1 Chapter 0: Welcome to Computational methods for partial differential equations

Lecturer: Dr Marco Fasondini, m.fasondini@leicester.ac.uk

Here we

1. Introduce the topic of computational methods for PDEs
2. Discuss module admin: see the Blackboard page of this module
3. Discuss programming / coding
4. Discuss the sources / reading list

1.1 Computational methods for PDEs

The purpose of this module is to introduce you to the analysis and implementation of computational methods for partial differential equations. We shall study **finite difference** methods, **spectral collocation** methods, the **ultraspherical spectral** method and, time permitting, the **finite element** method.

A broader perspective: A quote from *An Applied Mathematician's Apology* by L.N. Trefethen, 2022:

... the language of the laws of nature, partial differential equations or PDEs. When Maxwell discovered that light and radio waves are one, it was because of a PDE. When Einstein predicted gravitational waves, it was because of a PDE. Chemistry is built on Schrodinger's equation, fluid mechanics on the Navier-Stokes equations, and civil engineering on the equations of elasticity.

A few important PDEs (there are many variations!)

- Laplace equation ($f = 0$) / Poisson equation ($f \neq 0$)

$$\nabla^2 u = f \qquad \text{or} \qquad \Delta u = f$$

An elliptic equation

- Heat equation

$$u_t - \nabla^2 u = f$$

A parabolic equation

- Transport / Advection and Wave equation

$$u_t + (v \cdot \nabla) u = 0$$
$$u_{tt} - \nabla^2 u = f$$

Wave equation: a hyperbolic equation

- Navier's equations : small-strain elasticity

$$-\mu\nabla^2\mathbf{u} + (\lambda+\mu)\nabla(\nabla\cdot\mathbf{u}) = \mathbf{f}$$

- Navier-Stokes equations : incompressible viscous fluids

$$\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} = \nu\nabla^2\mathbf{u} - \nabla p, \quad \nabla\cdot\mathbf{u} = 0$$

Numerical solutions to the (quasilinear) Navier-Stokes equations have been computed for decades (for example, these equations are used for weather prediction), however the mathematical analysis of these PDEs is still not settled (see the Wiki page on Navier-Stokes existence and smoothness): the Clay Mathematics Institute in May 2000 made this problem one of its seven Millennium Prize problems in mathematics. It offered a US$1,000,000 prize to the first person providing a solution for a specific statement of the problem:

Prove or give a counter-example of the following statement:

In three space dimensions and time, given an initial velocity field, there exists a vector velocity and a scalar pressure field, which are both smooth and globally defined, that solve the Navier-Stokes equations.

- Reaction-Diffusion systems; example, Brusselator: simple model for an autocatalytic reaction (arises also in enzymatic reactions, plasma and laser physics)

$$\begin{aligned} u_t &= B + u^2 v - (A+1)u + \alpha\nabla^2 u \\ v_t &= Au - u^2 v + \alpha\nabla^2 v \end{aligned}$$

- Black-Scholes equation (mathematical finance)

$$\frac{\partial u}{\partial t} + \frac{1}{2}x^2\sigma^2\frac{\partial^2 u}{\partial x^2} + rx\frac{\partial u}{\partial x} - ru = 0$$

Through a change of variables, this equation can be reduced to the heat equation. According to Nassim Taleb, this equation should be called the Fokker-Plank and / or Kolmogorov Forward Equation and he also explains why we never use the Black-Scholes equation. The use of mathematical models have resulted in mixed fortunes in finance: for example, there's the cautionary tale When Genius Failed: The Rise and Fall of Long-Term Capital Management (in which Myron Scholes was involved) and then there's the extremely successful Renaissance Technologies, a hedge fund founded by the mathematician Jim Simons, see the book *The Man who Solved the Market: How Jim Simons Launched the Quant Revolution.*

- Maxwell's equations
- Einstein field equations
- Dirac equation
- ...

Only in very few cases are we able to solve PDEs exactly, so we use computational methods to approximate solutions to PDEs. Here are some videos of simulations produced by computing solutions to differential equations:

- Cahn-Hilliard equation: (phase separation of a binary fluid)

$$c_t = \Delta\Big(c^3 - c - \gamma\Delta c\Big)$$

- Molecular Dynamics: (or more general Hamiltonian dynamics)

$$m_n \ddot{r_n} = -\nabla E(r_1, \ldots, r_N)$$

used here to model water turing into ice.

- Nonlinear elasticity
- thermal convection in earth's mantle
- spread of covid-19
- simulating the spread of an epidemic
- salt crystal melting in water
- NASA climate simulations
- Predicting matter distribution in the universe
- flow past an airfoil - high resolution
- Aerodynamics of Different Geometries - 2d Navier-Stokes Simulation
- Another Navier-Stokes simulation

1.1.1 Reality → Model→ Simulation → Prediction

- Mathematical models of "real-world" processes (physics, chemistry, life sciences, engineering, ...) more often than not take the form of a system of ordinary or partial differential equations, integral equations, or closely related models. (sometimes stochastic, on networks, etc, but this is beyond the scope of this module)
- In this module we take the mathematical model as given and study how it is solved numerically.
- But we should not forget what the origin of the mathematical model is so that we are appreciative of the requirements.
  - the model is an approximation of reality, the numerical scheme / computational method is an approximation of the model.
  - how accurate is the model, i.e. how accurate should the numerical solution be?
  - how accurate are the model parameters?
  - are real-time simulations required?
  - long-time behaviour?
  - inverse and control problems

1.2 Module admin

See the Blackboard page of this module

1.3 Programming / coding

All the code in the lecture notes are in the Julia programming language. You are free to use whichever programming language you prefer (e.g., Matlab or Python) and Matlab code will be available on Blackboard.

If you prefer to do your coding in Matlab and need to refresh your memory, have a look at this MathWorks tutorial (you may need to create a mathworks account with your student email address to access it). For those using Matlab, to perform some of the numerical experiments in this module, you will find it helpful to use the Matlab package called Chebfun (see Chapter 1: Exercise 2). In Julia, the counterpart of Chebfun is ApproxFun.jl.

- If you have no programming background at all, the module is doable but will be hard.
- If you have only minimal background, then please ask questions during lectures, or come to office hours.

Back to Julia: it is a modern, compiled, high-level, open-source language developed at MIT. It is becoming increasingly important in high-performance computing and AI and it is used by Astrazeneca, Moderna and Pfizer in drug development and clinical trial accelleration, IBM for medical diagnosis, MIT for robot locomotion, and elsewhere.

Julia is a relatively young programming language, which aims to combine

- the simplicity of Matlab, *(not really, but maybe close enough...)*
- the dynamism of Python, *(true ... except for the pre-compilation times which are slowly improving...)*
- the speed of C and *(definitely true; this is impressive!)*
- the meta-programming capabilities of Lisp. *(also true; but we won't need it in this module.)*

For those interested in using Julia, see the notes titled "Julia" on Blackboard, which have instructions for running the Julia code in the notes in a Jupyter notebook. The Jupyter notebooks are available on Blackboard and have the file extension ".ipynb". The notes for this module are Jupyter notebooks that have been converted to PDFs using the package Weave.jl.

1.4 Sources

I used the following sources for writing the lecture notes (please ask me for relevant page numbers / chapters on a particular topic):

1. *A Practical Guide to Pseudospectral Methods* (1998), by B. Fornberg

2. *Computational Methods for Partial Differential Equations* (2009), notes by M. Georgoulis

3. *A First Course in the Numerical Analysis of Differential Equations* (2009), by A. Iserles

4. Fast algorithms using orthogonal polynomials (2020), by S. Olver, R.M. Slevinsky and A. Townsend. *Acta Numerica*, 29, pp.573-699.

5. Sheehan Olver's Numerical Analysis notes (2022)

6. Sheehan Olver and M. Fasondini's Applied Complex Analysis notes (2021)

7. Christoph Ortner's notes on Numerical Methods for Differential Equations (2022)

8. *Finite Difference and Spectral Methods for Ordinary and Partial Differential Equations* (1996), by L.N. Trefethen

9. *Spectral Methods in Matlab* (2000), by L.N. Trefethen

10. *Approximation Theory and Approximation Practice* (2019), by L.N. Trefethen

11. A MATLAB differentiation matrix suite (2000), by J.A.C. Weideman and S.C. Reddy, *ACM Transactions on Mathematical Software* (TOMS), 26(4), pp.465-519.