

Lecture 5

Spatial discretisation : finite differences

D. Dickinson

d.dickinson@york.ac.uk

York Plasma Institute, School of PET, University of York

Semester 1


Overview of course

This course provides a brief overview of concepts relating to numerical methods for solving differential equations.

Topics to be covered include:

- ~~Integrating ODEs~~
 - ~~Explicit techniques~~
 - ~~Implicit techniques~~
 - ~~Using Scipy to integrate ODEs~~

Boundary value problems
- commonly: how things vary in space

- Spatial discretisation
 - Finite differencing  uses matrices
 - Spectral methods
 - Finite elements
- Particle In Cell (PIC) approaches
- Continuum techniques

Notes available on the VLE.

Overview this lecture

This lecture will look at

- Two point boundary value problems
- Finite difference approximations

Two point boundary value problems

So far we've looked at how to integrate 1st order ODEs of the form


$$\frac{\partial \mathbf{f}}{\partial t} = F(\mathbf{f}, t)$$

e.g. heat equation
depends on how hot
ends of plate are

given some initial conditions for \mathbf{f} . Such situations are often referred to as **initial value** problems.

You will often come across equations depending on spatial derivatives but not on time derivatives. These often occur when you're looking at a **steady state situation** (including when calculating magnetic equilibria) and in calculating the EM fields. For example consider the **1d Poisson equation**

Could convert into 2 1st order ODEs, but then can only enforce initial conditions, and see how it evolves.

$$-\frac{\partial^2 \phi}{\partial x^2} = \rho$$


want to know final state

But, often need to enforce values/conditions at multiple points

Two point boundary value problems

Whilst we can use the techniques studied so far to integrate a spatial ODE given some initial condition, say $\phi(x=0) = 0$, we can't enforce any conditions at the end point, $x = x_f$. This is important as with spatial problems we often need to enforce conditions at multiple locations, such as at each end of the simulation domain. The resulting systems are sometimes referred to as two point boundary value problems.

We'll see later that enforcing multiple boundary values can make the system more demanding to solve than initial value problems.

Let's forget the boundaries for now and look at ways of evaluating the derivative of a discretised function.

write down derivative of phi in terms of values that we have

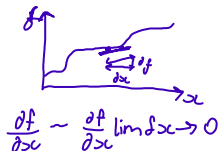
Finite differences

We'll start by looking at the finite difference (FD) method, which is widely used and relatively intuitive.

We approximate the solution by its value on a discrete grid, i.e. $f_i(x) \approx f(x_i)$.

As with the time integration methods we begin with a Taylor expansion:

$$f(x_{i+1}) \approx f(x_i) + \frac{df}{dx} \delta x + \frac{\delta x^2}{2} \frac{d^2 f}{dx^2} + \dots$$



We can then rearrange this to give

$$\left. \frac{df}{dx} \right|_{x_i} \approx \frac{\overset{\text{value at next point - value at current point / spacing}}{f(x_{i+1}) - f(x_i)}}{\delta x} - \frac{\delta x}{2} \frac{d^2 f}{dx^2} - \dots$$

first order, error scales with dx

which provides an estimate for the gradient of f at x_i in terms of $f(x_i)$ and $f(x_{i+1})$ and higher order derivatives. Neglecting the higher order derivatives allows us to calculate the gradient simply from the discrete values of f . As the dominant error term is proportional to δx this is a first order method. This is known as the right or **forward difference**.

Look at how function changes divided by grid space

Finite differences

Taylor expanding to the left leads to

$$\left. \frac{df}{dx} \right|_{x_i} \approx \frac{f(x_i) - f(x_{i-1}))}{\delta x} + \overbrace{\frac{\delta x}{2} \frac{d^2 f}{dx^2}}^{\text{error term}} + \dots$$

Again is 1st order accurate

Same error as forward difference but with opposite sign

which is the left or **backward** difference approximation.

The forward and backward approximations are both 1st order accurate. Noting that the dominant error term in these approximations have opposite sign we can **average these to give**

$$\left. \frac{df}{dx} \right|_{x_i} \approx \frac{f(x_{i+1}) - f(x_{i-1}))}{2\delta x} + \mathcal{O}(\delta x^2)$$

error average

which is the **2nd order accurate central differencing method**.

It's possible to extend this procedure to higher order derivatives, e.g.

$$\left. \frac{d^2 f}{dx^2} \right|_{x_i} \approx \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1}))}{\delta x^2} + \mathcal{O}(\delta x^2)$$

Approximates the derivative

which is a 2nd order approximation to the second derivative.

Finite differences

Let's consider the following equation

$$\alpha(x) \frac{d^2 f}{dx^2} + \beta(x) \frac{df}{dx} = \rho(x)$$

evaluate α and β at point we're using,
labelled as i

Using our centred finite differences we can write this as

$$\alpha_i \frac{f_{i+1} - 2f_i + f_{i-1}}{\delta x^2} + \beta_i \frac{f_{i+1} - f_{i-1}}{2\delta x} = \rho_i$$

no differential operators left

which can then be rearranged as

$$\overset{\text{A}}{\left[\frac{\alpha_i}{\delta x^2} - \frac{\beta_i}{2\delta x} \right] f_{i-1}} + \overset{\text{B}}{\left[-\frac{2\alpha_i}{\delta x^2} \right] f_i} + \overset{\text{C}}{\left[\frac{\alpha_i}{\delta x^2} + \frac{\beta_i}{2\delta x} \right] f_{i+1}} = \rho_i$$

Finite differences couple
together different part of
the grid

This shows that f at a grid point depends on the value to the left and right.

Finite differences

Due to the coupling of values at different grid points we actually end up with a set of N coupled equations for f (where N is the number of grid points). This can be written as a matrix equation:

$$\begin{pmatrix} B_1 & C_1 & & & \\ A_2 & B_2 & C_2 & & \\ & A_3 & B_3 & C_3 & \\ & & \ddots & \ddots & \ddots \\ & & & A_N & B_N \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_N \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \vdots \\ \rho_N \end{pmatrix}$$

Coupling gives you set of simultaneous equations to solve

where we've defined

$$A_i = \frac{\alpha_i}{\delta x^2} - \frac{\beta_i}{2\delta x}, \quad B_i = -\frac{2\alpha_i}{\delta x^2}, \quad C_i = \frac{\alpha_i}{\delta x^2} + \frac{\beta_i}{2\delta x}$$

A matrix with this structure is known as **tridiagonal**, and can be solved (inverted) easily.

Finite differences

Now that we've discretised the differential equation and found that this leads to a set of coupled equations we need to return to the boundaries. Consider the first row of the matrix, which defines our left boundary:

We don't want to solve for boundary values, as these are the conditions we enforce

$$B_1 f_1 + C_1 f_2 = \rho_1$$

Use first rho to set condition - use to choose B1 and C1 values and get f values

One of the most common choice of boundary condition is to set f at the boundary to a fixed value a , known as a **Dirichlet** boundary. We can achieve this by setting $B_1 = 1$ and $C_1 = 0$ and $\rho_1 = a$ which gives $f_1 = a$.

Another common choice of boundary condition is to set the gradient to a fixed value a , known as a **Neumann** boundary. One way to achieve this would be to choose $B_1 = -1/\delta x$, $C_1 = 1/\delta x$ and $\rho_1 = a$. The first line is then

$$\frac{f_2 - f_1}{\delta x} = a$$

which sets the 1st order forward difference approximation of the gradient to a .

Finite differences

Whilst the implementation of the Neumann boundary just introduced preserves the tridiagonal nature of the matrix, it has some downsides. As we're effectively using a 1st order method for the boundary condition, *this will limit the whole solution to 1st order accuracy* as every point is coupled together.

To retain 2nd order accuracy everywhere we need a more accurate approximation for the boundaries. Returning to the Taylor expansion of f we can consider the expansion at the next point¹, $f(x_i + 2\delta x)$

$$f_{i+2} \approx f_i + 2\delta x \frac{df}{dx} + 2\delta x^2 \frac{d^2 f}{dx^2} + \dots$$

This can then be used to cancel the $d^2 f/dx^2$ error term in the forward difference, leading to

$$\frac{df}{dx} \approx \frac{-f_{i+2} + 4f_{i+1} - 3f_i}{2\delta x}$$

Breaks tridiagonal nature of matrix

¹We can note the similarity to multi-step time integration techniques.

Finite differences

So now we know how to represent our discretised equation on the computer we need to know how to solve it. Fortunately solving matrix equations of the form

$$\underline{M} \cdot \underline{f} = \underline{\rho}$$

is a common task and a range of approaches have been developed.

- If the matrix is tridiagonal then the *Thomas algorithm* (serial) or *Cyclic reduction* (parallel) are available.
- If the matrix is near tridiagonal then we can start with an approximate solution assuming the matrix is tridiagonal and then apply the *Shermann-Morrison* formula to refine the solution.
- In more general situations it's a good idea to use a linear algebra library² which can either perform a direct solve (small matrices, $N < \sim 10,000$) using something like *LU decomposition* or an iterative solve (for large/parallel matrices) using *GMRES* or similar.

²There are many linear algebra libraries including LAPACK and PETSc. As we'll see next time SciPy includes a linear algebra module.

Summary

Today we've seen how we can approximate derivatives of a function using finite difference approximations and how this leads to a matrix equation in which all locations are coupled together.

Picking (and implementing) appropriate boundary conditions is an important part of solving such equations. Due to the coupling of locations errors in the boundary will propagate throughout the system.

Next time we'll look at some practical considerations in implementing such a problem which takes advantage of the (near) tridiagonal matrix structure.