Scientific Computing I

Summary by Gilberto Lem

Contents

1 Introduction to the Discipline

The classical approaches to gain scientific knowledge are theory and experiment, however simulation is sometimes needed:

- **Where experiments are impossible:** Astrophysics, Geophysics, Meteorology, Climate and Ocean Modelling.
- **When there is no second try:** Stability of buildings, Astronautics
- **When experiments have harmful side-effects:** Propagation of pollutants, nuclear research
- **When experiments are expensive:** Car industry

The standard simulation pipeline is the following:

1. Mathematical Modeling
2. Numerical Treatment
3. Implementation
4. Visualization, including validation and uncertainty quantification

2 Discrete Models

The general approach for this kind of models is to make a **discrete system** of the form $x^{(n+1)} = f(x^{(n)})$, i.e. express a certain quantity in an iteration $n+1$ as a function of the quantity in the previous iteration n.

We could use this model to find equilibrium states, analyse the evolution of the quantity, or see the behavior when $n \to \infty$.

2.1 Discrete Linear Systems

We can make a linear discrete system, with the form:

$$x^{(n+1)} = Ax^{(n)},$$

where A is a matrix.

2.1.1 Finding equilibrium states

An equilibrium state would be such state that does not get changed with iterations, i.e. eigenvectors corresponding to eigenvalue 1:

$$x^{(n)} = Ax^{(n)}$$

2.1.2 Evolution in terms of initial state

We could find the state in any iteration in terms of the initial state by applying the matrix repeatedly:

$$x^{(n+1)} = \underbrace{A...A}_{n}x^{(0)} = A^n x^{(0)}$$

2.1.3 Behavior for $n \to \infty$

Let $\{\lambda_k, v_k\}$ be the eigenpairs of A. Then we could express the initial state as a linear combination of the eigenvectors, and use their properties:

$$x^{(n+1)} = A^n \left[\sum_k c_k v_k \right] = \sum_k c_k A^n v_k = \sum_k c_k \lambda_k^n v_k,$$

where $c_k =< v_k, x^{(0)} >$.

If A is a stochastic matrix, then $|\lambda_k| \leq 1$, which means that when $n \to \infty$ our λ_k^n will either go to zero or to 1. This means that our convergence states are the eigenvectors with eigenvalues 1. The convergence speed will depend on value of the biggest eigenvalue different than 1.

If the set of states is finite and the matrix A is constant (i.e. probability only depends on current state, it has no memory), this system is called a **Markov Chain**.

Stochastic Matrices
Stochastic matrices can be thought as probability matrices, where each row (or column, or both), has the probability of going from one state to the other. They have the following properties:
Their eigenvalues are ≤ 1 in magnitude.
At least one eigenvalue is 1; all the elements of the corresponding eigenvectors are ≥ 0.
The sum of the elements of the rest of the eigenvectors is 0.
Let $y = Ax$, then the sum of y is equal to the sum of x.

2.2 Problems in Practice

2.2.1 Separate Partitions

We could have the case where our matrix can be separated in two sets of states that are not linked from each other. In this case we could express:

$$A = \begin{pmatrix} A_I & 0 \\ 0 & A_{II} \end{pmatrix},$$

where A_I and A_{II} are stochastic matrices with eigenvectors a_I and a_{II} for eigenvalue 1. This would be equivalent to having two independent stochastic processes, with independent rating of partitions I and II, with the peculiarity that A has three eigenvectors with eigenvalue 1: $\{\, (a_I \, a_{II})^T,\ (a_I \, 0),\ (0 \, a_{II})^T\}$.

2.2.2 Regularization

If at least one eigenvalue is close to 1, but not equal, we will have slow convergence. To overcome this we could define a slightly different matrix:

$$\tilde{A} \coloneqq \alpha A + (1-\alpha)\frac{1}{n} e e^T,$$

$\frac{1}{n} e e^T$ is just a normalized matrix of ones. Note that when α is 1, we have the original matrix.

We could solve the system for the new matrix, which since $e^T x = 1$ (sum of elements of a probabilistic vector), would be:

$$x^{(n+1)} = \alpha A x^{(n)} + (1-\alpha)\frac{1}{n} e$$

It is not the original system if α is not one, however, if α is small, the system would converge faster, although to a less accurate solution. This represents a tradeoff between speed and accuracy.

3 Ordinary Differential Equations

If instead of a discrete model we define a continuous model, we have a different type of mathematical problem, which can make use of powerful tools such as differential equations and calculus. In this chapter we will illustrate continuous models using population functions $p(t)$.

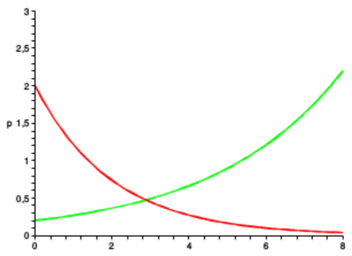
3.1 One-Species Population Model Examples

We start with a simple model, which assumes that the birth rate and the death rate are proportional to the size of the population. The difference between the two rates would yield a growth (or decay rate) r :

$$\frac{\mathrm{d}p(t)}{\mathrm{dt}} = rp(t)$$

with initial condition $p(0) = p_0$.

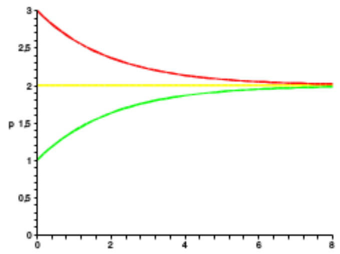
This is called the **Model of Malthus**, and represents exponential growth (or decay) of population.



However, populations cannot grow undefinitly, because there are limited resources. We would expect the population to start growing slower or even stop when the limit of resources is being reached, and if the limit is surpassed, the resources would not be enough and we would expect a decrease in population. Mathematically, we would expect the derivative to decrease to 0 in a certain limit, and beyond that we would expect the derivative to be negative.

In our ODE, we have a linear dependence between the derivative and p, and we get an exponential solution. We could generalize that linear dependence to achieve our requeriments:

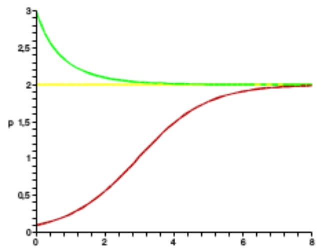
$$\frac{\mathrm{d}p(t)}{\mathrm{dt}} = \alpha(1 - \frac{p(t)}{\beta})$$



Here when $p(t)$ reaches the value of β, the derivative is zero, and if $p(t)$ is greater than that value, the derivative is negative. This is called the **Verhulst Model.**

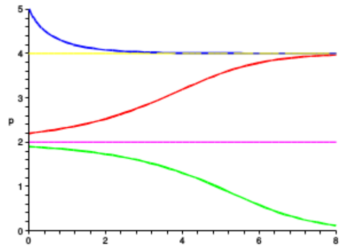
However, when we changed the form of the equation we lost the property that the population growth is small when the population is small and increases with the population. We could fix this by simply multiplying by the population. This would ensure that the growth at the beginning is slow. This is called the **Logistic Growth Model:**

$$\frac{\mathrm{d}p(t)}{\mathrm{dt}} = \alpha\left(1 - \frac{p(t)}{\beta}\right)p(t)$$



Furthermore, in an even more realistic model, we would expect a certain minimum population for the species to keep existing. If there exist only a few specimens of a certain species, the rate of reproduction to death could not be enough to keep the population stable or growing. We can add another shift to our model to reflect this:

$$\frac{\mathrm{d}p(t)}{\mathrm{dt}} = \alpha\left(1 - \frac{p(t)}{\beta}\right)\left(1 - \frac{p(t)}{\delta}\right)p(t)$$



We add an additional $\left(1 - \frac{p(t)}{\delta}\right)$ factor, because we keep wanting our growth to tend to 0 when $p(t)$ tends to zero. This is called the **Logistic Growth with Threshold Model.**

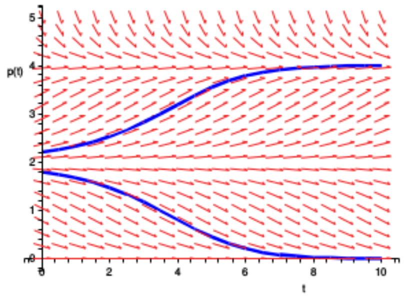
Note that we can get as many thresholds/limits we want by defining factors of the form $\left(1 - \frac{p(t)}{\gamma}\right)$, which go to zero when $p(t)$ reaches the value γ.

3.2 Analysis of ODE Models

All the previous models where relatively simple, however, analytical solutions may be difficult or even impossible to compute, and even the properties of the solutions (e.g. shape, steady state, critical points) may not be obvious. The following are some tools to analyse them:

3.2.1 Directon Fields

If we have an ODE, we can evaluate the derivative and get the value of the slope in any time and value of the population. Direction Fields are plots that show these values using arrows. The following is an example of a direction field for a Logistic Growth with Threshold model:



Note that the blue lines represent two different solutions to the ODE with two different corresponding initial values. We could sketch the solution for any initial value following the arrows in the direction field.

3.2.2 Critical Points

Critical points of an ODE are the values for $p(t)$ where $\frac{\mathrm{d}p}{\mathrm{d}t}$ is 0, which means that when $p(t)$ reaches the critical point it stays in the critical point.

Critical points in ODE's can be of three different types, depending on the behavior of the function in its neighborhood.

- **Attractive equilibrium** when the function goes to the critical point. They are asymptotically stable.

$$\dot{p} < 0 \ \text{ for } \ p = p_{crit} + \epsilon$$
$$\dot{p} > 0 \ \text{ for } \ p = p_{crit} - \epsilon$$

- **Unstable equilibrium** when the function goes away from the critical point

$$\dot{p} > 0 \text{ for } p = p_{crit} + \epsilon$$
$$\dot{p} < 0 \text{ for } p = p_{crit} - \epsilon$$

- **Saddle point** when the function gets attracted to the critical point in some parts of the neighborhood and away from it in other parts of the neighborhood.

3.3 Systems of ODE's

If we want to make a two-species model, we would have to take into account the way one population affects the other. This yields a system of ODE's. The direction fields for this kind of model are often parametric plots of p vs q.

They could be linear or nonlinear. The following is an example of a linear system of ODE's:

$$\dot{p}(t) = b_1 + a_{11}p(t) + a_{12}q(t)$$
$$\dot{q}(t) = b_2 + a_{21}p(t) + a_{22}p(t)$$

3.3.1 Linear Systems of ODE's

In general, we have the following system:

$$\dot{x} = Ax + b,$$

where x is a vector of functions, b is a constant vector and A is a matrix.

If x_λ is an eigenvector of the system, then $x_\lambda e^{\lambda t}$ is a solution to the homogeneous system (when b is 0), and the general solution, for the non-homogeneous system is:

$$x(t) = x_c + \sum_\lambda a_\lambda x_\lambda e^{\lambda t},$$

where $x_c = -A^{-1}b$ is actually a critical point.

We could look at the eigenvalues of the system to classify the critical points. As our solution has the eigenvalues in an exponential argument, if our eigenvalues are real and negative, our solution would tend to go to x_c. If our eigenvalues are positive our solution would grow exponentially, and if our eigenvalues are complex, they will oscilate. Here is a summary of the classification:

eigenval. ($\lambda_j = \mu_j + i\nu_j$)	critical point	stability
real, all $\lambda < 0$	node	stable, attr.
real, all $\lambda > 0$	node	unstable
real, $\lambda_k > 0$, $\lambda_l < 0$	saddle point	unstable
complex, all $\mu < 0$	spiral point	stable, attr.
complex, all $\mu > 0$	spiral point	unstable
complex, all $\mu = 0$	centre	stable

3.3.2 Nonlinear systems of ODE's

For nonlinear systems, we can use a linearization and apply the same approach to evaluate the nature of critical points (in their neighborhood).

$$\dot{x} = f(x(t)) \approx f(x_c) + \mathrm{J}_{\mathrm{f}}(x_c)(x(t) - x_c),$$

The eigenvalues that we would examine, would be the eigenvalues of J_{c}, which is the Jacobian matrix.

4 Numerical Methods for ODE's

Many ODE's do not have an analytical solution, and we have to use numerical methods to find approximate solutions to it. There is an easy and intuitive way of solving ODE's with initial conditions. The form of the ODE is:

$$\dot{y}(t) = f(t, y(t)), \quad y(t_0 = y_0)$$

Evaluating this expression on a grid of t and y, we could sketch a direction field, drawing the slope in each point of the grid. After that, we can follow the arrows, to get an approximation of the next step of our solution. Many methods are based in this principle, computing a slope in a point and moving in that direction to get the next point of the solution.

4.1 Explicit and Implicit Methods

Euler's Method is based in that principle, using for the derivative a finite difference approximation:

$$\frac{y_{n+1} - y_n}{\tau} \approx \dot{y}_n,$$

This would be equivalent to using the Taylor's expansion:

$$y(t_{n+1}) = y(t_n) + \tau\dot{y}(t_n) + \mathcal{O}(\tau^2)$$

Note that τ represents the stepsize, i.e. how much we are moving in the direction of the independent variable.

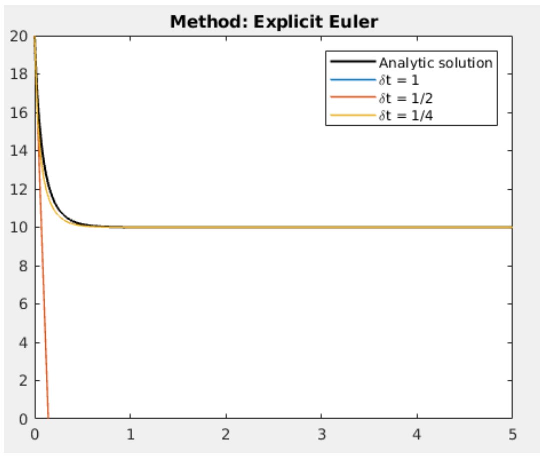
With this and an initial condition, we could get a sequence of values for solving the differential equation, however, we still have to decide how to get the value of our slope $\dot{y}$. We do have an expression to find it $f(t, y(t))$, but if we want to find the value of our solution for the step $n+1$, as it is a slope, we have the option to evaluate in the previous step (n), in the current step itself $(n+1)$, in an intermediate point, or a combination of those.

If we evaluate it in the previous step (n), we get an **Explicit** formulation. The following would be the Explicit Euler's method.

$$y_{n+1} = y_n + \tau\dot{y}(y_n, t_n),$$

Explicit formulations are very simple, however, they come with several issues. Depending on the meaning and context of the ODE, if we want to have a reasonable next step, τ would have to hold certain restrictions. For example, of our ODE represents a population system, there could be a threshold for τ for preventing us to get a negative population.

Another issue is that the value for τ could also change the stability of our solution. As we evaluate the function at the previous timestep, if the slope in that point is too big, or too negative, we could get diverging results for the next timestep. Find an example in the next figure ($\tau \equiv \delta t$):



As the derivative in the initial condition was very negative, the next evaluation for $\tau = 1/2$, was in the point $y = -\text{Inf}$, and the solution could not be computed for the following steps. However, using $\tau = 1/4$, was sufficient to have a stable solution. These restrictions on timesteps for the numerical schemes are called **Stiffness.**

One solution to the stiffness problem is to evaluate our derivative instead of in the previous step, in our step of interest:

$$y_{n+1} = y_n + \tau \dot{y}(y_{n+1}, t_{n+1}),$$

This is called an **Implicit** formulation. Note that evaluating our ODE in the current step would lead to have y_{n+1} in both sides of the equation, and solving the equation for y_{n+1}, could require a greater computational effort. Solving the equation would sometimes need a numerical method (such as Newton's method or an iterative solver) if the equation is too complicated. However, this is very useful because it lowers (and in most physical cases vanishes) the restrictions on the timestep τ.

Implicit methods would require to solve an aditional equation, using bigger timesteps and with this less iterations. Maybe some ODE would require a very small τ to get a stable solution with an Explicit method, and instead we could use an Implicit method with a less small τ, taking into account the additional computational effort needed for this. This situation represents a tradeoff between many cheap timesteps (explicit) and not that many expensive/impossible time steps.

4.2 Advanced Numerical Methods

4.2.1 Runge-Kutta Methods

Runge-Kutta Methods are higher order methods. They use the idea of considering additional evaluations of the derivative in the interval between previous and current timestep to obtain better accuracy. The general form is:

$$y_{n+1} = y_n + \tau \sum_{i=1}^{p} \beta_i f(t_n^i, y_n^i), \quad t_n^i \in [t_n, t_{n+1}]$$

These methods use numerical approximations for calculating the values of y_n^i, and choose the weights β_i using quadrature rules.

For example, the 2nd-order Runge-Kutta, also called **Method of Heun**, is of the following form:

$$y_{n+1} = y_n + \frac{\tau}{2} \left(f\left(t, y_n\right) + f\left(t_{n+1}, y_n + \tau f\left(t_n, y_n\right)\right)\right)$$

Note that it is taking the average of the derivative in the previous step and an approximate derivative in the current step, calculating a first approximation to y_{n+1} with an explicit Euler method.

Another example of 2nd-order Runge-Kutta is the **Modified Euler** method:

$$y_{n+1} = y_n + \tau f\left(t_n + \frac{\tau}{2}, y_n + \frac{\tau}{2} f\left(t_n, y_n\right)\right)$$

Note that it is similar to Euler's method, but it considers the slope in the midpoint of the interval, approximating the y value in that point with an Explicit-Euler iteration.

We can see that Runge-Kutta methods, include several intermediate steps. For higher order methods, the number of intermediate schemes increases. The multiple evaluations of the derivative make these methods expensive, however, they are stable, well-behaved, and easy to implement.

4.2.2 Multistep Methods and Midpoint Rule

Until now we have used only the interval between the previous step and the current step to compute our approximations. Multistep Methods use s previous steps to compute the current step. They use the integral form of the differential equation, together with a quadrature rule to solve the integrals, to acheive the following form:

$$y_{n+s} = y_{n+s-1} + \tau \sum_{j=0}^{s-1} \beta_j f\left(t_{n+j}, y\left(t_{n+j}\right)\right)$$

In general, multistep methods are of higher order than Runge-Kutta methods with only a few evaluations of f, which would be interesting if f is expensive to compute. However they have severe stability problems.

For example, one method that is formally a multistep scheme is the **Explicit Midpoint Rule**, which derives from the symmetric discretisation:

$$\dot{y}(t) \approx \frac{y(t+\tau) - y(t-\tau)}{2\tau},$$

which leads to $y_{n+2} = y_n + 2\tau f(t_{n+1}, y_{n+1})$.

This method is 2nd-order accurate with low computational effort, however, it tends to be unstable.

4.3 Properties of Numerical Schemes

In general we have the following relation:

$$\text{consistency} + \text{stability} = \text{convergency}$$

Consistency: The use of a finite difference approximation, or equivalently the truncation of a Taylor series, brings a discrepance with respect to the actual derivative. This difference is called **Local Discretisation Error**, and we will denote it with $I(\tau)$.

If $I(\tau) \to 0$ when $\tau \to 0$, i.e. when our discretisation approaches the true derivative as our discretisation unit approaches 0, then the numerical scheme is called consistent. A numerical scheme is called **p-th order consistent** if $I(\tau) = \mathcal{O}(\tau^p)$. This would mean for example, that if we divide our timestep by 2, our error would be reduced by 2^p.

Convergency: The **Global Discretisation Error** is obtained comparing the numerical solution with the exact solution. It is formulated as the maximum difference between the two solutions, and is denotated with $e(\tau)$.

Convergency implies that when $\tau \to 0$, $e(\tau) \to 0$, i.e. the maximum difference between the actual solution and the numerical solution vanishes as our precision in timestep goes to 0. This means that investing more and more computational effort (by reducing the timestep) will then lead to better approximations for the exact solution. A numerical scheme is called **p-th order convergent** if $e(\tau) = \mathcal{O}(\tau^p)$.

Stability: A stable scheme is such that the error decreases with the number of iterations. Abstractly, one could say that a numerical scheme is stable if its error is proportional to a^n, where n is the number of iteration, and $|a| < 1$. Single step methods are always stable, while multistep methods have additional stability conditions.

Note that although a method can be consistent and stable, these are asymptotic terms: these properties are totally true only when our discretisation tends to zero.

4.3.1 Problems for Numerical Methods for ODE's

III-Conditioned Problems: When small changes in input entail completely different results. This could be only at critical points or it could be everywhere, and would represent an important risk for non-precise inputs and round-off errors.

Stiffness: As we have seen in section 4.1, stiff equations lead to instabilities if the timestep does not obey certain restrictions. The remedy for stiffness is to use implicit or semi-implicit methods.

5 Partial Differential Equations

If we take into account more than one independent variable in an ODE, we get a PDE. These equations are a result from a limit process ($h \to 0$) from a discrete model. Stationary PDE's are time-independent, the whole equation depends only in spatial coordinates. On the other side, Instationary PDE's have derivatives in space and in time. Classical examples are:

- Laplace Equation: $\triangle u = 0$
- Poisson Equation: $\triangle u = g\left(\vec{r}\right)$
- Heat Equation: $k\triangle u = u_t$
- Wave Equation: $k\triangle u = u_{tt}$

With the following notation:

$$\triangle := \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}; \quad u_\xi = \frac{\partial u}{\partial \xi}.$$

One standard method of solving PDE's anallytically is **Separation of Variables**, where one makes the assumption that the solution is a product of functions each one dependent only in one of the independent variables. For example, for a PDE which depends on x and t :

$$u(x,t) = X(x)T(t)$$

We can plug that ansatz into the equation, get the dependence on x on one side of the equation and the dependence on t in the other and solve it.

5.1 Boundary/Initial Conditions

For a pth order ODE, we have to consider p restraining conditions to find a unique solution. These are generally called initial conditions or boundary conditions. Initial conditions make reference to the time-coordinate, and describe the state of the system at the beginning of our desired observation. The boundary condition make reference to the state of the system in the boundaries of our interest domain.

In a similar way, a PDE would need a restriction condition for each order of derivation and for each variable. For example, the wave equation would need two restriction conditions for each coordinate in space, and two restriction conditions for time. Note that each restriction condtion represent the value of the function (or its derivatives) in a hyperplane.

We call them **Dirichlet** boundary conditions when the function is fixed on the boundary:

$$u(x,y,z) = \phi(x,y,z), \text{ on } \delta\Omega$$

We call them **Neumann** boundary conditions when the normal derivative is fixed on (part of) the boundary:

$$\frac{\partial u}{\partial n}(x, y, z) = \phi(x, y, z), \text{ on } \delta\Omega$$

6 Numerical Methods for PDE's

A continuous form of a solution is hard to store in the computer, and often analytical solving is not possible. That is why we use discretisation.

6.1 Discretisation

Consists on solving the function only in some points. There are several types of discretisation: finite differences, finite volumes, finite elements.

6.1.1 Finite Differences and Stencil Notation

The Finite Differences method consists on replacing in each point the partial derivative by a different quotient. For example:

$$\frac{\partial^2 u}{\partial x^2}(\vec{r}_{i,j}) \approx \frac{u(r_{i+1,j}) - 2u(r_{i,j}) + u(r_{i-1,j})}{h_x^2}$$

Making a discretitation of these kind for each derivative would lead to a different equation for each $r_{i,j}$, which can be expressed as a linear system of equations:

$$A_h r_h = u_h,$$

where h makes reference to the discretisation step.

Generally the resulting A_h matrices are sparse, and only the diagonals close to the main diagonal are non-zero. One can use **Stencil Notation** to reflect in a more compact way the contents of these matrices. For example, for the following matrix:

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

we note that the pattern of the second row is repeated along the whole matrix. We could express that pattern in a much compact stencil notation as:

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

6.1.2 Finite Volumes

Consists on computing fluxes on the boundaries of control volumes and examine conservation laws on them. For this, one can subdivide the domain Ω where the PDE is valid into elements Ω_{ij}, where the value of the function $u \to u_{ij}$ on each element is an average. After that one can integrate the PDE and discretize the values of the derivatives to get to a system of linear equations, which would end up in having the same system as the one obtained with the Finite Differences approach.

6.2 Solving the linear systems

After discretising, we end up with a system of linear equations that we have to solve in order to get an approximate solution to our PDE. For solving them, there are two kinds of solving strategies:

- **Direct Solvers:** Solving for a full matrix by applying a QR method or LU method (or simply a MATLAB backslash operator); or solving for a sparse matrix.

- **Iterative solvers:** Point by point processing methods, e.g. Jacobi and Gauss-Seidel.

6.3 Splitting up space and time dependencies

The most natural form of PDE is one that depends on the space coordinates and in time. One way to solve this kind of PDE is to eliminate the spatial-differential part with a spatial discretisation, which would lead to having an ODE for each member of a grid. Then we could solve each ODE with an Explicit or an Implicit scheme (the restriction in timestep for the explicit scheme would now depende additionally on the spatial step).

The general error of our solution would now have a spatial and a temporal component:

$$e = e_{space} + e_{time}$$

For high-order space discretisations, the time component would dominate, and for high–order time discretisation, the space component would dominate. It is recommended to get a balance in order of discretisation for both components.

6.4 Stability Analysis

How do we predict if a solution is going to be stable according to our discretisation steps?

6.4.1 Von Neumann Stability Analysis

One way, the **Von Neumann** stability analysis, is to assume that the error of our discretisation can be modelled as a Fourier series in the form:

$$e_j^{(m)} = \sum_k (a_k)^m \, e^{i\pi k \vec{r}_j}$$

where m is the number of iteration.

If we want our solution to be stable, we need $e_j^{(m)} \to 0$ when $m \to \infty$, which means that $|a_k| < 1$.

We can after that insert our error ansatz into our discretised PDE and solve for a_k, to get the regions where that condition is met.

6.4.2 Mathematical Energy

Another way is through the concept of **Energy**. Energy is the following mathematical concept:

$$E(t) := \int u^2(\vec{r}, t) d\vec{r}$$

If $E'(t) \leq 0$, energy never increases. This would be the case, making a physics analogy, if we want to avoid enlarging oscillations in the solution. We could find the regions where our discretisation steps provide stable solutions coupling our discretised PDE with a discretised form of energy:

$$E^{(m)} := h \sum_j \left(u_j^{(m)} \right)^2 ,$$

and finding the region where $E^{(m+1)} \leq E^{(m)}$.

7 Finite Element Methods (FEM)

Until now, we have been computing approximate **values** at certain grid points. The idea behind FEM is to compute an approximate **function** to solve a PDE, finding that function as a linear combination of basis functions from a complete space:

$$u = \sum_i c_i \varphi_i,$$

where u is the solution to our PDE, and $\{\varphi_i\}$ forms the complete base of ansatz (also called shape) functions, with span W.

Our goal then would be to find the coefficients c_i, and this task can be accomplished solving a linear system.

To understand how to get the coefficients we have to use mathematical concepts as Weak Form of a differential equation, and its Weak Solutions. For illustrating this concept we will take as an example the Poisson Equation.

7.1 Weak Form and Weak Solutions

Consider the Poisson Equation with Dirichlet conditions:

$$-\triangle u(\vec{r}) = f(\vec{r}) \ \text{in}\, \Omega; \ \ u = 0 \text{ on } \partial\Omega$$

Our solution u has to hold on every point of $\vec{r}$. This is equivalent to having infinitely many equalities. This is called the **exact form of the solution**. If we integrate the PDE on the whole domain Ω, the value of the integral would throw only one value, so we would go from having infinitely many equalities, to having just one.

$$-\int\limits_\Omega \triangle u = \int\limits_\Omega f$$

Note that the value of $\int_\Omega \triangle u$ could be the same for many different functions u, so we now have a set of solutions that includes the strong solution, however has lower restrictions. This set of functions is called the **weak form of the solution**. Now, if we multiply our PDE by any test function $v \in V$, it would become:

$$-\int\limits_\Omega v\triangle u = \int\limits_\Omega vf$$

If we apply integration by parts[1] to the LHS:

$$\int\limits_\Omega \nabla v \nabla u - \int\limits_{\partial\Omega} v\nabla u = \int\limits_\Omega vf$$

[1]The multidimensional form of $\int v\,\mathrm{d}u = vu|_{\partial\Omega} - \int u\mathrm{d}v$, with $\mathrm{d}u$ as the Laplacian would be the Green Theorem: $\int_\Omega v\triangle u = \int_{\partial\Omega} v\nabla u - \int_\Omega \nabla v \nabla u$

Note that when we multiplied by v we introduced a new degree of freedom. Our equation has to hold for every v, so we have infinitely many equalities again, but our "infinityness" instead of being based on our independent variables of the PDE, is based in our test-space, which we can define convieniently. We can take advantage of this and define V as a function space such that $v = 0$ on $\partial\Omega$, and with that simplfy our equation:

$$\int_\Omega \nabla v \nabla u = \int_\Omega v f$$

This is called the **weak form of the PDE**. This is still an integral form, which holds for many solutions u, and has weaker requirements: instead of requiring it to be twice differentiable, we only require its first derivative to be integrable.

7.2 Assembling the system of equations

If we define a basis of test functions $\{v_j\}$ for the test space V, and our solution holds for each v_j, then our solution is valid for every possible v. We use this to reexpress our problem. Instead of having an equation for every v we now have an equation only for each v_j. This way, we get a system of linear equations:

$$\int_\Omega \nabla v_j \nabla u = \int_\Omega v_j f$$

Furthermore, if we now insert our proposed solution as a linear combination of ansatz functions $u = \sum c_i \varphi_i$, we get:

$$\sum_i c_i \int_\Omega \nabla v_j \nabla \varphi_i = \int_\Omega v_j f$$

And we could define

$$A_{ij} := \int_\Omega \nabla v_j \nabla \varphi_i; \quad b_j := \int_\Omega v_j f$$

to express the system in the compact matrix-vector form:

$$Ac = b$$

Solving for c would give us the coefficients necessary to get our solution. Now we just need to compute the elements of A_{ij} and b_j.

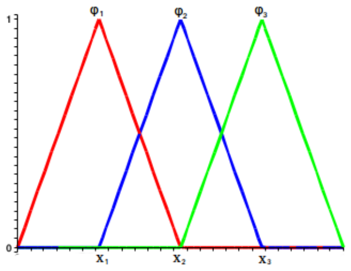
7.3 Ansatz and test functions

Recall that we have defined a space W with a set of **ansatz** functions $\{\varphi_i\}$ which linearly combined express u, and a space V with a set of **test** functions $\{v_j\}$ that until now only has the restriction of every v_j vanishing in $\partial\Omega$.

Often, V and W are chosen identical, which results in having as many equations as unknowns. This is called the Ritz-Galerkin method, and would simplify the computation of the matrix:

$$A_{ij} = \int_\Omega \nabla\varphi_i \nabla\varphi_j$$

Ideally, as we have to solve a linear system with a matrix A_{ij}, we would want it to have as little elements as possible (sparse), to spend as little computational effort as possible. For this, we would need $\int_\Omega \nabla\varphi_i\nabla\varphi_j$ to vanish in most cases. This could be achieved by defining functions that are zero everywhere except in the adjacency of a point x_i, i.e. local basis functions. Consider for example the following nodal basis:



The value of φ_i value is 1 in the node x_i and 0 in every other node, and thus A_{ij}, if you integrate in the whole space, would not vanish only for equal and adjacent indices, e.g. for the i-th row for A_{ii},$A_{i\,i-1}$ and $A_{i\,i+1}$. This pattern can be expressed very compactly with stencil notation (see section 6.1.1).

7.4 Time dependent problems

When we introduce time-dependence into the problem, as a result of the discretisation of the additional term, we get the extra matrix-vector multiplication shown in the following left hand side:

$$\frac{\partial}{\partial t}(Mc) = Ac + f$$

Following analogies with physics, M is called the **mass matrix**, and A is called the **stiffness matrix.**

The previous equation is no longer a system of linear equations, but a system of ODE's. To solve it, we would have to invert the mass matrix, which can be a heavy task. One approach is mass lumping, which consists in approximating the mass matrix with a simpler matrix, as a diagonal matrix.

However, if our basis functions are chosen smartly, we can get simple mass matrices. If we choose element-local basis functions, we would get a block-diagonal matrix, or furthermore, if we choose an orthogonal basis we would get a diagonal matrix.

7.5 Implementation

All the previous theory was valid for a whole domain. However, if we subdivide our domain and apply the method to each one of the subsets, we can get more detailed solutions. Those subdivisions are called **elements**.

$$\Omega \to \sum_k \Omega^{(k)}$$

Each element would have its basis functions. The integrals on the whole domain of the weak formulation of the PDE, as they are linear operators, would then be sums of integrals on each element.

$$\int_\Omega g = \sum_k \int_{\Omega^{(k)}} g$$

Discretised, this would lead to have a local stiffness matrix and a local system of linear equations for each element. These are called element stiffness matrices, and are generally noted by $\hat{A}^{(k)}$.

These element matrices, however, will represent just one element of a global matrix, which includes all elements and therefore can be of much higher dimension. We could define an intermediate matrix which has the same dimension as the global matrix, but corresponds only to the element k. This matrix would include the local matrix, in its part corresponding to that element, and would be zero in the rest of the elements, because the basis functions are defined nonzero only locally.

$$A^{(k)}c = b^{(k)}$$

Then the global system of linear equations would be just the sum of all the local systems.

$$\sum_k A^{(k)}c = \sum_k b^{(k)}$$

Now we can solve the global system to find the coefficients.

7.6 Usefulness of FEM

One of the advantages of FEM is that on the contrary to finite differences, the method does not define a fixed stepsize along the discretisation. One can modify the size of the elements in different parts of the grid, to have more detail on the computations on zones that are of more interest in the simulation and less detail in less interesting zones.

When this is the case, the element stiffness matrices will be different for elements with different sizes. To quantify and simplify this, one can define alternative variables ξ, η, and ζ which are scaled versions of x, y, and z with a size 1. This scales the stiffness matrix by a factor of $|\mathcal{J}|$, the determinant of the Jacobian:

$$|\mathcal{J}| = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{vmatrix}$$