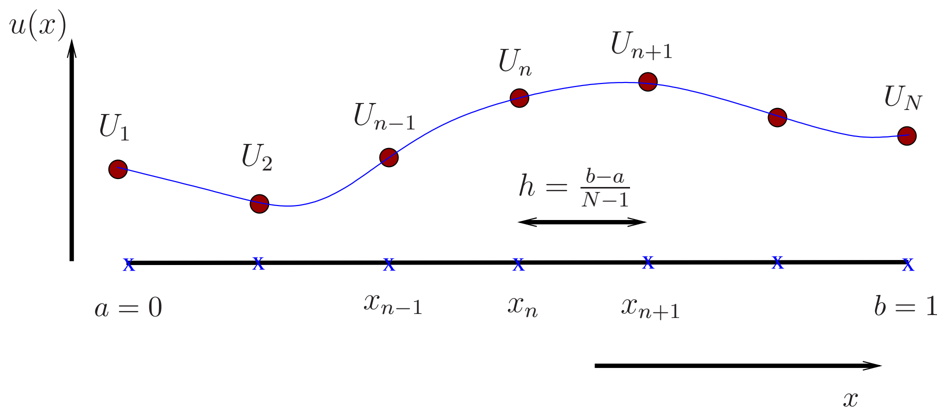

Figure 3.2: Sketch of a equi-spaced grid and a discrete representation of a continuous function $u(x)$ on this grid,

Even if our PDE is **nonlinear**, we almost always arrange things so that the system of equations we have to solve is **linear.** However there are usually a very large number of such equations, and we may have to give thought to the best way of solving them efficiently.

Depending on the method we use, they may be **explicit** or **implicit**. An explicit equation is of the form “unknown = easily calculable stuff", but an implicit system still requires further work, perhaps an iterative solution.

3.1 Finite-difference formulae via Taylor series expansions

To determine the weight coefficients $a,\ b$ and c in expression (3.4) for the first derivative, we use a Taylor series expansion of the function $u(x)$ about the point x_n where the derivative is sought. Using Taylor series we can write

$$u_{n+1} \equiv u(x_n + h) = u(x_n) + hu'(x_n) + \tfrac{1}{2}h^2u''(x_n) + \tfrac{1}{6}h^3u'''(x_n) + \tfrac{1}{24}h^4u^{iv}(x_n) + \ldots \tag{3.5}$$

$$u_{n-1} \equiv u(x_n - h) = u(x_n) - hu'(x_n) + \tfrac{1}{2}h^2u''(x_n) - \tfrac{1}{6}h^3u'''(x_n) + \tfrac{1}{24}h^4u^{iv}(x_n) + \ldots \tag{3.6}$$

It follows by addition and subtraction that

$$\textbf{subtraction :}\quad u_{n+1} - u_{n-1} = 2hu'(x_n) + \tfrac{1}{3}h^3u'''(x_n) + O(h^5)^{\dagger}, \tag{3.7}$$

[*]note in what follows we have changed our notation from using the ∂ symbol to d here.
[†]referred to as the *order*(h^5) truncated term

and

$$\textbf{addition :}\quad u_{n+1}+u_{n-1}=2u(x_n)+h^2u''(x_n)+\tfrac{1}{12}h^4u^{iv}(x_n)+O(h^6). \tag{3.8}$$

We can therefore approximate the *first*-derivatives as follows

$$\frac{du}{dx}|_n=\frac{u_{n+1}-u_{n-1}}{2h}+O(h^2)\equiv\frac{(\Delta+\nabla)u_n}{2h}+O(h^2), \tag{3.9}$$

with the weights $a=-1/h, b=0, c=1/h$. The *second*-derivative FD form is given by

$$\frac{d^2u}{dx^2}|_n=\frac{u_{n-1}-2u_n+u_{n+1}}{h^2}+O(h^2)\equiv\frac{\delta^2u_n}{h^2}+O(h^2), \tag{3.10}$$

with the weights $a=1/h^2, b=-2/h^2, c=1/h^2$ – note usage above of *difference operator* notation too (see §3.2 below). Equations (3.10) and (3.9) are known as finite difference approximations to u_{xx} and u_x. They are **second order** as the error is $O(h^2)$. They are called **centred** which basically means that the approximation is symmetrical about n.

3.2 Difference operators

The difference operators Δ and δ^2 operate on n, just as d/dx operates on x. They may be used symbolically for non-integer n, if we want. So we define the following operators:

Forward difference : $\Delta x_n = x_{n+1}-x_n$

Backward difference : $\nabla x_n = x_n - x_{n-1}$

Shift operator : $Ex_n = x_{n+1}$

Averaging operator : $\mu x_n = \frac{1}{2}(x_{n+1/2}+x_{n-1/2})$

Central difference : 1^{st}-derivative : $\delta u_n = u_{n+1/2}-u_{n-1/2}$

Central difference : 2^{nd}-derivative : $\delta^2 u_n = u_{n-1}-2u_n+u_{n+1}$

Differential operator : $\mathcal{D}y = dy/dx$

As an example (3.9) may also be written

$$\frac{du}{dx}|_n=\frac{u_{n+1}-u_{n-1}}{2h}=\frac{1}{h}\mu\delta u_n. \tag{3.11}$$

Further note the 1^{st}-order accurate forward difference may be written

$$\frac{du}{dx}|_n=\frac{u_{n+1}-u_n}{h}=\frac{1}{h}(E-1)u_n. \tag{3.12}$$

To derive finite difference using the above operators, $\mathcal{D}$ needs to be defined in terms of the other operators. We proceed with the Taylor series

$$f(x+h)=f(x)+h\frac{df}{dx}+\frac{h^2}{2!}\frac{d^2}{dx^2}+\frac{h^3}{3!}\frac{d^3}{dx^3}+\frac{h^4}{4!}\frac{d^4}{dx^4}+\dots\ ,$$

written in operator form

$$f(x+h) = Ef(x) = \left[1 + h\mathfrak{D} + \frac{(h\mathfrak{D})^2}{2!} + \frac{(h\mathfrak{D})^3}{3!} + \dots\right] f(x) = e^{h\mathfrak{D}}\, f(x).$$

We see that

$$E = e^{h\mathfrak{D}},$$

on basis that equality of the operators, whether using E or $e^{h\mathfrak{D}}$ give identical results when operating on a polynomial. Hence it is easy to establish the following relationships

$$\begin{aligned} h\mathfrak{D} &= \log E = \log(1+\Delta) = -\log(1-\nabla) \\ &= 2\sinh^{-1}\delta/2 = 2\log[(1+\tfrac{1}{4}\delta^2)^{1/2} + \tfrac{1}{2}\delta]. \end{aligned} \qquad (3.13)$$

Hence the FD expressions follow by expansion and further manipulation of these expressions. For example

$$\log(1+\Delta)) \approx \Delta - \frac{\Delta^2}{2} + \frac{\Delta^3}{3} + \dots,$$

truncation of this to first order then gives (3.12)[‡].

3.3 Truncation error

Above, the coefficients were obtained by requiring a match between the Taylor-expanded right-hand side and the derivative term on the left-hand side. By our design, only three constants $a,\ b,\ c$ were available to make this match. As a consequence, an error term appears when higher-order derivative terms on the right-hand side do not cancel or are neglected. The lowest of the non-cancelling terms is referred to as the **truncation error** e. In our case above (in 3.7) we have

$$e = (c-a)\frac{u'''|_n}{3!}h^3 = \frac{u'''|_n}{3}h^2 \sim O(h^2). \qquad (3.14)$$

For h sufficiently small, the error will be dominated by u''' or the so-called *order*-(h^2) term, with all other even higher-order derivative contributions being negligible *relative* to this term. As the grid is refined, the truncation error decreases quadratically with the mesh width h. Representations of the first derivative to higher than second order require more u_n $(\equiv U_n)$ function values such that more coefficients can be used to eliminate even higher-order derivatives on the right-hand side, thus enabling the truncation error to be reduced further.

[‡]see W. F. Ames, 1977 for further details

3.4 Higher order and sided-differences

For improved accuracy, to be more precise, to reduce the truncation errors, higher-order difference formulae can be deduced using the basic formalism of Taylor series. Doing so, leads to using more discrete functional values on either side of the n^{th}-*centered* point about which derivatives are to be discretised. Note that these higher-order FDs may need modification near boundaries. When derivatives have to be evaluated on or near the boundary of the computational domain, but function values are only available on one side or a *centered-discretised* stencil is not possible, approximations to derivatives of functions there are known as **sided formulae**. For example, with reference to Fig. 3.2, at $n = 1$ we have no information or knowledge of the u-state at points $n < 1$ or $x < a$, or for that matter $n > N$ or $x > b$.

3.4.1 4^{th}-Order or $O(h^4)$ accurate finite-differences

For this order of accuracy, a 5-point scheme is required with the central differenced weights given as follows :

$$\frac{du}{dx}|_n = \frac{u_{n-2} - 8u_{n-1} + 8u_{n+1} - u_{n+2}}{12h},$$

$$\frac{d^2u}{dx^2}|_n = \frac{-u_{n-2} + 16u_{n-1} - 30u_n + 16u_{n+1} - u_{n+2}}{12h^2},$$

4^{th}-Order or $O(h^4)$ accurate sided differences, near boundaries

Left-sided non-centered boundary weights :

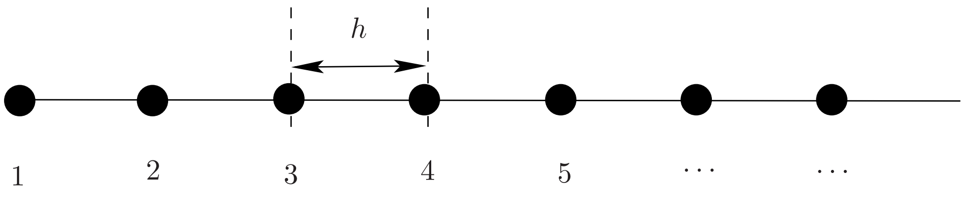

Figure 3.3: Sided differencing stencil, left-sided boundary points.

$$\frac{du}{dx}|_2 = \frac{-3u_1 - 10u_2 + 18u_3 - 6u_4 + u_5}{12h},$$

$$\frac{d^2u}{dx^2}|_2 = \frac{11u_1 - 20u_2 + 6u_3 + 4u_4 - u_5}{12h^2},$$

$$\frac{du}{dx}|_1 = \frac{-25u_1 + 48u_2 - 36u_3 + 16u_4 - 3u_5}{12h},$$

$$\frac{d^2u}{dx^2}|_1 = \frac{35u_1 - 104u_2 + 114u_3 - 56u_4 + 11u_5}{12h^2},$$

Right-sided non-centered boundary weights :

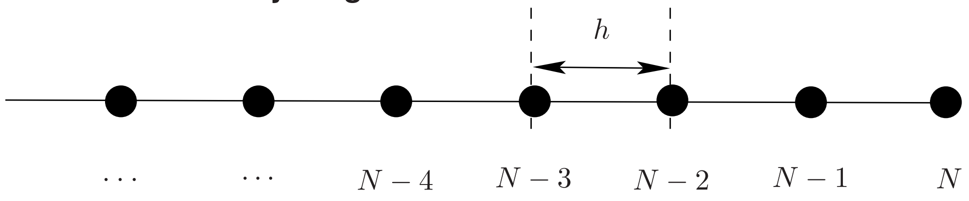

Figure 3.4: Sided differencing stencil, right-sided boundary points.

$$\frac{du}{dx}|_{N-1} = \frac{3u_N + 10u_{N-1} - 18u_{N-2} + 6u_{N-3} - u_{N-4}}{12h},$$

$$\frac{d^2u}{dx^2}|_{N-1} = \frac{11u_N - 20u_{N-1} + 6u_{N-2} + 4u_{N-3} - u_{N-4}}{12h^2},$$

$$\frac{du}{dx}|_{N} = \frac{25u_N - 48u_{N-1} + 36u_{N-2} - 16u_{N-3} + 3u_{N-4}}{12h},$$

$$\frac{d^2u}{dx^2}|_{N} = \frac{35u_N - 104u_{N-1} + 114u_{N-2} - 56u_{N-3} + 11u_{N-4}}{12h^2},$$

Further details on this and other ways of computing weights may be found in the reading list (LeVeque, 2007). To summarise the weights can be determined using:

- Taylor series with method of undetermined coefficients;
- Difference operators described in §3.2 above;
- Lagrange or Polynomials interpolation.

3.5 Finite-difference formulae via Lagrange interpolation

An alternative and more flexible method to develop finite-difference approximations to continuous derivatives is based on Lagrange interpolation. The idea is to reconstruct a local, approximate, but continuous representation of the function $u(x)$ by interpolating the discrete function values u_n using a polynomial of appropriate degree. This polynomial interpolant is then differentiated exactly and evaluated at the point of interest. The concept thus relies on a three-step procedure (see LeVeque, 2007):

- interpolation of the data points by a polynomial of appropriate degree,
- exact differentiation of the polynomial interpolant, and
- evaluation of the differentiated interpolant at the desired grid point.

A very robust and efficient procedure to evaluate weights to any order of accuracy is given by Fornberg (1996) (see Reading list).

3.6 Our first finite difference method

For example, we could seek to solve the ordinary differential equation (ODE) :

$$u' \equiv \frac{du}{dx} = -u \text{ in } x > 0 \text{ with } u(0) = 1. \tag{3.15}$$

Choosing a step-length and grid as shown in Fig. 3.2, we could represent the ODE fairly accurately as

$$\frac{U_{n+1} - U_{n-1}}{2h} = -U_n, \qquad \text{with} \quad U_1 = 1. \tag{3.16}$$

This is a linear recurrence relation. If we know U_1 and U_2, by varying n we can find U_n for all n. Obviously the exact solution to this problem is $u = e^{-x}$. So let's take $U_2 = e^{-h}$.

$$U_{n+1} = -2hU_n + U_{n-1} \qquad \text{for } n \geq 1, \text{ with} \quad U_1 = 1, \quad U_2 = e^{-h}. \tag{3.17}$$

More formally this is known as an **explicit** method, whereby we simply march forward in the variable x_{n+1} and values of the solution there U_{n+1} simply involves previously evaluated values of the function U at $n, \; n-1,$ or all $n < n+1$.

Try it and see what happens. Does $|U_n - u_n|$ remain small as n increases? Does the approximation improve as h decreases?

It is very important to realise that even if our equations are a good approximation to the ODEs, the solutions obtained may be completely different, for various reasons.

How well does U approximate the true exact solution $u(x) = e^{-x}$? In most problems of practical interest the true exact solution is usually not known. How do we ascertain that our discretised numerical solution is a true solution of the exact governing equation, or that as h the step size is decreased the errors reduce and the solution does indeed converge to the true solution? A very good introduction on aspects of **Truncation errors**, **Global error**, **Numerical Stability**, **Consistency** and **Convergence** are given in LeVeque (2007) (see sections 2.4 – 2.10).

PDEs are trickier than ODEs, and are quite sensitive to their boundary conditions. Before we can hope to model them well, we need to have some idea of the possible underlying behaviour. So next time, we'll consider the basic types of PDE. We will then consider solving each type in turn.

Typical results of our first Finite Difference Method, Eqn. (3.16)

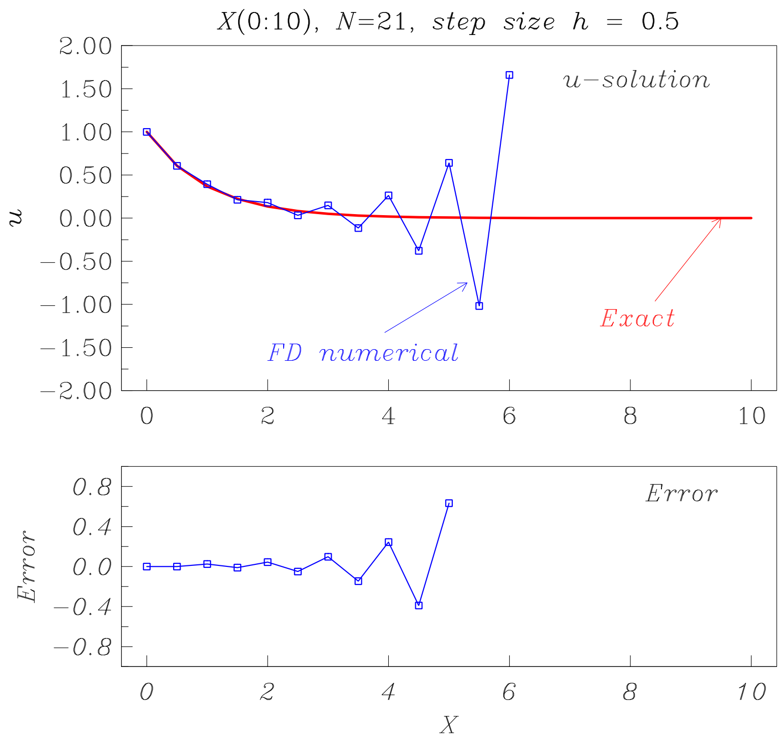

Figure 3.5: Solution of $u' = -u$ with $u(0) = 1$, N=21 grid points.

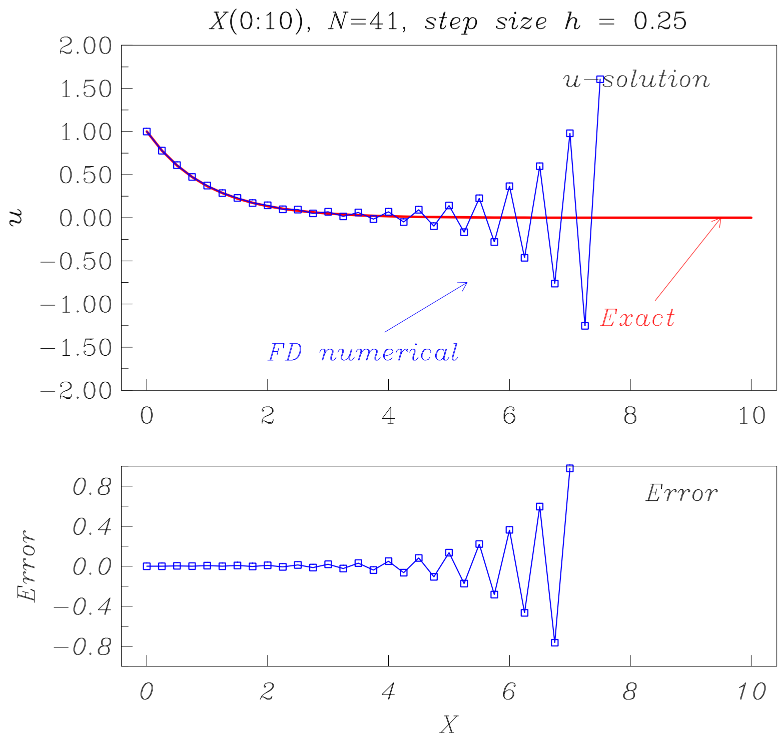

Figure 3.6: Solution of $u' = -u$ with $u(0) = 1$, N=41 grid points.

Typical results of improved Finite Difference Method

Discretisation:
$$\frac{U_{n+1}-U_{n-1}}{2h} = -\frac{U_{n+1}+U_{n-1}}{2}, \qquad \text{with} \quad U_1 = 1. \tag{3.18}$$

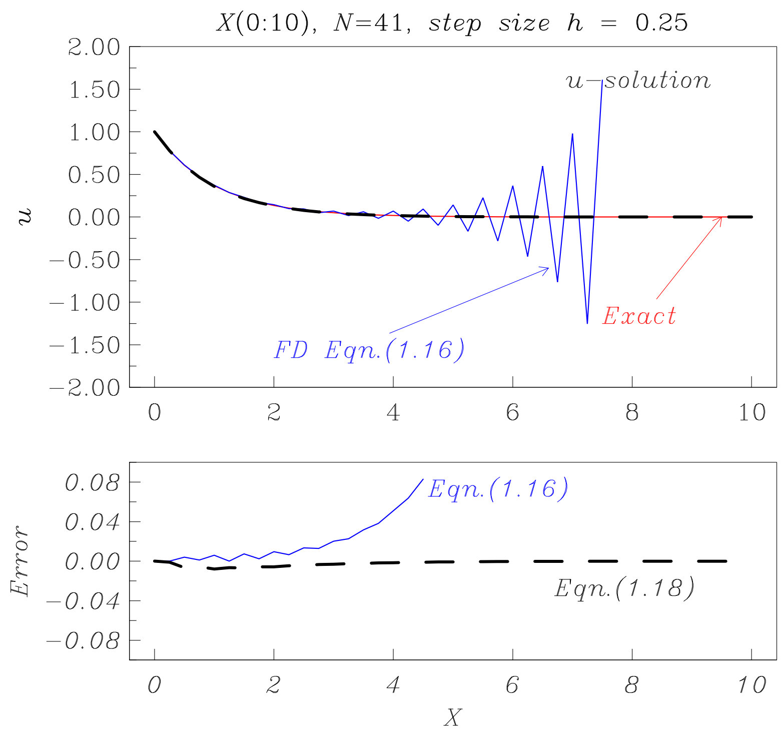

Figure 3.7: Solution of $u' = -u$ with $u(0) = 1$, N=41 grid points, using expression (3.18) given by the dashed line (– – –).

Comparing figure 11.2 with figure 3.6, why do you think the expression (3.18) gives us a much better result then using Eqn. (3.16)?

3.6.1 Numerical stability of ODEs

To conceptualise ideas, consider a general ODE of the form
$$\frac{du}{dt} = -\lambda u, \quad \text{given some initial state} \ \ u(0) = \alpha, \tag{3.19}$$
and $\lambda > 0$. The **Euler**-method is the simplest example of an **explicit** method. We may discretise the above as follows
$$U_{n+1} = U_n(1 - h\ \lambda), \tag{3.20}$$
hence, given some initial state at $u(0)$, it is trivial to see we can write the solution as
$$U_n = u(0)(1 - h\ \lambda)^n. \tag{3.21}$$
Now we know that the exact solution is given by
$$u(t) = u(0)\ e^{-\lambda t},$$

which in the limit as $t \to \infty$ must go to zero. It follows for Eqn. 3.20 to give this behaviour we require $(1 - h\ \lambda)^n \to 0$ as $n \to \infty$. This will be so, provided

$$|(1 - h\ \lambda)| \leq 1. \tag{3.22}$$

This condition can also be related to the requirement that the **general error** *(see § 3.6.2) should be damped (or remain constrained) as the t-variable increases*. Hence it follows we require

$$h \leq \frac{2}{\lambda}$$

for stability of the numerical method – this is known as **conditional stability**. The numerical solution blows up if $h > 2/\lambda$.

Next consider discretising Eqn. 3.19 as follows

$$U_{n+1} - U_n = -h\ \lambda\ U_{n+1}, \tag{3.23}$$

i.e. known as the fully **implicit** discretisation, which upon re-arrangement gives

$$U_n = \frac{u(0)}{(1 + h\ \lambda)^n}. \tag{3.24}$$

For numerical stability, we thus require

$$\frac{1}{(1 + h\ \lambda)} \leq 1, \tag{3.25}$$

In this case we see the method is **unconditionally** stable for any h ! Though of course the larger h is then truncation errors may well be significant. Just because a method is stable does not mean it is accurate – accuracy is attained by making the step h asymptotically small.

Stability limits for the forward Euler and Implicit method are shown in Fig. 3.8. More gener-

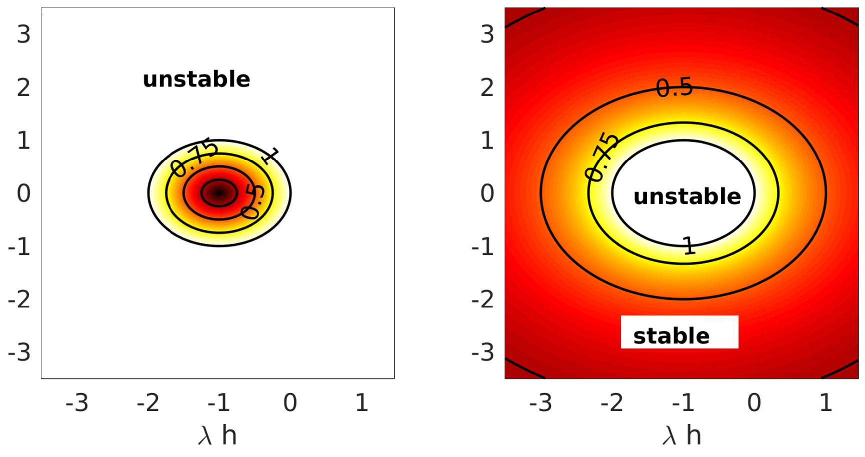

Figure 3.8: Stability region (contours ≤ 1) for the forward Euler Eqn. 3.22, and fully implicit method Eqn. 3.25.

ally we need to allow λ to be a complex value[§]. A method is then called **absolutely stable**

[§]for a comprehensive treatment see Grifiths, D. F. & Higham, D. J. (2010), *Numerical Methods for Ordinary Differential Equations – Initial Value Problems, Springer SUMS.*

if $Re(\lambda) < 0$, such that the numerical solution for, say, a scalar equation (such as Eqn. 3.19) or more general form

$$\frac{du}{dt} = \lambda f(u,t),$$

where the general errors decay to zero. We call the **region of absolute stability** the set of complex numbers

$$z = \lambda h.$$

For Euler's method, with $f(u,t) = u(t)$ the region of absolute stability is

$$|1+z| \leq \ 1.$$

If $h\lambda$ is real, the interval of absolute stability is given by

$$-1 < 1 + h\lambda < 1.$$

which means that z must be in a unit disk in the complex plane centered at $(-1,0)$. To see this, let $z = x+iy$, then the above leads to

$$|1+z|^2 = \ 1 \ \text{ or } \ (x+1)^2 + y^2 = 1.$$

Undertake similar analysis for expressions 3.17 and 3.18. What are the stability restrictions with these expressions if any?

3.6.2 General Error discrete form

Now the finite difference given by Eqn. 3.20 was constructed by setting the truncation term e to zero. Retaining this in the FD-form gives

$$\hat{u}_{n+1} = \hat{u}_n(1 - h\ \lambda) + h^2 e_n, \ \text{ where } \ e_n \equiv u''_n/2. \tag{3.26}$$

Subtracting this from Eqn. 3.20, namely constructing the general **error propagation** equation by defining $\hat{z}_n = \hat{u}_n - U_n$ gives :

$$|\hat{z}_{n+1}| = \beta|\hat{z}_n| + h^2|e_n|, \ \text{ where } \ \beta = 1 - h\ \lambda. \tag{3.27}$$

There are two aspects we expect, (1) as $h \to 0$ the truncation error contribution approaches zero; and (2) the general error $\hat{z}_n$ remains constrained for all n and ideally remains "infinitesimally" small **relative to the true solution** U_n as $n \to \infty$. That is for stability of the numerical discretisation, the errors $\hat{z}$ should not be magnified by the algorithmic process.

Considering Eqn. 3.27, it follows $\hat{z}_0 = 0$, given we enforce a known (exact) initial condition to start the procedure, thus at the $n = 1$'th-step

$$|\hat{z}_1| = h^2|e_n| = Ah^2, \ \text{ where } \ A \ \text{ is some positive constant.}$$

Next considering a sequence of steps we get

$$|\hat{z}_{n+1}| = Ah^2(1 + \beta + \beta^2 + \beta^3 + \ldots + \beta^n),$$

i.e. a geometric sum, hence for numerical stability of the method, or the errors to be constrained we require

$$|\hat{z}_{n+1}| = Ah^2\,|\beta^n| \ \to 0 \ \text{ as } \ n \to \infty.$$

Numerical stability of the scheme is achieved if the norm of the error remains bounded as n increases, it follows

$$|\beta| = |1 - h\lambda| < \ 1.$$