CentraleSupélec

Numerical Methods

Ronan Vicquelin, Aymeric Vié

2025-2026

Contents

Part I

Course contents

Chapter 1

Introduction

Contents

1.1 Course objectives

Numerical simulations of physical phenomena have become inevitable. On the one hand, benefiting from computational resources in the 21st century, simulations are common practice. On the other hand, due to the increasing complexity and transdisciplinarity of practical engineering systems, no analytical solutions are available and the cost of experimental investigations becomes prohibitive. Therefore, engineers in charge of the design of such systems have no choice but to rely on numerical simulations.
The course objectives are:

1. Understanding standard numerical methods
2. Applying these methods in workshops
3. Critical analysis of simulations results

Combining their skills in computer science, heat transfer, fluid mechanics, mathematical analysis and numerical methods, the students will write their own programs from a blank page and answer several practical engineering problems such as:

1. Find the optimal residence time in a BioReactor.
2. Identify the value and location of the maximum temperature of a heated structure.
3. Control pollutant dispersion or flame flash-back.
4. Predict of recirculation length backwards a facing step
5. ...

On completion of the course, students will be able to: spontaneously solve a simple problem with a small script to implement a numerical resolution; formalize a physical problem into equations and identify their mathematical nature; discretize a set of differential equations; analyze the accuracy and stability of a numerical method; derive an adapted numerical method in terms of accuracy and efficiency to solve the problem; ensure the validity of the results though hypotheses checking and numerical errors characterization; have a critical interpretation of the physical results; solve problems found in engineering applications.

1.2 Course references

The theoretical contents of the course are based on a number of reference books listed below. Books which cover more advanced topics are also given.

Donea, J. and Huerta, A (2003). **Finite Elements Methods for Flow Problems**. Wiley.

Ferziger, J. H. and Peric, M. (2002) **Computational Methods for Fluid Dynamics**. Springer.

Hairer, E. and Wanner, G. (2000). **Solving Ordinary Differential Equations I**. Spinger; 2nd Edition.

Hairer, E. and Wanner, G. (2000). **Solving Ordinary Differential Equations II**. Spinger; 2nd Edition.

Hundsdorfer, W. and Verwer, J. (2003). **Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations**. Springer.

Hirsch, C. (2007) **Numerical computation of internal & external flows** (2nd Edition). Butterworth-Heinemann, Oxford.

Leveque, R. J. (2002) **Finite Volume Methods for Hyperbolic Problems**. Cambridge Texts in Applied Mathematics.

Moin, P. (2010) **Fundamentals of Engineering Numerical Analysis**. Cambridge University Press; 2nd Edition.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (2007) **Numerical Recipes. The Art of Scientific Computing**. Cambridge University Press; 3rd Edition.

Schäfer, M (2006). **Computational Engineering. Introduction to Numerical Methods**. Springer.

van der Vorst, H. A. (2003) **Iterative Krylov Methods for Large Linear Systems**. Cambridge University Press.

Saad, Y. (2003). **Iterative Methods for Sparse Linear Systems**. Society for Industrial and Applied Mathematics.

Toro, E. F. (2009) **Riemann Solvers And Numerical Methods for Fluid Dynamics: A Practical Introduction**. Springer.

1.3 Examples of differential equations

The mathematical expression of physical phenomena such as transfer of mass, momentum and energy yield partial derivative equations (PDEs). Several examples are listed below.
Laplace equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1.1}$$

Poisson equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = S \tag{1.2}$$

Unsteady heat equation:

$$\frac{\partial T}{\partial t} = \frac{\lambda}{\rho c_p}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \tag{1.3}$$

Stokes problem:

$$\frac{\partial u}{\partial t} = \mu \frac{\partial^2 u}{\partial y^2} \tag{1.4}$$

Transport/Advection:

$$\frac{\partial Y_k}{\partial t} + v_0 \frac{\partial Y_k}{\partial x} = 0 \tag{1.5}$$

Wave equation:

$$\frac{\partial^2 u}{\partial t} - a^2 \frac{\partial^2 u}{\partial x^2} = 0 \tag{1.6}$$

Ordinary differential equation:

$$\frac{\mathrm{d}Y_k}{\mathrm{d}t} = \frac{\dot{\omega}_k}{\rho} \tag{1.7}$$

Conservation equations for reactive multi-species gases:

$$\begin{cases} \dfrac{\partial}{\partial t}\rho + \dfrac{\partial}{\partial x_i}\rho u_i = 0 \\ \dfrac{\partial}{\partial t}\rho u_j + \dfrac{\partial}{\partial x_i}\rho u_i u_j = -\dfrac{\partial}{\partial x_j}P + \dfrac{\partial}{\partial x_j}\tau_{ij} + \rho g_j \\ \dfrac{\partial}{\partial t}(\rho E) + \dfrac{\partial}{\partial x_i}(\rho u_i E) = -\dfrac{\partial}{\partial x_i}(P u_i) + \dfrac{\partial}{\partial x_i}(\tau_{ij} u_j) + \rho g_j u_j - \dfrac{\partial}{\partial x_j} q_j + Q \\ \dfrac{\partial}{\partial t}\rho Y_k + \dfrac{\partial}{\partial x_i}\rho Y_k u_i = -\dfrac{\partial}{\partial x_i}\rho Y_k V_{k,i} + \dot{\omega}_k \end{cases} \tag{1.8}$$

where ρ is the density, u_j the j-th component of the velocity, E the energy (sensible and kinetic) and Y_k the mass fraction of species k.

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \mu \frac{\partial u_k}{\partial x_k} \delta_{ij} \tag{1.9}$$

$$q_j = -\lambda \frac{\partial T}{\partial x_j} + \sum_{k=1}^{N} \rho h_k V_{k,j} \tag{1.10}$$

$$Y_k V_{k,j} = -D \nabla X_k \tag{1.11}$$

General form of conservation equations:

$$\frac{\partial \overline{U}}{\partial t} + \nabla \overline{\overline{F}}(\overline{U}, \nabla \overline{U}) = \overline{S} \tag{1.12}$$

Temperature equation:

$$\rho c_p \frac{\partial T}{\partial t} + \rho c_p u_j \frac{\partial}{\partial x_j} T = \frac{\partial \rho}{\partial t} + u_j \frac{\partial P}{\partial x_j} + \sigma_{ij} \frac{\partial u_i}{\partial x_j} + Q - \frac{\partial q_i}{\partial x_i} - \sum_{k=1}^{N} h_k \dot{\omega}_k + \sum_{k=1}^{N} \rho Y_k V_{k,j} \frac{\partial h_k}{\partial x_j} \quad \text{+ Radiation} \tag{1.13}$$

1.4 Classification of differential equations

1.4.1 Classification of 1st order PDE

Let us consider the first-order PDE:

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} = S \tag{1.14}$$

where U is the vector of variables, $F(U)$ its flux and S a source term. If we linearise the system:

$$\frac{\partial U}{\partial t} + A \frac{\partial U}{\partial x} = S, \tag{1.15}$$

where $A = \partial_U F(U)$, the type of the PDE depends on the eigenvalues of A:

- if all eigenvalues are real, the system is hyperbolic
- if all eigenvalues are pure imaginary, the system is parabolic
- if some eigenvalues are complex and others are real, the system is hybrid

1.4.2 Classification of 2nd order PDE

Let us consider a second-order PDE:

$$\sum_{i,j} a_{ij} \frac{\partial^2 U}{\partial x_i \partial x_j} = 0 \tag{1.16}$$

The type of the PDE is linked to the type of the hyperplane:

$$\sum_{i,j} a_{ij} X_i X_j = 0 \tag{1.17}$$

The type of the equation depends on the eigenvalues of the coefficient matrix a_{ij}:

- if eigenvalues are all positive or all negative, the system is elliptic
- if eigenvalues are all positive or all negative except one that is zero, the system is parabolic
- if there is only one negative (resp. positive) eigenvalue and all the rest are positive (resp. negative), the system is hyperbolic

For example, let us consider the 2D equation:

$$A\frac{\partial^2 U}{\partial x^2} + B\frac{\partial^2 U}{\partial x \partial y} + C\frac{\partial^2 U}{\partial y^2} = 0 \tag{1.18}$$

The associated hyperplane is:

$$AX^2 + BXY + CY^2 = 0 \tag{1.19}$$

The type of the equation depends on $\Delta = B^2 - 4AC$:

- if $\Delta < 0$, the system is elliptic
- if $\Delta = 0$, the system is parabolic
- if $\Delta > 0$, the system is hyperbolic

1.4.3 Consistency of terminology for first and second order PDE's :

Example 1

Let us consider a first order PDE system with two unknown fields (u, v) defined by the matrix A as

$$A = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

This corresponds to the following equations

$$\begin{cases} \dfrac{\partial u}{\partial x} + \dfrac{\partial v}{\partial y} = 0 \\ \dfrac{\partial v}{\partial y} - \dfrac{\partial u}{\partial x} = 0 \end{cases} \tag{1.20}$$

Following the classification of 1^{st}-order PDE's, eigenvalues of the matrix A must be determined. Here, the pair of eigenvalues is $(1 + i, 1 - i)$ are both complex, denoting an elliptic system of 1^{st}-order PDE's. However, the previous system can be transformed into two 2^{nd}-order PDE as

$$\begin{aligned} \Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \\ \Delta v = \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = 0 \end{aligned} \tag{1.21}$$

The system becomes a set of uncoupled Laplace equations, which correspond to elliptic 2^{nd}-order PDE's. Both 1^{st}-order and 2^{nd}-order classification of partial derivative equations are indeed consistent.

Example 2

The set of equations governing acoustic waves in fluids is obtained by linearizing the Euler equations around a mean state of the flow at rest. The density ρ, the velocity u and the pressure p are written as

$$\begin{aligned} \rho &= \rho_0 + \rho_1 \\ u &= u_0 + u_1, \\ p &= p_0 + p_1 \end{aligned} \tag{1.22}$$

where ρ_0, $u_0 = 0$ and p_0 are the uniform conditions of the flow at rest. u_1, ρ_1 and p_1 are small perturbations such that $\rho_1 \ll \rho_0$ and $p_1 \ll p_0$. The resulting set of linearized Euler equations is

$$\frac{\partial \rho_1}{\partial t} + \rho_o \frac{\partial u_1}{\partial x} = 0 \tag{1.23}$$

$$\rho_0 \frac{\partial u_1}{\partial t} = -\frac{\partial p_1}{\partial x} = -c^2 \frac{\partial \rho_1}{\partial x}, \tag{1.24}$$

where $c = \left(\left. \frac{\partial p}{\partial \rho} \right|_s \right)^{1/2}$ is the speed of sound. Both equations can be put into the canonical form of a 1^{st}-order PDE as

$$\frac{\partial}{\partial t} \begin{bmatrix} \rho_1 \\ u_1 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & \rho_0 \\ \frac{c^2}{\rho_0} & 0 \end{bmatrix}}_{A} \frac{\partial}{\partial x} \begin{bmatrix} \rho_1 \\ u_1 \end{bmatrix} = 0 \tag{1.25}$$

$$\tag{1.26}$$

The eigenvalues of the matrix A are $(c, -c)$, both are real numbers. The PDE is then hyperbolic. Eliminating the velocity perturbation u_1, one can also express a 2^{nd}-order PDE for the density perturbation:

$$\frac{\partial^2 \rho_1}{\partial t^2} - c^2 \frac{\partial^2 \rho_1}{\partial x^2} = 0 \tag{1.27}$$

This is the classical wave equation which is an hyperbolic 2^{nd}-order PDE.

1.4.4 Discretisation of PDE

In general, PDE or systems of PDE do not have analytical solutions. In such cases, the only way to get the solution is to use approximations of the solution and the PDE system. The methods to approximate the solutions depend on the way the solution will be represented. There are three main types of approximations in the literature:

- Finite differences: the solution is evaluated at specific points in the domain.
- Finite Volumes: the solution is decomposed into a finite sum of control volumes.
- Finite Elements: this method is based on the variational formulation of the PDE system. The solution is represented into a set of basis functions

Chapter 2

Finite Differences

Contents

2.1 Principle of finite differences

In PDEs, the equations involve partial derivatives of functions such as $\frac{\partial f}{\partial x}$, $\frac{\partial^2 f}{\partial x^2}$. The discretized version of the PDE will then introduce discrete approximations of derivatives. In the finite difference approach, such approximations are built from the knowledge of the function values on a set of points on a structured mesh as depicted in Fig. 2.1. Numerical schemes based on finite differences

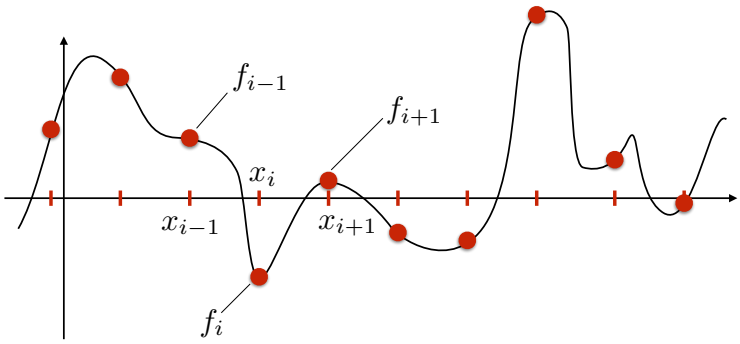


Figure 2.1 : *Continuous function approximated by its discrete values f_i, f_{i+1}, ... on the mesh composed of points x_i, x_{i+1}....*

approximate the function derivatives from the $f_i = f(x_i)$ values. This is done by combining Taylor series written as different points. The Taylor series expressed around x_i to estimate $f(x_{i+1})$ is given by

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + f''(x)\frac{1}{2}(x_{i+1} - x_i)^2 + f'''(x_i)\frac{1}{6}(x_{i+1} - x_i)^3 + ... \quad (2.1)$$

If one considers a uniform mesh with step size $h = x_{i+1} - x_i$, the Taylor series becomes:

$$f_{i+1} = f_i + hf_i' + \frac{h^2}{2}f_i'' + \frac{h^3}{6}f_i''' + ... \tag{2.2}$$

Such expression highlight the link between point values f_i and its derivatives f_i', f_i'' at the same location. This relationship is the key ingredient in an approach such as finite differences to approximate numerically the derivatives $\frac{\partial f}{\partial x}$, $\frac{\partial^2 f}{\partial x^2}$ involved in ODEs and PDEs.

2.2 Finite difference approximations of f_i'

If one is interested in an approximation of the first derivative $f'(x)$, the aforementioned Taylor series gives

$$f'(x_i) = f_i' = \frac{f_{i+1} - f_i}{h} \underbrace{-\frac{h}{2}f_i'' - \frac{h^2}{6}f_i''' + ...}_{\varepsilon = O(h)} \tag{2.3}$$

The first term on the right hand side is the numerical approximation of f_i' from f_i and f_{i+1}. The remaining terms constitute the **truncation error** ε and quantifies the difference between the real value $f'(x_i)$ and its numerical estimation. In the present example, this truncation error is dominated by the leading term for small values of h and one can write this asymptotic behavior as $\varepsilon = \mathcal{O}(h)$. This Big O notation indicates that for small values of h, the ratio ε/h is bounded, *i.e.* there exists a constant C such that $\varepsilon \leq Ch$.

In the general case where the truncation error is given by $\varepsilon = \mathcal{O}(h^\alpha)$, the obtained numerical approximation is said to be of α**-order accuracy**. High-order accurate formulas are interesting since, as h decreases, the difference between the real value of the derivative and its numerical approximation decreases as Ch^α, which yield small numerical errors as α increases. In the present example in Eq. (2.3), the obtained formula is then 1^{st}-order accurate and gives us an initial finite difference formula highlighted below

1^{st}-order Forward Difference Formula

$$f_i' = \frac{f_{i+1} - f_i}{h} + O(h) \tag{2.4}$$

Another formula can be obtained by writing the Taylor series around x_i to evaluate $f(x_{i-1})$:

$$f_{i-1} = f_i - hf_i' + \frac{h^2}{2}f_i'' - \frac{h^3}{6}f_i''' + ... \tag{2.5}$$

This yield the so-called *1^{st}-order backward* approximation of $f'(x)$:

1^{st}-order Backward Difference Formula

$$f_i' = \frac{f_i - f_{i-1}}{h} + O(h) \tag{2.6}$$

A higher order approximation can be achieved by combining the two aforementioned Taylor series formulas,

$$\begin{aligned} f_{i+1} &= f_i + hf'_i + \frac{h^2}{2}f''_i + \frac{h^3}{6}f'''_i + ... \\ f_{i-1} &= f_i - hf'_i + \frac{h^2}{2}f''_i - \frac{h^3}{6}f'''_i + ..., \end{aligned}$$

and by subtracting them, one gets

$$f_{i+1} + f_{i-1} = 2hf'_i + \frac{h^3}{3}f'''_i + ..., \tag{2.7}$$

or

$$f'_i = \frac{f_{i+1} - f_{i-1}}{2h} - \frac{h^2}{6}f'''_i + ... \tag{2.8}$$

The following second-order difference formula is obtained

2^{nd}-order Centered Difference Formula

$$f'_i = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2) \tag{2.9}$$

The formula is *centered* because computing f'_i requires the knowledge of neighbouring points on both sides $(i-1)$ and $(i+1)$. This formula is also a $2h$-stencil or 2Δ-stencil where Δ is another common notation for the mesh cell size. The stencil denotes the pattern of points used to approximate the quantity of interest. In 1D problems as investigated so far, the stencil outlines the spatial extension of the formula which depends on the considered points in the approximations. The forward and backward formula, in Eqs. (2.4) and (2.6) respectively, are 1Δ-stencil formulas.

By further combining Taylor series around x_i expressed at different points, it is possible to achieve finite difference formula of higher-order accuracy. Following such an approach, higher-order accurate formula will then rely on larger stencil.

Exercise 1

Demonstrate that the following centered formula is fourth-order accurate :

$$f'_i = \frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{12h} + O(h^4) \tag{2.10}$$

The treatment for boundary conditions is slightly particular, since, with the aforementioned approaches, points are only available at the inner side of the boundary. One is then compelled to choose either a:

- Lower-order formula with the same stencil
- Forward or Backward formula with a larger stencil to keep the same order of accuracy

The former approach will typically be retained by considering that the reduced accuracy at he boundaries will have a negligible impact on the global performance of the numerical scheme.

2.3 Finite difference approximations of f_i''

An approximation of the second order derivate $f''(x_i)$ is obtained by summing both Taylor series in x_{i+1} and x_{i-1}:

$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + \frac{h^2}{12} f_i'''' + ..., \tag{2.11}$$

which then yields a centered second-order formula highlighted below.

2^{nd}-order Centered Difference Formula

$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + O(h^2) \tag{2.12}$$

Similarly to formulas for the first-order derivative f_i', formulas of higher-order accuracy are built for f_i'' by involving more points. The opposite is not true: Obtaining a formula with a larger stencil does not necessarily gives a higher accuracy (see exercise below).
More points does not imply higher order!

Exercise 2

Let us consider an approximation of the second-order derivative f_i'' following a centered formula based on f':

$$f_i'' = \frac{f_{i+1}' - f_{i-1}'}{2h} + O(?) \tag{2.13}$$

Depending on the approximations chosen for f_{i+1}' and f_{i-1}', the final formula gives a different order of accuracy. Determine it for both choices detailed below.

1. a. 1^{st}-order forward difference formula for f_{i+1}' and 1^{st}-order backward difference formula for f_{i-1}'
2. b. 2^{nd}-order centered difference formula for both f_{i+1}' and f_{i-1}',

Solution 2

For both methods, the first centered formula based on f' gives

$$f_i'' = \frac{f_{i+1}' - f_{i-1}'}{2h} + O(h^2) = \frac{f_{i+1}' - f_{i-1}'}{2h} - \frac{h^2}{6} f_i^{(IV)} + O(h^4) \tag{2.14}$$

Method a
Two different formulas are used for f_{i+1}' and f_{i-1}':

$$1^{st}\text{-order forward difference formula: } f_{i+1}' = \frac{f_{i+2} - f_{i+1}}{h} - \frac{h}{2} f_{i+1}'' + O(h^2)$$

$$1^{st}\text{-order backward difference formula: } f_{i-1}' = \frac{f_{i-1} - f_{i-2}}{h} - \frac{h}{2} f_{i-1}'' + O(h^2)$$

Injecting both expressions in the formula for f_i'' gives

$$f_i'' = \frac{f_{i+2} - f_{i+1} - f_{i-1} + f_{i-2}}{2h^2} + \left(\frac{-\frac{h}{2}(f_{i+1}'' + f_{i-1}'') + O(h^2)}{2h} \right)$$

In the last term, one can write

$$f''_{i+1} = f''_i + O(h)$$
$$f''_{i-1} = f''_i + O(h)\text{,}$$

therefore the last term can be expressed as

$$\frac{1}{4h}[-2hf''_i + O(h)] = -\frac{1}{2}f''_i + O(h) = O(1)$$

Method a. then results in an inconsistant formula! Despite using correct difference formula, the undesired accumulation of truncation errors here give a useless formula although a larger stencil has been used.

$$\boxed{f''_i = \frac{f_{i+2} - f_{i+1} - f_{i-1} + f_{i-2}}{2h^2} + \underbrace{O(1)}_{inconsistant!}}$$

Method b

This time centered formula are used for both first-order derivatives:

$$f'_{i+1} = \frac{f_{i+2} - f_i}{2h} - \frac{h^2}{6}f'''_{i+1} + O(h^4) \tag{2.15}$$
$$f'_{i-1} = \frac{f_i - f_{i-2}}{2h} - \frac{h^2}{6}f'''_{i-1} + O(h^4) \tag{2.16}$$

Injecting both expressions in the formula for f''_i gives

$$f''_i = \frac{f_{i+2} - 2f_i + f_{i-2}}{4h^2} - \frac{h^2}{12h}(f'''_{i+1} - f'''_{i-1}) + O(h^3) - \frac{h^2}{6}f_i^{(IV)} + O(h^4) \tag{2.17}$$

Writing

$$f'''_{i+1} = f'''_i + hf_i^{(IV)} + O(h^2) \tag{2.18}$$
$$f'''_{i-1} = f'''_i - hf_i^{(IV)} + O(h^2), \tag{2.19}$$

one gets

$$\begin{aligned} f''_i &= \frac{f_{i+2} - 2f_i + f_{i-2}}{4h^2} - \frac{h}{12}(2hf_i^{(IV)} - O(h^2)) - \frac{h^2}{6}f_i^{(IV)} + O(h^3) \\ &= \frac{f_{i+2} - 2f_i + f_{i-2}}{4h^2} - \frac{h^2}{3}f_i^{(IV)} + O(h^3) \end{aligned}$$

Finally, the difference formula obtained through method b. is

$$\boxed{f''_i = \frac{f_{i+2} - 2f_i + f_{i-2}}{4h^2} + O(h^2)}, \tag{2.20}$$

which, despite having a larger stencil than Eq. (2.12), is still second-order accurate.

2.4 Optimal Difference Formula with Fixed Stencil

2.4.1 General principle

Using the Taylor series, a method is here presented to determine an optimal difference formula for a fixed stencil, *i.e* when fixing the points considered in the difference formula. The derivative at point x_j is approximated using points x_k from the interval $j-l \leq k \leq j+q$. Hence, we have l points on the left and q points on the right. Expressing the approximated derivative as a weighted sum of f_k, the purpose is to determine the weighted coefficients a_k to achieve the highest order accuracy:

$$f'_j + \sum_{k=-l}^{q} a_k f_{j+k} = O(h^?) \tag{2.21}$$

Once a_k coefficients are determined as well as the order α, one can write the obtained optimal formula

General Difference Formula

$$f'_j = -\sum_{k=-l}^{q} a_k f_{j+k} + O(h^\alpha) \tag{2.22}$$

Deriving for each point $x_k k$ the Taylor series around x_j gives the following general expression:

$$\begin{aligned} f'_j + \sum_{k=-l}^{q} a_k f_{j+k} \quad = \quad & \left(\sum_{k=-l}^{q} a_k\right) f_j + \left(1 + \sum_{k=-l}^{q} k a_k \frac{h}{1!}\right) f'_j + \left(\sum_{k=-l}^{q} k^2 a_k \frac{h^2}{2!}\right) f''_j \\ & + \left(\sum_{k=-l}^{q} k^3 a_k \frac{h^3}{3!}\right) f'''_j + ... \end{aligned} \tag{2.23}$$

Cancelling the right-hand-side terms enables us to determine the weighting coefficients. As more and more first terms on the right hand side are nullified, the leading term becomes of higher and higher order. The number of terms to set to zero is set to provide enough equations to determine the a_k factors. Considering more points in the difference formula will then automatically increase the desired order accuracy by following this methodology. The problem becomes the determination a_k such that equation (2.21) produces the highest possible order. The equation (2.23) is usually summed up in a **Taylor table** for the sake of clarity. One such Taylor table is presented in Tab. 2.1 where each term from the left-hand-side of Eq. (2.22) is developed in terms of derivatives in x_j: f_j, f'_j, f''_j, ...

Table 2.1 : *Taylor table representation of a general difference formula*

	f_j	f'_j	f''_j	$\cdots$	$f_j^{(n)}$	$\cdots$
f'_j	0	1	0	$\cdots$	0	$\cdots$
f_j	1	0	0	$\cdots$	0	$\cdots$
$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$
$a_1 f_{j+1}$	a_1	$a_1 h$	$a_1 \frac{h^2}{2!}$	$\cdots$	$a_1 \frac{h^n}{n!}$	$\cdots$
$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$
$a_k f_{j+k}$	a_k	$k a_k \frac{h^1}{1!}$	$k a_k \frac{h^2}{2!}$	$\cdots$	$k a_k \frac{h^n}{n!}$	$\cdots$
$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$

2.4.2 Example

An example is detailed for the first-order derivative by determining an optimal forward difference formula using two neighbouring points ($l = 0$ and $q = 2$):

$$f'_j + \sum_{k=0}^{2} a_k f_{j+k} = O(?) \tag{2.24}$$

The corresponding Taylor table is given in Tab. 2.2.

Table 2.2 : *Taylor table for the 2Δ forward formula*

	f_j	f'_j	f''_j	f'''_j
f'_j	0	1	0	0
$a_0 f_j$	a_0	0	0	0
$a_1 f_{j+1}$	a_1	$a_1 h$	$a_1 \frac{h^2}{2}$	$a_1 \frac{h^3}{6}$
$a_2 f_{j+2}$	a_2	$a_2 2h$	$a_2 2h^2$	$\frac{4}{3} a_2 h^3$

Hence, injecting Taylor series in the formula gives

$$\begin{aligned} f'_j + \sum_{k=0}^{2} a_k f_{j+k} &= (a_0 + a_1 + a_2) f_j + (a_1 h + 2a_2 h) f'_j + (a_1 \frac{h^2}{2} + 2a_2 h^2) f''_j \\ &\quad + (a_1 \frac{h^3}{6} + \frac{4}{3} a_2 h^3) f'''_j + ... \end{aligned} \tag{2.25}$$

Cancelling the three first terms on the right hand side of the equation enables us to fix the through weighting coefficients a_0, a_1 and a_2 through a linear system:

$$\begin{cases} a_0 + a_1 + a_2 = 0 \\ a_1 h + 2a_2 h = 1 \\ a_1 \frac{h^2}{2} + 2a_2 h^2 = 0 \end{cases} \rightarrow \begin{cases} a_1 = -4h \\ a_2 = \frac{1}{2} h \\ a_3 = \frac{3}{2} h \end{cases} \tag{2.26}$$

Putting back these results into the formula for f'_j without forgetting to estimate the truncation error term to determine the order of accuracy finally gives

$$f'_j = \frac{-3f_j + 4f_{j+1} - f_{j+2}}{2h} + \frac{h^2}{3} + ... \tag{2.27}$$

The obtained formula is therefore second order.

2^{nd}-order Forward Difference Formula

$$f'_j = \frac{-3f_j + 4f_{j+1} - f_{j+2}}{2h} + O(h^2) \tag{2.28}$$

The same methodology can be applied to derive optimal finite difference formula for the second-order derivative f''_j.

Exercise 3

Determine a centered fourth-order difference formula for f''_j on a 4Δ-stencil.

2.5 Accuracy analysis with the modified wavenumber

2.5.1 Principle

The order accuracy is not the only metric to appreciate differences between different numerical schemes, here difference formulas. Besides, order accuracy only gives insight in the numerical approximation behavior for small values of the cell size h. The approach detailed here studies the impact of the numerical discretization on a specific kind of functions: $f(x) = e^{ikx}$. The relevance of limiting our study to this family of functions comes from the discrete-time Fourier transform that will not be detailed here.

When considering the harmonic function $f(x) = e^{ikx}$ of wavenumber k and wavelength $\lambda = \dfrac{2\pi}{k}$, the numerical approximation will not impact similarly the large and well-resolved wavelengths and the small and coarsely captured wavelengths. Quantifying thoroughly this effect is the **modified wavenumber analysis**. The principle is the following one. The derivative of the considered continuous function is well known:

$$f(x) = e^{ikx} \Rightarrow f'(x) = ike^{ikx} = ik\, f(k) \tag{2.29}$$

Thus, evaluated at the point x_j, the exact derivative is easily determined

$$f'_j = ikf_j \tag{2.30}$$

The retained difference formula will give a different result because of the numerical error. Nonetheless, the obtained approximation can always be put in the form

$$f'_j = ik'f_j, \tag{2.31}$$

where k' is the modified wavenumber. The modified wavenumber analysis consists then in comparing k and k' for different reference wavenumber values k. This is usually represented in plots of $hk' = f(hk)$. When both wavenumbers are identical, the approximation is perfect for all wavelengths and there is no numerical error. This case is only met when using *spectral schemes*, not detailed here, where Eq. (2.30) is enforced through discrete Fourier transforms. In general and for the difference formulas seen before, k and k' are different.

2.5.2 Examples of modified wavenumber analysis

First-order derivative approximations

Let us determine the modified wavenumber k' associated to the 2^{nd}-order centered difference formula for f'_j:

$$f'_i = \frac{f_{i+1} - f_{i-1}}{2h}.$$

With the considered harmonic function $f(x) = e^{ikx}$, one finds that $f_j = e^{ikx_j} = e^{ikjh}$ since, without loss of generality, $x_j = jh$. The difference formula then becomes

$$f'_j = \frac{e^{ik(j+1)h} - e^{ik(j-1)h}}{2h} = \underbrace{e^{ikjh}}_{f_j} \frac{e^{ikh} - e^{-ikh}}{2h} \tag{2.32}$$

Hence,

$$f'_j = if_j \frac{\sin(hk)}{h} = ik'f_j, \tag{2.33}$$

where the modified wavenumber of the considered scheme is given by

$$\boxed{hk' = \sin(hk)} \tag{2.34}$$

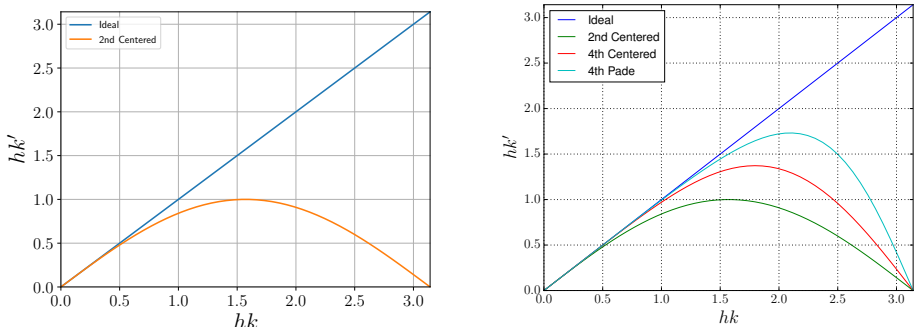


Figure 2.2 : *Plots of modified wavenumbers. a) 2^{nd}-order centered formula from Eq.* (2.9) *compared to ideal curve. b) Comparison between ideal curve, 2^{nd}-order centered formula from Eq.* (2.9)*, 4^{th}-order centered formula from Eq.* (2.10) *and 4^{th}-order Padé formula*

The comparison of both wavenumbers is shown in Fig. 2.2 a). The abscissa axis range from 0 to π. Infinitely long wavelengths correspond to $k = 0$, while, following the Shannon criterion, the smallest wavelength captured on the discretized mesh is $\lambda = 2h$ that corresponds to $hk = \pi$. The figure shows that the investigated 2^{nd}-order centered difference formula is accurate for small wavenumbers as expected given that the difference formula becomes more and more accurate as more points are retained in one wavelength. One can demonstrate that the departure of the hk' wavenumber curve from the ideal one is directly linked to the order accuracy of the scheme.
Figure 2.2 b) compares the modified wavenumbers of different approximations for f'_j. The modified wavenumber analysis enables us to clearly compare in quantitative terms several schemes of different or same orders. While being of the same order as the centered difference formula in Eq. (2.10), the presented Padé scheme captures much more accurately the harmonic function on a coarse mesh. This highlights that the order of accuracy is certainly not the pinnacle metric.

Exercise 4

Show that the modified wavenumber associated to the 4^{th}-order centered formula from Eq. (2.10) is given by

$$hk' = \frac{3\sin(hk)}{2+\cos(hk)} \tag{2.35}$$

Second-order derivative approximations

Difference formula for f''_j are also characterized in terms of modified wavenumbers. The second-order derivative of the continuous harmonic function is

$$f''(x) = -k^2 f(x) \Rightarrow f''j = -k^2 f_j \tag{2.36}$$

Correspondingly, the numerical approximation introduces a modified wavenumber k' fulfilling

$$f''j = -k'^2 f_j \tag{2.37}$$

Different difference formulas are compared in Fig. 2.3. While of same order, the 4Δ centered formula of second order is seen to be quite inaccurate compared to the classical second order approximation given in Eq. (2.12).

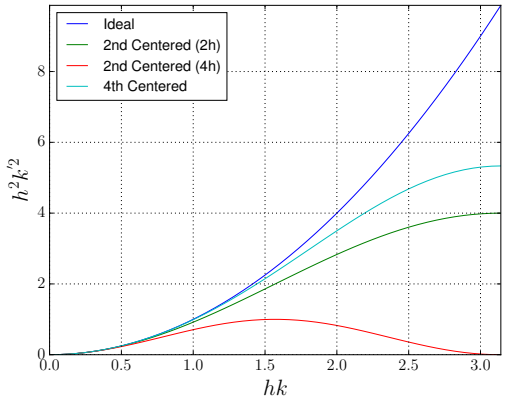


Figure 2.3 : *Comparison of modified wavenumbers for different approximations of f''_j: 2^{nd}-order centered formula from Eq.* (2.12)*, 2^{nd}-order centered formula from Eq.* (2.20) *with a larger stencil, and the 4^{th}-order centered formula derived in a previous exercise*

Exercise 5

- Determine the modified wavenumber for the 2^{nd}-order centered formula in Eq. (2.12)
- Same for the 4Δ 2^{nd}-order formula in Eq. (2.20)
- Same for 4^{th}-order centered formula derived in a previous exercise

2.6 Padé schemes

TODO

Cheat Sheet: Finite Difference Formulas

- **Finite difference formula for** $f'_j = \left.\frac{df}{dx}\right|_{x_j}$
  - 1^{st}-order Forward Difference Formula
$$f'_i = \frac{f_{i+1} - f_i}{\Delta x}$$
  - 1^{st}-order Backward Difference Formula
$$f'_i = \frac{f_i - f_{i-1}}{\Delta x}$$
  - 2^{nd}-order Centered Difference Formula
$$f'_i = \frac{f_{i+1} - f_{i-1}}{2\Delta x}$$
  - 2^{nd}-order Forward Difference Formula
$$f'_j = \frac{-3f_j + 4f_{j+1} - f_{j+2}}{2\Delta x}$$
  - 2^{nd}-order Backward Difference Formula
$$f'_j = \frac{3f_j - 4f_{j-1} + f_{j-2}}{2\Delta x}$$
  - 4^{th}-order Centered Difference Formula
$$f'_i = \frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{12\Delta x}$$
- **Finite difference formula for** $f''_j = \left.\frac{d^2f}{dx^2}\right|_{x_j}$
  - 2^{nd}-order Centered Difference Formula
$$f''_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2}$$
  - 4^{th}-order Centered Difference Formula
$$f''_i = \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i-1} - j_{i-2}}{12\Delta x^2}$$

Chapter 3

Ordinary Differential Equations

Contents

We consider the following general initial value problem:

$$\begin{cases} \dfrac{d\boldsymbol{Y}}{dt} = \boldsymbol{F}(\boldsymbol{Y}, t) \qquad \text{with } \boldsymbol{Y} = (y^1, ..., y^m)^T; \boldsymbol{F} = (f^1, ..., f^m)^T \\ \boldsymbol{Y}(t=0) = \boldsymbol{Y}_0, \end{cases} \tag{3.1}$$

each y^p and F^p, $p \in [\![1, m]\!]$ being regular functions.
The scalar version of this system of equations reads:

$$\begin{cases} \dfrac{dy}{dt} = f(y, t) \\ y(t=0) = y_0. \end{cases} \tag{3.2}$$

To solve such problems, a common way is to use time-marching methods. This class of methods consists in solving for $y(t)$ step-by-step, as:

- for a given $y_n = y(t_n)$, find $y_{n+1} = y(t_{n+1})$,
- repeat the process until the desired time t is reached.

3.1 Forward Euler method

In order to estimate the update y_{n+1} from a given state $y_n = y(t_n)$, one can use a Taylor series expansion:

$$y_{n+1} = y_n + hy'_n + \frac{h^2}{2}y''_n + \frac{h^3}{6}y'''_n + ... \,, \tag{3.3}$$

with $h = t_{n+1} - t_n$ a **constant** time-step and $y'_n = \left.\frac{dy}{dt}\right|_{t=t_n} = f(y_n, t_n)$. Therefore, the first-order truncation of Eq (3.3) yields

$$y_{n+1} = y_n + hy'_n + \mathcal{O}(h^2). \tag{3.4}$$

This defines the Forward Euler method, also simply known as the Euler methods.

Forward Euler Method

$$y_{n+1} = y_n + hf(y_n, t_n) \tag{3.5}$$

This method is second-order accurate **locally**, which means it is second-order accurate over one step. Alternatively to a local error accuracy,

$$\varepsilon_{loc} = \|y^{\text{num}}(t_{n+1}) - y^{\text{exact}}(t_n)\|, \tag{3.6}$$

one usually prefers to determine the **global error** accuracy after a finite time T as

$$\varepsilon_{glob} = \|y^{\text{num}}(t = T) - y^{\text{exact}}(t = T)\|. \tag{3.7}$$

For a fixed time T, decreasing the time-step h increases the number K of time-steps to be computed, as $T = Kh$. Thus, ε_{loc} and ε_{glob} differ from each other because of cumulative errors in the global accuracy. It can be shown that, **while the Forward Euler method is second-order accurate locally, it is first-order accurate locally**.

Demonstration

Decomposing the solution at time T, one gets

$$y(T = Kh) = y_K = (y_K - y_{K-1}) + (y_{K-1} - y_{K-2}) + ... + (y_1 - y_0) + y_0 \tag{3.8}$$

$$= \sum_{n=0}^{K-1} (y_{n+1} - y_n) + y_0 \tag{3.9}$$

$$= \sum_{n=0}^{K-1} \left[hf(y_n, t_n) + \frac{h^2}{2} y''_n + ... \right] + y_0 \tag{3.10}$$

$$= \underbrace{y_0 + \sum_{n=0}^{K-1} hf(y_n, t_n)}_{\text{Forward Euler}} + \underbrace{\frac{h^2}{2} \sum_{n=0}^{K-1} y''_n + ...}_{\text{Numerical Error}} . \tag{3.11}$$

For a fixed time T, as h decreases, the number of steps $K = T/h$ increases. The term $\sum_{n=0}^{K-1} y''_n$ is then expected to increase. This effect is quantified precisely by using a discrete version of the *Mean value theorem* which states that there is a value $E \in [0, T]$ such that

$$\sum_{n=0}^{K-1} y''_n = Ky'''(E). \tag{3.12}$$

Therefore,

$$y_K = y_0 + h \sum_{n=0}^{K-1} f(y_n, t_n) + \frac{h^2}{2} K y_n'''(E) + ... \tag{3.13}$$

$$= \underbrace{y_0 + h \sum_{n=0}^{K-1} f(y_n, t_n)}_{\text{Forward Euler}} + \underbrace{\frac{hT}{2} y_n'''(E) + ...}_{\text{Error: } \mathcal{O}(h)} \tag{3.14}$$

The Forward Euler method is then indeed first-order accurate globally.

More generally, one can demonstrate similarly that **if a time-marching method is $(\alpha + 1)$-order accurate locally, then it is α-order accurate globally**.

Remark

The global error being a more practical metric of accuracy, if not mentioned, global error is the one that is meant when speaking of order accuracy.

3.2 Stability analysis

Two families of methods are distinguished to solve ODEs: implicit and explicit methods.

Explicit Methods: The update for explicit methods can be expressed as:

$$y_{n+1} = \text{known R.H.S.} \tag{3.15}$$

For explicit methods, the formula explicitly gives the computation of y_{n+1} to be implemented. **Forward Euler is an explicit integration method.**

Implicit Methods: On the contrary, for implicit methods, no explicit formula is available to compute y_{n+1}. For instance, the right-hand-side in the update formula might contain $f(y_{n+1}, t_{n+1})$ which depends on the value to determine itself. Implicit methods then imply an algebraic equation to be solved with a root finding algorithm. They are then more expansive than explicit methods but **implicit numerical schemes are usually characterized by a desired enhanced numerical stability**.

Numerical Stability: Numerical stability is defined as follows:

Definition of numerical stability

If $y(t)$, continuous solution of (3.2), is bounded for $t \leq t_0$, the numerical scheme is stable if the discrete numerical solution also remains bounded for $n \leq n_0$.

Stability and accuracy are different concepts. One can derive an accurate time-marching method to solve the initial value problem but, if it is not stable, the computed iterates will quickly diverge from the exact solution, which makes the method useless. Consequently, a numerical method *should* be **accurate** but it definitely *must* be **stable**. Three types of schemes can be found:

- Unconditionally stable: the numerical solution remains bounded whatever the numerical parameters (h, ...),

- Unstable: the numerical solution diverges whatever the parameters. Such numerical methods are then useless,
- Conditionally stable: the numerical solution remains bounded for specific values/ranges of parameters.

Stability Analysis: The stability analysis of a numerical method can be performed by studying the numerical stability of the method when applied to the following scalar and linear ODE :

$$y' = \lambda y. \tag{3.16}$$

Demonstration

Using a first-order Taylor expansion, one gets:

$$\frac{d\boldsymbol{Y}}{dt} = \boldsymbol{F}(\boldsymbol{Y},t) = \boldsymbol{F}(\boldsymbol{Y}_0,t_0) + \left.\frac{\partial \boldsymbol{F}}{\partial t}\right|_{t_0,\boldsymbol{Y}_0}(t-t_0) + \underbrace{\left.\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{Y}}\right|_{t_0,\boldsymbol{Y}_0}}_{\boldsymbol{J}_0}(\boldsymbol{Y}-\boldsymbol{Y}_0), \tag{3.17}$$

where $vecJ_0$ is the JAcobian matrix of the r.h.s. function at the considered instant. The departure of the state vector $\boldsymbol{Y}$ from its initial value is then given by

$$\frac{d(\boldsymbol{Y}-\boldsymbol{Y_0})}{dt} = \underbrace{\boldsymbol{J}_0(\boldsymbol{Y}-\boldsymbol{Y_0})}_{\text{critical}} + \underbrace{\boldsymbol{F}(t_0,\boldsymbol{Y_0}) + \left.\frac{\partial \boldsymbol{F}}{\partial t}\right|_{t_0,\boldsymbol{Y}_0}(t-t_0)}_{\text{non-critical}}. \tag{3.18}$$

Only the first term on the right -hand side is critical regarding stability. If the Jacobian matrix $\boldsymbol{J}_0$ is diagonalizable,

$$J_0 = \boldsymbol{P}\begin{pmatrix}\lambda_1 & & 0\\ & \ddots & \\ 0 & & \lambda_m\end{pmatrix}\boldsymbol{P}^{-1}, \tag{3.19}$$

one can introduce the following change of variables

$$\boldsymbol{U} = \boldsymbol{P}^{-1}(\boldsymbol{y}-\boldsymbol{y}_0). \tag{3.20}$$

Th original system of non-linear ODEs is then transformed after linearization into a set of uncoupled scalar ODEs:

$$\frac{dU_i}{dt} = \lambda_i U_i. \tag{3.21}$$

Then, studying a method stability through equation (3.16) for any value of λ allows to generalize the result to any operator $\boldsymbol{F}$ in equation (3.1), as long as its Jacobian matrix is diagonalizable.

Remark

If the Jacobian matrix is not diagonalizable, the same conclusion stands with Jordan decomposition.

The general stability characterization of a numerical scheme then considers equation (3.16), with a complex number λ. For this simple case, the exact solution is known and is given by:

$$y(t) = y_0 e^{\lambda t}. \tag{3.22}$$

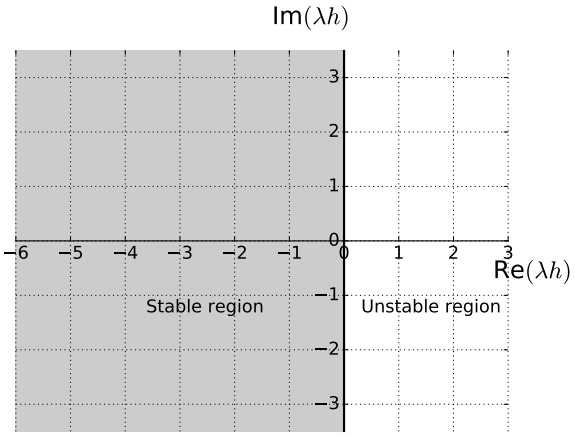


Figure 3.1 : *Stability region of the exact solution*

This solution remains bounded as long as the following condition is satisfied:

$$Re(\lambda) \leq 0, \tag{3.23}$$

where $Re(\lambda)$ is the real part of the complex number. The corresponding stability region in the complex plane $(Re(\lambda h), Im(\lambda h))$ is the half-space shown in gray in Fig. 3.1.

Stability of the Forward Euler method: The Forward Euler method,

$$y_{n+1} = y_n + hf(y_n, t_n),$$

applied to the ODE

$$y' = \lambda y \tag{3.24}$$

gives the following updating formula for y_{n+1}:

$$y_{n+1} = y_n + h\lambda y_n \tag{3.25}$$
$$= (1 + h\lambda)y_n. \tag{3.26}$$

Introducing the **amplification factor**,

$$\sigma = (1 + h\lambda), \tag{3.27}$$

the updating formula reads

$$\boxed{y_{n+1} = \sigma y_n.} \tag{3.28}$$

One has

$$y_n = y_0\sigma^n, \tag{3.29}$$

which is bounded if and only if

$$|\sigma| \leq 1. \tag{3.30}$$

Decomposing λ as $\lambda = \lambda_R + i\lambda_I$, one gets:

$$|\sigma|^2 = (1 + h\lambda_R)^2 + (h\lambda_I)^2 \tag{3.31}$$

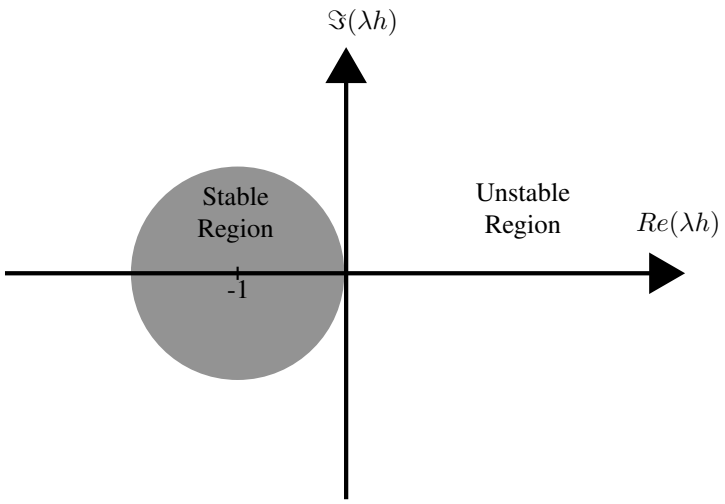


Figure 3.2 : *Stability region of the Forward Euler Method*

Condition $\sigma \leq 1$ corresponds then to the disc of radius 1 and center (-1, 0) which is shown in Fig. 3.2. The **forward Euler method is then conditionally stable**: depending on the problem, the time step h must be small enough to secure the stability of the numerical method *i.e.* $|\sigma| \leq 1$. For large time steps, the method diverges. Summing up the properties of the first time-marching method seen so far, we have

Properties of the Forward Euler Method

- 1^{st}-order accurate globally
- Conditionally stable (see Fig. 3.2)
- If the eigenvalue λ is real and negative ($\lambda \in \mathbb{R}^-$) $\Longrightarrow$ Stable if $h \leq \frac{2}{|\lambda|}$
- If the eigenvalue λ is purely imaginary ($\lambda \in i\mathbb{R}$) $\Longrightarrow$ Unstable

3.3 Accuracy analysis with the amplification factor σ

Each numerical scheme is characterized by a different amplication factor formula. The knowledge of σ can also determine the accuracy of the scheme which is explained in this section. Considering the linear scalar ODE, $y' = \lambda y$, the exact solution at time t_{n+1} from a given state t_n is known:

$$y_{n+1}^{\text{exact}} = y_n e^{\lambda h}. \tag{3.32}$$

For the same ODE, the numerically predicted value is given by the amplification factor:

$$y_{n+1} = \sigma y_n. \tag{3.33}$$

Then,

$$y_{n+1}^{\text{exact}} - y_{n+1} = y_n \left(e^{\lambda h} - \sigma\right). \tag{3.34}$$

A power series expansion yields:

$$e^{\lambda h} = 1 + \lambda h + \frac{(\lambda h)^2}{2} + \frac{(\lambda h)^3}{6} + ... = \sum_{k=0}^{+\infty} \mu_k (\lambda h)^k, \tag{3.35}$$

and one can write similarly

$$\sigma(\lambda h) = \sum_{k=0}^{+\infty} \nu_k (\lambda h)^k, \tag{3.36}$$

which, for the example of Forward Euler Method, reads $\sigma = 1 + \lambda h$. Then the accuracy can be evaluated from:

$$y_{n+1}^{\text{exact}} - y_{n+1} = y_n \sum_{k=0}^{+\infty} (\mu_k - \nu_k)(\lambda h)^k \tag{3.37}$$

Accuracy analysis from σ

The method is of order α (**globally**) if $\mu_k = \nu_k$ for $k = 0, ..., \alpha$

The fact that the forward Euler method global order is 1 is retrieved.

3.4 Backward Euler Method

The formula of the Backward Euler Method is given below

Backward Euler Method

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}). \tag{3.38}$$

Since the right-hand-side depends on the value y_{n+1} itself, the defined numerical method is **implicit**.

3.4.1 Accuracy Analysis:

Let us characterize the method accuracy by following two different methods. Both will show **the Backward Euler Method is 1^{st}-order accurate (globally)**.

Method 1: Taylor series The Taylor series expansion around t_{n+1} reads

$$y_n = y_{n+1} - hy'_{n+1} + \frac{h^2}{2} y''_{n+1} + ... \tag{3.39}$$

$$= y_{n+1} - hf(y_{n+1}, t_{n+1}) + \frac{h^2}{2} y''_{n+1} + ... \tag{3.40}$$

$$\tag{3.41}$$

Hence

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}) - \underbrace{\frac{h^2}{2} y''_{n+1} + ...}_{\mathcal{O}(h^2)}, \tag{3.42}$$

The backward Euler method is second-order *locally* and then first-order *globally*.

Method 2: Amplification factor Following Sec. 3.3, let us now compute the amplification factor of the method. For the particular ODE, $y' = \lambda y$, one finds that

$$y_{n+1} = y_n + \lambda h y_{n+1} \tag{3.43}$$

$$y_{n+1} = \frac{1}{1-\lambda h} y_n. \tag{3.44}$$

Then,

$$\boxed{\sigma = \frac{1}{1-\lambda h}} = \underbrace{1+\lambda h}_{1^{\text{st}}\text{-order}} + (\lambda h)^2 + ... \tag{3.45}$$

Only the first two terms match the Taylor series expansion of the exponential in Eq. 3.35, the backward Euler method is then first-order accurate *globally*.

3.4.2 Stability Analysis:

We consider equation $y' = \lambda y$ with $Re(\lambda) \leq 0$. The stability condition reads

$$|\sigma|^2 \leq 1, \tag{3.46}$$

which gives

$$\frac{1}{|\sigma|^2} \geq 1 \tag{3.47}$$

and eventually

$$(1-\lambda_R h)^2 + (\lambda_I h)^2 \geq 1. \tag{3.48}$$

This region is the complementary region of the disc of center $(1, 0)$ and radius 1. The stability constraint is therefore always fulfilled in the half-space $\lambda_R < 0$. The Backwork Euler Method is then always stable.

Properties of the Backward Euler Method

The backward Euler Method is a **1st-order implicit** method which is **inconditionnaly stable**

Remark

- *The advantage of implicit methods is their enhanced stability.*
- *Actually, there exist implicit methods with a limited stablity: they are not used, since the additional cost they suffer is not worth it.*
- *Stability and accuracy are two different concepts characterizing numerical schemes. Stability **does not** imply accuracy, and vice-versa.*

3.5 Trapezoidal method

From equation (3.2), one can write:

$$y(t) = y_n + \int_{t_n}^{t} f(y, \tau) d\tau, \tag{3.49}$$

in particular,

$$y(t_{n+1}) = y_n + \int_{t_n}^{t_{n+1}} f(y,\tau)d\tau. \tag{3.50}$$

This formula is always valid and can be estimated with different quadrature rule. When using the trapezoidal quadrature rule, one obtains the Trapezoidal ODE method:

Trapezoidal Method

$$y_{n+1} = y_n + \frac{h}{2}\Big(f(y_n,t_n) + f(y_{n+1},t_{n+1})\Big) \tag{3.51}$$

According to the time-marching formula, the **Trapezoidal method is implicit**.

Amplification Factor: The amplification factor is evaluated through equation (3.16), which yields

$$y_{n+1} = y_n + \frac{h\lambda}{2}\big(y_n + y_{n+1}\big) \tag{3.52}$$

$$\left(1 - \frac{h\lambda}{2}\right) = \left(1 + \frac{h\lambda}{2}\right) y_n, \tag{3.53}$$

hence

$$y_{n+1} = \sigma y_n \tag{3.54}$$

with the amplification factor

$$\boxed{\sigma = \frac{1 + \frac{\lambda h}{2}}{1 - \frac{\lambda h}{2}}} \tag{3.55}$$

Accuracy Analysis: Using Taylor series expansion:

$$\sigma = \left(1 + \frac{\lambda h}{2}\right)\left(1 + \frac{\lambda h}{2} + \left(\frac{\lambda h}{2}\right)^2 + \left(\frac{\lambda h}{2}\right)^3 + ...\right) \tag{3.56}$$

$$= 1 + \lambda h + \underbrace{\frac{(\lambda h)^2}{2}}_{2^{\text{nd}}\text{-order}} + \frac{(\lambda h)^3}{4} + ... \tag{3.57}$$

The **Trapezoidal method is 2^{nd}-order accurate (globaly).**

Stability Analysis: Stability condition is expressed through the amplification factor σ as:

$$|\sigma|^2 \leq 1, \tag{3.58}$$

hence

$$\left|1 + \frac{\lambda h}{2}\right| \leq \left|1 - \frac{\lambda h}{2}\right| \tag{3.59}$$

which is always true for $Re(\lambda) \leq 0$. The **Trapezoidal Method is thus inconditionally stable**.

Properties of the Trapezoidal Method

- Implicit,
- 2^{nd}-order accurate,
- Inconditionally stable.

Remark

- *Solving the implicit formula of the Trapezoidal method is as costly as the Backward Euler method.*
- *Trapezoidal Method is ill-behaved for $Re(\lambda h) \to +\infty$, i.e. $h \to +\infty$, which corresponds to $\sigma \to -1$ and results in an oscillatory behavior.*
- *When applied to parabolic PDE, this method is called the* Crank-Nicholson Method.

3.6 Linearization of implicit methods

Solving algebraic equations (*e.g.* with Newton Solvers) appearing in implicit methods is expensive, especially when dealing with vectorial systems. A common practice is to linearize the right-hand-side function in order to replace the algebraic equations by a linear system of equations. Solving the system is then much less expensive. As long as the linearization is valid for moderate (case-dependent) h values, the linearized numerical scheme keeps the same accuracy and stability properties.

Example : Let's consider the trapezoidal method on a scalar ODE

$$y_{n+1} = y_n + \frac{h}{2}\Big(f(y_n, t_n) + f(y_{n+1}, t_{n+1})\Big) \tag{3.60}$$

$f(y_{n+1}, t_{n+1})$ is linearized without losing the order of accuracy.

$$f(y_{n+1}, t_{n+1}) = f(y_n, t_{n+1}) + \left.\frac{\partial f}{\partial y}\right|_{y_n, t_{n+1}} (y_{n+1} - y_n) + \frac{1}{2}\left.\frac{\partial^2 f}{\partial y^2}\right|_{y_n, t_{n+1}} (y_{n+1} - y_n)^2 + ... \tag{3.61}$$

where

$$y_{n+1} - y_n = \mathcal{O}(h). \tag{3.62}$$

Substituing (3.61) into (3.60) yields

$$y_{n+1} = y_n + \frac{h}{2}\left[f(y_n, t_n) + \left.\frac{\partial f}{\partial y}\right|_{y_n, t_{n+1}} (y_{n+1} - y_n) + f(y_n, t_{n+1})\right] + \mathcal{O}(h^3) \tag{3.63}$$

hence

$$\boxed{y_{n+1} = y_n + \frac{h}{2}\frac{f(y_n, t_n) + f(y_n, t_{n+1})}{1 - \frac{h}{2}\left.\frac{\partial f}{\partial y}\right|_{y_n, t_{n+1}}} + \mathcal{O}(h^3)} \tag{3.64}$$

This method is then explicit and offers the same accuracy and stability as its corresponding implicit method as long as Taylor series (3.61) remains valid.

Remark

In the case of a non-linear system of ODEs, the linearization involves the Jacobian matrix $\boldsymbol{J} = \left.\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{Y}}\right|_{\boldsymbol{Y}_n, t_{n+1}}$ and the denominator in Eq. (3.64) *becomes $\left(1 - \frac{h}{2}\boldsymbol{J}\right)$ which requires to solve a linear system.*

3.7 Stiffness

A specific phenomenon is taking place in vectorial systems: **Stiffness**. The ODE reads

$$\frac{d\boldsymbol{Y}}{dt} = \boldsymbol{F}(\boldsymbol{Y}, t) \tag{3.65}$$

Stability is studied by considering a linear system, which in the case of a systems of ODEs, involves a matrix:

$$\boldsymbol{Y}' = \boldsymbol{A}\boldsymbol{Y}, \tag{3.66}$$

Let's illustrate the phenomenon with the Forward Euler Method given here by

$$\boldsymbol{Y}_{n+1} = \boldsymbol{Y}_n + h\boldsymbol{A}\boldsymbol{Y}_n, \tag{3.67}$$

i.e.

$$\boldsymbol{Y}_n = (\boldsymbol{I}_d + h\boldsymbol{A})^n \boldsymbol{Y}_0. \tag{3.68}$$

This remains bounded as long as the eigenvalues α_i of matrix $(\boldsymbol{I}_d + h\boldsymbol{A})$ verify the condition $\forall\ i \in [\![1, m]\!], |\alpha_i| \leq 1$. Note that the α_i values are related to the eigenvalues $(\lambda_i)_{i\in[\![1,m]\!]}$ of matrix $\boldsymbol{A}$:

$$\alpha_i = 1 + h\lambda_i. \tag{3.69}$$

Therefore, the stability constraint must be verified by all eigenvalues as

$$|1 + h\lambda_i| \leq 1, \quad \forall\ i \in [\![1, m]\!] \tag{3.70}$$

The eigenvalues fo the largest magnitude yields the most stringent constraint. The spectral radius, defined as $|\lambda|_{max}$, determines then the stability. If all eigenvalues are real, one finds that the condition $h \leq \frac{2}{|\lambda|_{\text{max}}}$ enforces stability in the forward Euler method. While the fast dynamics of $y(t)$ is controlled by $|\lambda|_{\text{max}}$, the presence of larger time scales (hence smaller eigenvalues) provides additional slower dynamic characteristics to $y(t)$. The issue is when $|\lambda|_{\text{min}}$ is very small compared to $|\lambda|_{\text{max}}$: capturing the slow transient whose time scale is determined by $|\lambda|_{\text{min}}$ while the spectral radius imposes tiny time steps for the sake of stability makes the computation unaffordable. When such characteristics are met, the system is said to be stiff.

Definition of Stiffness

A system is **stiff** when the eigenvalues λ_i of operator $\boldsymbol{A}$ are such that:

$$\frac{|\lambda_{\text{max}}|}{|\lambda_{\text{min}}|} \gg 1 \tag{3.71}$$

- **Explicit Methods**: stiffness can become quickly overwhelming because of the competition between the necessity of tiny time steps compared to the long-term dynamics to capture $\Longrightarrow$ The necessary amount of iterations is unaffordable and **explicit methods cannot handle stiff problems**.
- **Implicit Methods** are the only way in such circumstances. However, if h is too large, the method becomes inaccurate. Accuracy control is then required $\Longrightarrow$ time-step adaption and dedicated solvers are needed.

3.8 Towards Higher-Order Methods

Let's recall the Forward Euler Method:

$$y_{n+1} = y_n + h \underbrace{y'_n}_{f(y_n,t_n)} + \mathcal{O}(h^2)$$

Expanding the Taylor series to higher order terms yields seems a straightforward approach to derive a higher-order numerical method:

$$y_{n+1} = y_n + hy'_n + \frac{h^2}{2}y''_n + \frac{h^3}{6}y'''_n + ... \tag{3.72}$$

with

$$y'_n = f(y_n, t_n) \tag{3.73}$$

$$y''_n = \left.\frac{dy'}{dt}\right|_{t_n} = \frac{df}{dt} = \left(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}\right)_{t=t_n} = \left(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}f\right)_{t=t_n} \tag{3.74}$$

$$y'''_n = \left.\frac{dy''}{dt}\right|_{t_n} = \left(\frac{\partial y''}{\partial t} + \frac{\partial y''}{\partial y}\frac{\partial y}{\partial t}\right)_{t=t_n} \tag{3.75}$$

$$= \left(\frac{\partial^2 f}{\partial t^2} + 2f\frac{\partial f}{\partial y}\frac{\partial f}{\partial t} + \frac{\partial f}{\partial t}\frac{\partial f}{\partial y} + f\left(\frac{\partial f}{\partial y}\right)^2 + f^2\frac{\partial^2 f}{\partial y^2}\right)_{t=t_n} \tag{3.76}$$

$$y_n^{(4)} = ... \tag{3.77}$$

Such developments get rapidly impractical. Which solutions are viable then? Two categories of methods are distinguished

- One-step methods, also known as **Runge-Kutta** methods.
  Higher accuracy is achieved by evaluating $f(y,t)$ at several points between t_n and $t_{n+1} = t_n + h$.
  $\rightarrow$ several sub-steps are computed
- **Multi-step** methods:
  Use knowledge of previous steps $y_{n-1}, y_{n-2}, ...; f(y_{n-1}), f(y_{n-2}), ...$

3.9 Runge-Kutta Methods

Runge-Kutta methods are denoted as RK-s where s is the number of stages in the numerical scheme that are carried out to determine the value y_{n+1}. The 1^{st}-order explicit RK1 method is noting else than the Forward Euler method. Explicit RK2, RK3 and RK4 methods are presented in this section.

3.9.1 Standard RK2 method

Integrating $\frac{dy}{dt}$ between instants t_n and t_{n+1} gives the following exact estimation of y_{n+1}:

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(y(\tau), \tau)d\tau$$

Approximating the integral by the trapezoidal rule previously enabled to derive the Trapezoidal method. Another quadrature rule is here considered: the mid-point rule, which yields

$$y_{n+1} \approx y_n + h\ f(y(t), t)|_{t=t_{n+1/2}} \tag{3.78}$$

$$y_{n+1} = y_n + hf(y_{n+1/2}, t_{n+1/2}) \tag{3.79}$$

However, the value $y_{n+1/2}$ is not known and must be determined to close the formula. Let us estimate $y_{n+1/2}$ with Forward Euler:

$$y_{n+1/2} \approx y^*_{n+1/2} = y_n + \frac{h}{2} f(y_n, t_n) \tag{3.80}$$

The obtained method is the standard explicit 2-stage RK Method which is defined by

Standard RK2 method

$$y^*_{n+1/2} = y_n + \frac{h}{2} f(y_n, t_n) \tag{3.81}$$

$$y_{n+1} = y_n + h f(y^*_{n+1/2}, t_{n+1/2}) \tag{3.82}$$

Stability and accuracy analyses of standard RK2

- **Amplification factor**:
  Considering the $y' = \lambda y$ ODE, we can express the amplification factor, from:

$$y^*_{n+1/2} = y_n + \frac{h}{2}\lambda y_n = \left(1 + \frac{h\lambda}{2}\right) y_n \tag{3.83}$$

$$y_{n+1} = y_n + h\lambda y^*_{n+1/2} = y_n + (h\lambda + \frac{h^2\lambda^2}{2}) y_n = \sigma y_n \tag{3.84}$$

  Hence

$$\boxed{\sigma = 1 + h\lambda + \frac{h^2\lambda^2}{2}.} \tag{3.85}$$

- **Accuracy**: this corresponds to 2[nd]-order (global) accuracy.
- **Stability**: the stability condition writes

$$|\sigma| < 1, \tag{3.86}$$

  hence

$$\left|1 + h\lambda + \frac{(h\lambda)^2}{2}\right| < 1 \tag{3.87}$$

  The corresponding stability region is shown in Fig. 3.3. The RK2 method is then conditionally stable with a stadium-shaped stabiliy region. The region has a larger vertical extent than the Forward Euler method. When the λ is real and negative the stability constraint is the same as the Euler method.

Properties of RK2 method

- Explicit, 2[nd]-order accurate,
- Conditionally stable

- $\lambda \in \mathbb{R}^-$: same stability region as Forward Euler

- $\lambda \in i\mathbb{R}$: unstable, yet with a smaller growth rate than Forward Euler.

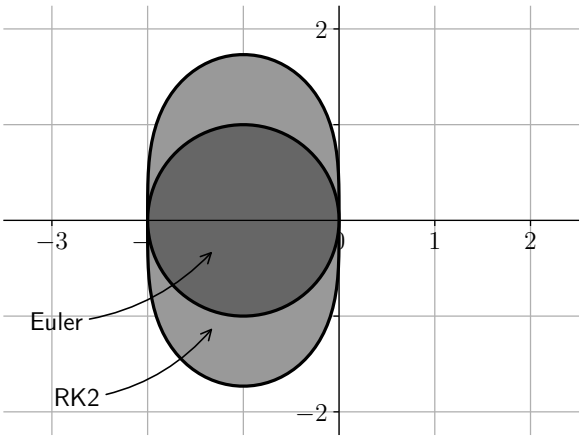


Figure 3.3 : *Stability region of the 2-stage Runge-Kutta Method*

3.9.2 Family of 2-stage explicit Runge-Kutta Methods

Before discussing the general formalism of RK methods, let us highlight that their exist infinite variants of RK-s schemes which is illustrated here for RK2. A general 2-stage explicit Runge-Kutta method is built from the knowledge of $f(y,t)$ in two points in the interval between t_n and t_{n+1}. Let's then write a general explicit RK2 scheme as

$$y_{n+1} = y_n + \gamma_1 h k_1 + \gamma_2 h k_2, \tag{3.88}$$

where

$$k_1 = f(y_n, t_n) \tag{3.89}$$
$$k_2 = f(y_n + \beta h k_1, t_n + \alpha h) \tag{3.90}$$

Let us determine $\alpha, \beta, \gamma_1, \gamma_2$ that provide the highest accuracy order for y_{n+1}
We have seen that

$$y_{n+1} = y_n + h y'_n + \frac{h^2}{2} y''_n + ... \tag{3.91}$$
$$= y_n + h f(y_n, t_n) + \frac{h^2}{2}\left(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f\right)_{t=t_n} + ... \tag{3.92}$$

which is to be matched up to the 3$^{\text{rd}}$ term by the considered RK method in Eq. (3.88). Using Taylor series, the general RK2 method writes:

$$y_{n+1} = y_n + \gamma_1 h f(y_n, t_n) + \gamma_2 h f(y_n + \beta h k_1, t_n + \alpha h) \tag{3.93}$$
$$y_{n+1} = y_n + \gamma_1 h f(y_n, t_n) + \gamma_2 h \left[f(y_n, t_n) + \beta h k_1 \left.\frac{\partial f}{\partial y}\right|_{t_n} + \alpha h \frac{\partial f}{\partial t}_{t_n} + ... \right] \tag{3.94}$$

Then

$$y_{n+1} = y_n + h(\gamma_1 + \gamma_2) f(y_n, t_n) + \gamma_2 \beta h^2 f(y_n, t_n) \left.\frac{\partial f}{\partial y}\right|_{t_n} + \gamma_2 \alpha h^2 \left.\frac{\partial f}{\partial t}\right|_{t_n} + ... \tag{3.95}$$

Matching the terms in Eq. (3.91) results in the following conditions for the coefficients in an explicit RK2 method:

$$\begin{cases} \gamma_1 + \gamma_2 = 1 \\ \gamma_2 \alpha = \dfrac{1}{2} \\ \gamma_2 \beta = \dfrac{1}{2} \end{cases} \tag{3.96}$$

Hence a family of 2^{nd}-order accurate RK methods:

$$\begin{cases} \beta = \alpha \\ \gamma_1 = 1 - \dfrac{1}{2\alpha} \\ \gamma_2 = \dfrac{1}{2\alpha} \end{cases} \tag{3.97}$$

where α is a **free** parameter. In particular, the standard RK2 method described in the previous section is retrieved for $\alpha = \frac{1}{2}$. This specific choice of α also corresponds to the Runge method, which is then another name of the so-called standard RK2.

3.9.3 General explicit Runge-Kutta Method

The general form of s-stage explicit Runge-Kutta schemes is:

$$\begin{cases} k_1 = f(t_0, y_n) \\ k_2 = f(t_0 + c_2 h, y_n + h a_{21} k_1) \\ k_3 = f(t_0 + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2)) \\ \quad \vdots \\ k_s = f(t_0 + c_s h, y_n + h(a_{s1} k_1 + ... + a_{s,s-1} k_{s-1})) \\ y_{n+1} = y_n + h(b_1 k_1 + ... + b_s k_s) \end{cases} \tag{3.98}$$

Corresponding coefficients are usually represented in a Butcher's table:

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
...	...		$\ddots$		
c_s	a_{s1}	a_{s2}	...	$a_{s,s-1}$	
	b_1	b_2	...	b_{s-1}	b_s

As for 2-stage RK family, there is no unicity of k^{th}-order RK methods.

Remark

In addition to constraining the order accuracy, it is typically desired that intermediate values of f, k_j, remain 1^{st}-order approximation of $f(y(t), t)$, which requires:

$$\sum_{j=1}^{i-1} a_{ij} = c_i \tag{3.99}$$

3.9.4 Usual explicit Runge-Kutta Methods

Common explicit RK2, RK3 and RK4 methods are listed below with their Butcher tables and corresponding updating formulas:

RK2 (*Runge*)

The method is 2^{nd}-order accurate.

Butcher Tableau:

0		
$1/2$	$1/2$	
	0	1

Scheme formulation:

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \dfrac{h}{2}, y_n + \dfrac{hk_1}{2}\right) \\ y_{n+1} = y_n + hk_2 \end{cases}$$

RK3 (*Heun*)

The method is 3^{rd}-order accurate.

Butcher Tableau:

0			
$1/3$	$1/3$		
$2/3$	0	$2/3$	
	$1/4$	0	$3/4$

Scheme formulation:

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \dfrac{h}{3}, y_n + \dfrac{hk_1}{3}\right) \\ k_3 = f\left(t_n + \dfrac{2h}{3}, y_n + \dfrac{2hk_2}{3}\right) \\ y_{n+1} = y_n + \dfrac{h}{4}k_1 + \dfrac{3h}{4}k_3 \end{cases}$$

RK3 (*Kutta*)

The method is 3^{rd}-order accurate.

Butcher Tableau:

0			
$1/2$	$1/2$		
1	-1	2	
	$1/6$	$2/3$	$1/6$

Scheme formulation:

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \dfrac{h}{2}, y_n + \dfrac{hk_1}{2}\right) \\ k_3 = f\left(t_n + h, y_n - hk_1 + 2hk_2\right) \\ y_{n+1} = y_n + \dfrac{h}{6}k_1 + \dfrac{2h}{3}k_2 + \dfrac{h}{6}k_3 \end{cases}$$

RK4 (*Classical*)

The method is 4^{th}-order accurate.

Butcher Tableau:

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

Scheme formulation:

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{hk_1}{2}\right) \\ k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{hk_2}{2}\right) \\ k_4 = f\left(t_n + h, y_n + hk_3\right) \\ y_{n+1} = y_n + \frac{h}{6}k_1 + \frac{h}{3}k_2 + \frac{h}{3}k_3 + \frac{h}{6}k_4 \end{cases}$$

RK4 ($3/8$-*Rule*)

The method is 4^{th}-order accurate.

Butcher Tableau:

0				
1/3	1/3			
2/3	−1/3	1		
1	1	−1	1	
	1/8	3/8	3/8	1/8

Scheme formulation:

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \frac{h}{3}, y_n + \frac{hk_1}{3}\right) \\ k_3 = f\left(t_n + \frac{2h}{3}, y_n - \frac{hk_1}{3} + hk_2\right) \\ k_4 = f\left(t_n + h, y_n + hk_1 - hk_2 + hk_3\right) \\ y_{n+1} = y_n + \frac{h}{8}k_1 + \frac{3h}{8}k_2 + \frac{3h}{8}k_3 + \frac{h}{8}k_4 \end{cases}$$

Remark

- *The order accuracy of RK2, RK3 and RK4 methods can **wrongly** lead us to believe that explicit s-stage Runge-Kutta method are s^{th}-order accurate. In fact, for order accuracy $n \geq 5$, explicit RK methods require $s > n$ stages.*
- *As mentioned before, an infinity of possible s-stage RK exist. Some specific RK are preferred thanks to:*
  - *desired properties (e.g. Strong Stability Preserving, Total Variation Diminishing)*
  - *storage savings*

3.9.5 Stability of explicit Runge-Kutta Methods

One can show that all s-stage explicit RK Methods have the following amplification factor:

$$\boxed{\sigma = \sum_{j=0}^{s} \frac{(h\lambda)^j}{j!}} \tag{3.100}$$

Hence, for **all**:

- 2-stage RK, $\sigma = 1 + h\lambda + \frac{(h\lambda)^2}{2}$
- 3-stage RK, $\sigma = 1 + h\lambda + \frac{(h\lambda)^2}{2} + \frac{(h\lambda)^3}{6}$
- 4-stage RK, $\sigma = 1 + h\lambda + \frac{(h\lambda)^2}{2} + \frac{(h\lambda)^3}{6} + \frac{(h\lambda)^4}{24}$

For a given number of stages, RK methods have then the same stability properties. The stability region in the complex plane is shown in Fig. 3.4. The RK3 and RK4 methods are characterized by a larger stability domain. **Specifically, notice that RK3 and RK4 stability regions include one part of the imaginary axes**, which makes them very useful when purely imaginary complex eigenvalues are present in the ODE system. So far, the Backward Euler and Trapezoidal methods were the only methods able to remain stable in such circumstances (Forward Euler and RK2 methods are unstable) but they were implicit. RK3 and RK4 are then widely used explicit methods.

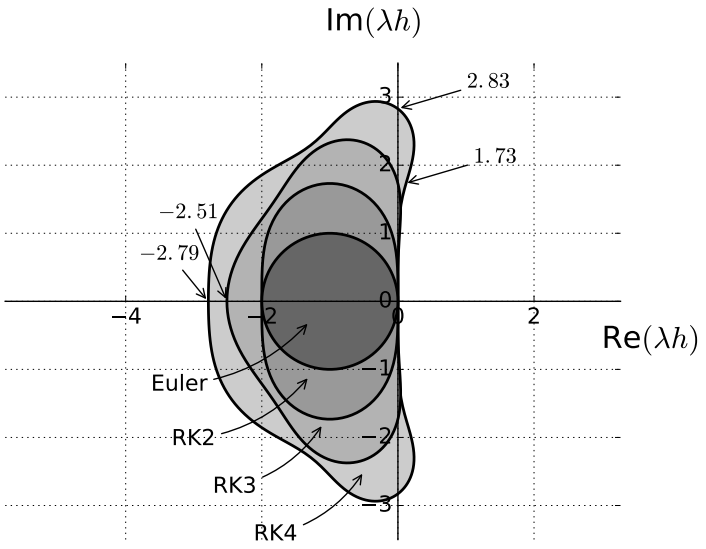


Figure 3.4 : *Stability regions of Runge-Kutta schemes of orders ranging from 1 to 4. Grey areas indicate stable regions.*

Remark

Implicit RK methods can also be built (see Hairer's books). Yet, not all of them are inconditionnaly stable.

3.10 Multi-step methods

Contrary to RK methods which feature several evaluations between t_n and t_{n+1}, multi-step methods consider previous steps data such as $y_{n-1}, f_{n-1}, ...$

Remark

Since the updating formula of multi-step methods uses values such as y_{n-1}, they cannot be applied for the first integration step because only y_0 is known then, and there is no knowledge of y_{-1}. Multistep methods require then a specific initialization process using other formulas.

3.10.1 Adams Methods

The derivation of Adams methods starts from the integrated ODE expression between instants t_n and t_{n+1},

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f\big(\tau, y(\tau)\big) d\tau.$$

The function $f\big(t, y(t)\big)$ in the integral is then approximated as polynomial which interpolates the previous values $f_n, f_{n-1}, ...$ to build an explicit formula. Explicit Adams methods are called Adams-Bashforth methods. The first-order Adams-Bashforth consider scheme considers $f\big(t, y(t)\big) \approx f_n$, which yields the Forward Euler method.

The second-order Adams-Bashforth The second-order Adams-Bashforth method is then built by considering $f\big(t, y(t)\big)$ as polynomial of degree 1. The linear approximation is deduced from previous data at t_{n-1}:

$$f\big(t, y(t)\big) = \alpha t + \beta \tag{3.101}$$

$$= \frac{f_n - f_{n-1}}{t_n - t_{n-1}}(t - t_{n-1}) + f_{n-1}. \tag{3.102}$$

Hence, the previous integral is written as

$$\int_{t_n}^{t_{n+1}} f\big(\tau, y(\tau)\big) d\tau = \left[\frac{f_n - f_{n-1}}{h}\frac{(t - t_{n-1})^2}{2} + f_{n-1}t\right]_{t_n}^{t_{n+1}} \tag{3.103}$$

$$= \frac{3}{2}hf_n - \frac{1}{2}hf_{n-1}, \tag{3.104}$$

which defines the updating formula for the second-order Adams-Bashforth method.

2nd-order Adams-Bashforth method

$$y_{n+1} = y_n + \frac{3}{2}hf_n - \frac{1}{2}hf_{n-1} \tag{3.105}$$

- **Accuracy:** The 2nd-order Adams-Bashforth method can also be derived from the Taylor series of $y(t)$:

$$y_{n+1} = y_n + h\underbrace{y'_n}_{f_n} + \frac{h^2}{2}\underbrace{y''_n}_{\frac{f_n - f_{n-1}}{h} + \mathcal{O}(h)} + \frac{h^3}{6}y'''_n + ... \tag{3.106}$$

$$= \underbrace{y_n + \frac{3}{2}hf_n - \frac{1}{2}hf_{n-1}}_{2^{\text{nd}}\text{-order Adams-Bashforth}} + \underbrace{\mathcal{O}(h^3)}_{\text{Local truncation error}} \tag{3.107}$$

Hence, **the so-called 2^{nd}-*order* Adams-Bashforth is indeed 2^{nd}-order accurate globally**.

- **Stability:** Considering $y' = \lambda y$ in a multi-step method such as the 2^{nd}-order Adams-Bashforth gives the following second-order difference equation

$$y_{n+1} - \left(1 + \frac{3h\lambda}{2}\right) y_n + \frac{h\lambda}{2} y_{n-1} = 0 \tag{3.108}$$

This recurrence equation does not result in $y_{n+1} = \sigma y_n$ with a single amplification factor anymore. Instead, solving the quadratic characteristic equation

$$\sigma^2 - \left(1 + \frac{3h\lambda}{2}\right) \sigma + \frac{h\lambda}{2} = 0, \tag{3.109}$$

gives

$$y_n = c_1 \sigma_1^n + c_2 \sigma_2^n, \tag{3.110}$$

if the characteristic equation has two distinct roots σ_1 and σ_2:

$$\sigma_{1,2} = \frac{1}{2}\left[\frac{1}{2} + \frac{3}{2}h\lambda \pm \sqrt{1 + h\lambda + \frac{9}{4}(h\lambda)^2}\right], \tag{3.111}$$

that are complex numbers in general for complex $h\lambda$. The numerical method is stable if $|\sigma_1| \leq 1$ and $|\sigma_2| \leq 1$. The stability region is shown in Fig. 3.5 along with Adams-Bashforth methods or order 1 and 3. It is seen that the stability domain is reduced as the order-accuracy increases.

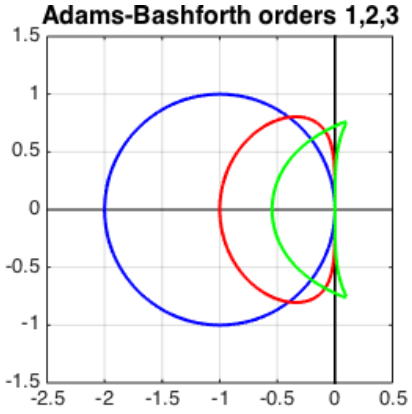


Figure 3.5 : *Stability region (within each colored curve) of the 1^{st}(blue), 2^{nd}(red) and 3^{rd}(green) Adams-Bashforth methods.*

Generalization of Adams method: Increasing the degree of the polynomial approximating $f\big(t, y(t)\big)$ gives higher-order Adams-Bashforth methods.

Exercise 6

Decomposing the solution at time T, one gets Considering a quadratic polynomial for $f\big(t, y(t)\big)$ which interpolates the values f_n, f_{n-1} and f_{n-2}, demonstrate that the 3^{rd}-order Adams-Bashforth

formula is given by:

$$y_{n+1} = y_n + h\left(\frac{23}{12}f_n - \frac{4}{3}f_{n-1} + \frac{5}{12}f_{n-1}\right) \tag{3.112}$$

Remark

Implicit *Adams methods, called Adams-Moulton, are derived by including f_{n+1} in the interpolated values by the polynomial approximating $f\big(t, y(t)\big)$.*

3.10.2 Leap-Frog

Integrating the ODE between the instants t_{n-1} and t_{n+1},

$$y_{n+1} = y_{n-1} + \int_{t_{n-1}}^{t_{n+1}} f\big(\tau, y(\tau)\big)d\tau, \tag{3.113}$$

yields a new set of numerical methods. Using the mid-point quadrature rule,

$$\int_{t_{n-1}}^{t_{n+1}} f\big(\tau, y(\tau)\big)d\tau \approx 2hf_n, \tag{3.114}$$

gives the updating formula for the Leap-Frog method, also known as the mid-point rule method.

Leap-Frog or Mid-point Rule method

$$y_{n+1} = y_{n-1} + 2hf_n \tag{3.115}$$

Only the Leap-Frog method is detailed here. Other methods of higher-order accuracy are obtained similarly to the Adams methods by increasing the degree of the interpolating polynomial $f\big(\tau, y(\tau)\big)$. The obtained formulas are called Nyström methods when explicit and Milne-Simpson methods when implicit. Because of their very small stability region (even implicit methods), these numerical schemes are of little practical interest.

- **Accuracy:** The Leap-Frog formula is also obtained with the 2^{nd}-order differential formula

  $$y'_n = \frac{y_{n+1} - y_{n-1}}{2h} + \mathcal{O}(h^2) \tag{3.116}$$

  injected in the Taylor sries of $y(t)$, yielding

  $$y_{n+1} = y_{n-1} + 2hf_n + mathcalO(h^3). \tag{3.117}$$

  The Leap-Frog method is then 2^{nd}-order accurate.

- **Stability:** considering $y' = \lambda y$, the updating formula becomes $y_n = c_1\sigma_1^n + c_2\sigma_2^n$ with σ_1, σ_2 roots of the characteristic equation

  $$\sigma^2 - 2h\lambda\sigma - 1 = 0. \tag{3.118}$$

  It is then stable for $|\sigma_1| \leq 1$ and $|\sigma_2| \leq 1$. Let us consider the image of the unit circle $\sigma = e^{i\theta}$ by $h\lambda = f(\sigma)$.

  $$h\lambda = \frac{\sigma^2 - 1}{2\sigma} = \frac{e^{i\theta} - e^{-i\theta}}{2} = i\sin(\theta) \tag{3.119}$$

with $\theta \in [0, 2\pi]$. The image of the disc $\sigma \in D(0,1)$ is then the segment $h\lambda \in [-i, i]$. **The $[-i, i]$ segment is then the stability region of the Leap-Frog method**. The Leap-Frog method is then only stable on the imaginary axes. The usage of this scheme is then limited but is useful in specific conditions such as centered schemes used for advection, which will be described later in hyperbolic PDEs.

3.10.3 Backward differentiation formula

BDF methods are a family of implicit multistep methods defined by writing:

$$y'_{n+1} = f_{n+1}, \tag{3.120}$$

and using backward difference formula for y_{n+1}.

BDF methods

- BDF1 (1^{st}-order accurate):

$$y_{n+1} - y_n = hf_{n+1} \quad \big(\equiv \text{Backward Euler} \big) \tag{3.121}$$

- BDF2 (2^{nd}-order accurate):

$$\frac{3}{2}y_{n+1} - 2y_n + \frac{1}{2}y_{n-1} = hf_{n+1} \tag{3.122}$$

- BDF3 (3^{rd}-order accurate) :

$$\frac{11}{6}y_{n+1} - 3y_n + \frac{3}{2}y_{n-1} + \frac{1}{3}y_{n-2} = hf_{n+1} \tag{3.123}$$

- BDF4 ...

BDF7 formula and other formula of higher-order are unstable and then useless. Only the BDF1 method, a.k.a the Backward-Euler method, is unconditionally stable. The other methods, BDF2-BDF6, are only conditionally stable with an interesting large stability region, which is what has made these methods popular. There is in fact no multistep methods of high-order that is unconditionally stable. This is stated by the **2^{nd}Barrier of Dahlquist: unconditionally stable linear multistep methods are necessarily of order ≤ 2**. Therefore, Backward-Euler and Trapezoïdal methods are the only unconditionally stable multi-step methods.

Remark

A similar procedure for explicit schemes, using backward difference formula for $y'_n = f_n$, is useless:

- *Obtained lower-order methods are already known methods: Forward Euler, Leap-Frog.*
- *Higher-orders always yield unstable schemes.*

Cheat Sheet: numerical schemes for Ordinary Differential Equations

- **Basic one-step methods**
  - Forward Euler Method
$$y_{n+1} = y_n + hf(y_n, t_n)$$
  - Backward Euler Method
$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1})$$
  - Trapezoidal Method
$$y_{n+1} = y_n + \frac{h}{2}\Big(f(y_n, t_n) + f(y_{n+1}, t_{n+1})\Big)$$
- **Higher-oder Runge-Kutta methods**
  - Standard RK2 method
$$\begin{aligned} y^*_{n+1/2} &= y_n + \frac{h}{2} f(y_n, t_n) \\ y_{n+1} &= y_n + hf(y^*_{n+1/2}, t_{n+1/2}) \end{aligned}$$
  - RK3 (*Heun*) method
$$\begin{aligned} y^*_n &= y_n + \frac{h}{3} f(t_n, y_n) \\ y^{**}_n &= y_n + \frac{2h}{3} f\left(t_n + \frac{h}{3}, y^*_n\right) \\ y_{n+1} &= y_n + \frac{h}{4} f(t_n, y_n) + \frac{3h}{4} f\left(t_n + \frac{2h}{3}, y^{**}_n\right) \end{aligned}$$
- **Higher-oder Multi-step methods**
  - Second-order Adams-Bashforth method
$$y_{n+1} = y_n + \frac{3}{2} hf_n - \frac{1}{2} hf_{n-1}$$
  - Leap-Frog or Mid-point Rule method
$$y_{n+1} = y_{n-1} + 2hf_n$$
  - BDF2
$$\frac{3}{2} y_{n+1} - 2y_n + \frac{1}{2} y_{n-1} = hf_{n+1}$$

Chapter 4

Elliptic Partial Differential Equations

Contents

4.1 Numerical resolution of Elliptic PDEs

In elliptic problems, the solution at one point physically depends on all other points, and reciprocally. Consequently, the construction of an equivalent approach as time-marching methods used to solve ODEs is not possible. This will be different for parabolic and hyperbolic problems studied in next chapters. The solution field governed by an elliptic PDE must then be **solved as a whole**. Additionally, the nature of elliptic problems require **boundary conditions at all boundaries** of the considered computational domain .

4.1.1 Linear Elliptic PDEs

Laplace and Poisson equations are examples of linear elliptic partial derivative equations[1]. They are encountered in the steady heat equation with constant properties,

$$\Delta T = 0 \text{ or } \Delta T = S, \tag{4.1}$$

or in fluid mechanics in Stokes flows, where the viscous effects dominate (for very small Reynolds number), yielding for each component of the velocity field:

$$\Delta u_i = 0 \tag{4.2}$$

Given the properties of elliptic PDEs, the discretized PDE system involves linear equations which makes the solution at one point coupled to all the others. The continuous PDE is then expressed as a linear system:

[1] The *linear* property of a partial derivative equation characterizes the homogeneous PDE

Discretized Elliptic PDE

Linear system: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{x}$ gathers all unknowns in a single vector (4.3)

The coupling between all points is determined by the matrix $\boldsymbol{A}$. **Solving numerically elliptic systems is all about solving a *large* linear system**. The fact that the system is large is a significant source of difficulty to carry out the numerical resolution. Let's consider for example a three-dimensional problem discretized with 100 points in each direction. The resulting mesh results in $n = 100 \times 100 \times 100 = 10^6$ unknowns. Consequently, the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ involves a $10^6 \times 10^6$ matrix!!

4.1.2 Non-linear Elliptic PDE

In several cases, the elliptic PDE is not linear. Steady heat equation with variable properties where the thermal conductivity depends on temperature, $\lambda(T)$, is one example:

$$\frac{\partial}{\partial x_j}\left(\lambda(T)\frac{\partial T}{\partial x_j}\right) = P. \tag{4.4}$$

The steady incompressible Navier-Stokes equations are another example,

$$\rho u_j \frac{\partial u_i}{\partial x_j} = \rho g_i - \frac{\partial p}{\partial x_i} + \mu \frac{\partial^2 u_i}{\partial x_i \partial x_j}, \tag{4.5}$$

where the non-linear convective terms make the PDE non-linear. Discretizing such PDEs result in solving a non-linear vectorial function

$$\boldsymbol{F}(\boldsymbol{x}) = 0, \tag{4.6}$$

where $\boldsymbol{x}$ is the vector of unknowns. Solving $\boldsymbol{F}(\boldsymbol{x}) = 0$ is done by using one of two following iterative methods:

- **Newton's methods.** Truncation of the Taylor series to the first-order term leads to the Newton method,

  $$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{J}(\boldsymbol{x_k})^{-1}\boldsymbol{F}(\boldsymbol{x}_k), \tag{4.7}$$

  where $\boldsymbol{J} = \dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}$ is the Jacobian matrix of the function $\boldsymbol{F}$. Upon correct conditions, the sequence of $\boldsymbol{x}_k$ converges to the searched solution. This classical Newton formula is usually improved (trust region, damping methods, ..) to yield a larger convergence region around the solution. Similarly to a linear elliptic PDE, the linearized equation obtained by the Newton's method requires the resolution of the linear system $\boldsymbol{J}(\boldsymbol{x_k})(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = -\boldsymbol{F}(\boldsymbol{x}_k)$.

- **Fixed point method or Picard iterations.** The fixed point iterations, $\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k)$, is an iterative methods to find the fixed point, $x = f(x)$, of a function. Our investigated equation, $\boldsymbol{F}(\boldsymbol{x}) = 0$, is easily turned into a fixed-point problem by considering the equivalent formula, $\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{G}(\boldsymbol{x})$. Applying the fixed-point method to $\boldsymbol{G}$ gives following iterative formula,

  $$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{F}(\boldsymbol{x}_k), \tag{4.8}$$

  which makes it a straightforwardly solved linear system for $\boldsymbol{x}_{k+1}$. In this simple fixed point method, the evaluations of $\boldsymbol{F}(\boldsymbol{x})$ are computed from the old iterate. When the sequence of iterates converges, $\boldsymbol{x}_k$ will ultimately reach the root of $\boldsymbol{F}(\boldsymbol{x})$. Picard iterations is a similar

approach which considers that the non-linear equation $\boldsymbol{F}(\boldsymbol{x}) = 0$ can be written as quasi-linear one:

$$\boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} = \boldsymbol{b}(\boldsymbol{x}) \tag{4.9}$$

The Picard iterations then solve the sequence,

$$\boldsymbol{A}(\boldsymbol{x}_k)\boldsymbol{x}_{k+1} = \boldsymbol{b}(\boldsymbol{x}_k) \Rightarrow \boldsymbol{x}_{k+1} = \boldsymbol{A}(\boldsymbol{x}_k)^{-1}\boldsymbol{b}(\boldsymbol{x}_k), \tag{4.10}$$

which is identical as the fixed-point iterations for the function $\boldsymbol{A}(\boldsymbol{x})^{-1}\boldsymbol{b}(\boldsymbol{x})$. Carrying out Picard iterations introduces outer iterations that surround the numerical resolution of a linear system. For example, the aforementioned steady heat equation with variable properties is iteratively solved by setting a frozen field of thermal conductivity $\lambda^{old} = \lambda(T^{old})$, solving the obtained linear PDE

$$\frac{\partial}{\partial x_j}\left(\lambda^{old}\frac{\partial T}{\partial x_j}\right) = P, \tag{4.11}$$

and updating the frozen thermal conductivity with $T^{old} = T^{new}$. Similarly, the linear PDE considered by Picard iterations to solve the steady incompressible Navier-Stokes equations is

$$\rho u_j^{old}\frac{\partial u_i}{\partial x_j} = \rho g_i - \frac{\partial p}{\partial x_i} + \mu\frac{\partial^2 u_i}{\partial x_i \partial x_j}. \tag{4.12}$$

The distinction between Newton and Picard iterations is not about the detailed algorithm since Newton iterates can also be regarded as a sequence of a fixed-point method. The difference is mainly about how the linearization of the PDE is carried. While derived rigorously from Taylor series in Newton's method, it is achieved somehow manually in Picard iterations by manipulating the terms which are the sources of the non-linearity. **When the non-linearity arises from a complexification of a linear-variant of the PDE, Picard iterations are most often preferred because the algorithm used for the linear PDE is simply extended with outer iterations**. This is the case in both examples that have been considered in this section. Whatever the method, all of them eventually require solving linear systems, which once again highlights the key role of numerical resolution of large $\boldsymbol{Ax} = \boldsymbol{b}$ systems.

4.1.3 Boundary conditions

Elliptic PDEs require boundary conditions at all boundaries to determine a unique field ϕ. Three kinds of boundary conditions are typically considered:

- Dirichlet boundary conditions where the boundary values are fixed (not necessarily uniformly)

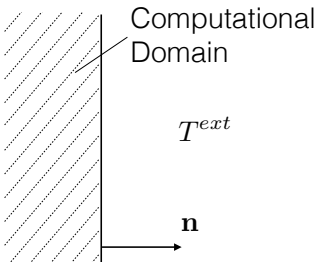
$$\phi = a \tag{4.13}$$

- Neumann boundary conditions where the normal derivative is fixed (not necessarily uniformly)

$$\frac{\partial \phi}{\partial n} = b \tag{4.14}$$

- Robin or mixed boundary conditions where a linear combination of Dirichlet and Neumann conditions is imposed

$$\alpha\phi + \beta\frac{\partial \phi}{\partial n} = c \tag{4.15}$$



In heat transfer problem, a common case where mixed boundary conditions are encountered is when a heat transfer coefficient h is provided. The normal component of the boundary heat flux $\boldsymbol{\varphi}$ is then given by

$$\boldsymbol{\varphi} \cdot \boldsymbol{n} = -\lambda \frac{\partial T}{\partial n} = h(T - T^{ext}), \tag{4.16}$$

t which relates both T and $\dfrac{\partial T}{\partial n}$ in a linear relationship.

4.2 2D Laplace/Poisson equation

This will be our running example to illustrate the numerical resolution of a linear elliptic PDE.

4.2.1 Discretization

The system of equations is made of the two-dimensional Poisson equation and boundary conditions

$$\begin{cases} \dfrac{\partial^2 \phi}{\partial x^2} + \dfrac{\partial^2 \phi}{\partial y^2} = f(x, y) \\ + \text{Boundary Conditions} \end{cases} \tag{4.17}$$

Using finite differences, the system is solved on a 2D structured mesh, shown in Fig. 4.2. Each point

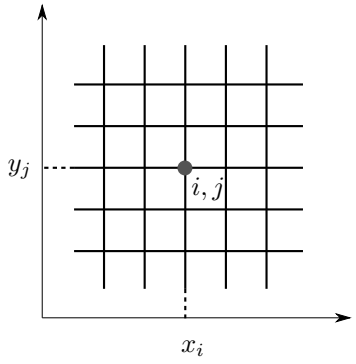


Figure 4.1 : *Two-dimensional structured mesh*

is identified by a unique pair of indexes (i, j) and its coordinates are (x_i, y_j). Assuming a uniform discretization in x and y directions, $\Delta x = \Delta y = h$, the second-order derivatives in the Poisson equations are approximated with difference formula. Using 2nd-order centered difference formula

$$\left.\frac{\partial^2 \phi}{\partial x^2}\right|_{x,y} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{h^2} + \mathcal{O}(h^2) \tag{4.18}$$

$$\left.\frac{\partial^2 \phi}{\partial y^2}\right|_{x,y} = \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{h^2} + \mathcal{O}(h^2), \tag{4.19}$$

yields a 2nd-order accurate discretization of the PDE:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{h^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{h^2} = f_{ij} + \mathcal{O}(h^2). \tag{4.20}$$

In the previous equation, $f_{ij} = f(x_i, y_j)$. A linear systems of equations is obtained by writing for each point in the domain

Discretized Poisson equation

$$\phi_{i+1,j} + \phi_{i-1,j} - 4\phi_{i,j} + \phi_{i,j+1} + \phi_{i,j-1} = h^2 f_{ij} \tag{4.21}$$

This formula is obtained with a specific 2D stencil. Other difference formulas yield different ones. 2D stencils are generally plotted to be distinguished from each other. The one used is depicted below

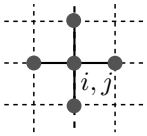


Figure 4.2 : *2D stencil corresponding to the 2^{nd}-order centered difference formula applied to the Poisson/Laplace equation.*

4.2.2 Boundary conditions

Boundary conditions must also be discretized if necesary.

4.2.2.1 Dirichlet boundary conditions

If ϕ_{ij} are known at all boundaries, the other ϕ_{ij} values are computed by solving the previous equation the inside the domain, *i.e.* for

$$\begin{cases} 1 \leq i \leq N-1 \\ 1 \leq j \leq M-1 \end{cases}, \tag{4.22}$$

where there are $N+1$ points in the x-direction and $M+1$ in the y-direction in total as shown in Fig. 4.3. The total number of unknown is then given by $n = (N-1) \times (M-1)$. A natural choice

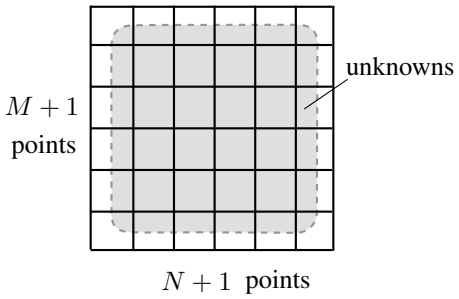


Figure 4.3 : *Unknown values in problems with Dirichlet conditions at all boundaries.*

in a problem with Dirichlet boundary conditions is to consider fixed ϕ_{ij} values as parameters which

are moved on the right-hand side of the linear system. Hence, on the left boundary, for $i = 1$,

$$\phi_{2,j} + \underbrace{\phi_{0,j}}_{\text{fixed}} - 4\phi_{1,j} + \phi_{1,j+1} + \phi_{1,j-1} = h^2 f_{ij} \tag{4.23}$$

$$\Rightarrow \phi_{2,j} - 4\phi_{1,j} + \phi_{1,j+1} + \phi_{1,j-1} = h^2 f_{ij} - \underbrace{\phi_{0,j}}_{\text{moved to RHS}}, \tag{4.24}$$

while at the lower-left corner, for $i = 1$ and $j = 1$,

$$\phi_{2,1} - 4\phi_{1,1} + \phi_{1,2} = h^2 f_{11} - \phi_{0,1} - \phi_{1,0} \tag{4.25}$$

Adopting such an implementation of Dirichlet boundary conditions is not practical. Indeed, when accounting for other possible boundary conditions, boudary values of ϕ_{ij} remain unknown variables and one would have then first to determine the number of unknown specific to the problem depending on where Dirrichlet conditions are applied. **In order to easily implement any kind of boundary conditions, all boundary values are considered unknown**. If a Dirichlet condition is applied at the left boundary for example, one adds the following set of equations $\phi_{0,j} = \phi_j^0$, where ϕ_j^0 the the imposed values at the boundary.

4.2.2.2 Other boundary conditions

Neumann boundary condition. Here, we consider the example of a Neumann boundary condition on the left boundary of the computational domain:

$$\left.\frac{\partial \phi}{\partial x}\right|_{x=0} = g(y) \tag{4.26}$$

The values $\phi_{0,j}$ are unknowns which are determined by discretizing the derivative in the boundary condition. Two difference formula are considered:

- 1^{st}-order accurate approximation

$$\left.\frac{\partial \phi}{\partial x}\right|_{x=0} = \frac{\phi_{1,j} - \phi_{0,j}}{h} = g_j \tag{4.27}$$

$$\boxed{\phi_{0,j} = -h g_j + \phi_{1,j}} \tag{4.28}$$

- 2^{nd}-order accurate approximation

$$\left.\frac{\partial \phi}{\partial x}\right|_{x=0} = \frac{-3\phi_{0,j} + 4\phi_{1,j} - \phi_{2,j}}{2h} = g_j \tag{4.29}$$

$$\boxed{\phi_{0,j} = -\frac{2}{3} h g_j + \frac{4}{3}\phi_{1,j} - \frac{1}{3}\phi_{2,j}} \tag{4.30}$$

Mixed boundary conditions. Mixed boundary conditions are treated similarly by discretizing the boundary condition.

Exercise 7

Considering a Robin boundary condition applied to the left boundary,

$$-\lambda \frac{dT}{dx} = h^{ext}(T^{ext} - T), \tag{4.31}$$

determine the 1^{st}- and 2^{nd}-order approximation of the boundary condition resulting in equations

for $\phi_{0,j}$. The expression will be written in terms of the Biot number based on the cell size Δx:

$$\text{Bi} = \frac{h^{ext}\Delta x}{\lambda}. \tag{4.32}$$

4.2.3 Characteristics of the obtained linear system

Concatenating the unknowns ϕ_{ij}, including the boundary values, gives a long vector of size $(N+1)\times(M+1)$:

$$\boldsymbol{x} = (\phi_{0,0}, \phi_{1,0}, ..., \phi_{n,0}, \phi_{1,1}, ..., \phi_{i,j}, \phi_{i+1,j}, ...)^T \tag{4.33}$$

The nth component of the vector $\boldsymbol{x}$ corresponds to a unique pair (i,j). The discretized Poisson equation and boundary conditions,

$$\begin{cases} \phi_{i+1,j} + \phi_{i-1,j} - 4\phi_{i,j} + \phi_{i,j+1} + \phi_{i,j-1} = h^2 f_{ij} \\ \qquad\qquad +\text{Boundary Conditions}, \end{cases} \tag{4.34}$$

generate a large linear system $\boldsymbol{Ax} = \boldsymbol{b}$. The matrix $\boldsymbol{A}$ is sparse, *i.e* many of its entries are null. It looks like this where dots indicate non-zero values. It is also banded with a M bandwidth. The RHS

$$\mathbf{A} = \left[\begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right] \quad \begin{array}{l} \text{n}^{th}_{(i,j)} \text{ line of } \mathbf{Ax} : \\ \phi_{i+1,j} + \phi_{i-1,j} - 4\phi_{i,j} + \phi_{i,j+1} + \phi_{i,j-1} \end{array}$$

M-banded

vector $\boldsymbol{b}$ is determined by the Poisson source term and the boundary conditions,

$$\boldsymbol{b} = \left[h^2 f_{ij} \text{ \& BC's}\right]$$

Eigenvalues of $\boldsymbol{A}$. The eigenvalues of the linear system matrix will determine a lot the behavior of the numerical methods that will be introduced later. They are here given for a case where homogeneous Dirichlet conditions ($\phi = 0$) are applied on all boundaries. Apart from the domain boundaries, they are then $n = (N-1)\times(M-1)$ unknowns. The $n \times n$ matrix $\boldsymbol{A}$ is then characterized by $(N-1)\times(M-1)$ eigenvalues given below:

Eigenvalues of $\boldsymbol{A}$ (Homogeneous BC's)

$$\begin{aligned} \lambda^A_{pq} &= -4 + 2\cos(\frac{p\pi}{N}) + 2\cos(\frac{q\pi}{M}) \\ &= -4\left(\sin^2(\frac{p\pi}{2N}) + \sin^2(\frac{q\pi}{2M})\right) \text{ with } \begin{cases} 1 \le p \le N-1 \\ 1 \le q \le M-1 \end{cases} \end{aligned} \tag{4.35}$$

Demonstration

Let's demonstrate that $\mathbf{\Phi} = \phi_{i,j} = \exp(I(k_x x_i + k_y y_j))$ are eigenvectors of the matrix $\boldsymbol{A}$ involved in a Poisson 2D problem with homogeneous Dirichlet boundary conditions. Setting $\boldsymbol{x}$, the vector of unknowns, from the field denoted by $\phi_{i,j}$, the nth line, associated to the (i,j) pair, of the matrix-vector product $\boldsymbol{Ax}$ is

$$(\boldsymbol{Ax})_{i,j} = \phi_{i+1,j} + \phi_{i-1,j} - 4\phi_{i,j} + \phi_{i,j+1} + \phi_{i,j-1} \tag{4.36}$$
$$= (\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i+1,j}) + (\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}) \tag{4.37}$$

Let's recall the modified wave number of 2$^{\text{nd}}$-order centered formula for f''_j:

$$f''_j = \frac{f_{j+1} - 2f_j + f_{j+1}}{h^2} = -k'^2 h^2 f_j \quad \text{with} \quad k'^2 h^2 = 2(1 - cos(hk)) \tag{4.38}$$

Hence,

$$(\boldsymbol{Ax})_{i,j} = -k'^2_x h^2 \phi_{i,j} - k'_y h^2 \phi_{i,j} \tag{4.39}$$
$$= \underbrace{(-k'_x h^2 - k'_y h^2)}_{\text{Eigenvalue for } \Phi} \phi_{i,j} \tag{4.40}$$

Therefore,

$$\boldsymbol{A\Phi} = \underbrace{(-4 + 2cos(k_x h) + 2cos(k_y h))}_{\text{Eigenvalue } \lambda^A \text{ of } \boldsymbol{A}} \mathbf{\Phi}. \tag{4.41}$$

The boundary conditions demand $\phi = 0$ for $x = 0$, $x = L_x$, $y = 0$ and $y = L_y$, which only allow specific wavenumbers k_x and k_y:

$$\begin{cases} k_x = \dfrac{p\pi}{L_x} = \dfrac{p\pi}{Nh} \text{ for } 1 \leq p \leq N-1 \\ k_y = \dfrac{q\pi}{L_y} = \dfrac{q\pi}{Mh} \text{ for } 1 \leq q \leq M-1 \\ \phi_{i,j} = \sin\left(\frac{p\pi}{N} i\right) \sin\left(\frac{q\pi}{M} j\right) \end{cases} \tag{4.42}$$

The eigenvalue of $\boldsymbol{A}$ for homogeneous Dirichlet boundary conditions are then the ones given in Eq. (4.35)

Condition number of A. The condition number of a matrix is the ratio of the maximum and minimum eigenvalue magnitudes. All numerical methods are strongly penalized by very large condition numbers $K(\boldsymbol{A})$, *i.e* when the system is ill-conditioned. For Laplace equation with homogeneous Dirichlet conditions and with $N = M$,

$$\lambda_{pq} = -4\left[\sin^2\left(\frac{p\pi}{2N}\right) + \sin^2\left(\frac{q\pi}{2N}\right)\right], \tag{4.43}$$

and then

$$|\lambda|_{min} = |\lambda|_{p=1,q=1} = 8\sin^2\left(\frac{\pi}{2N}\right) \sim \frac{2\pi^2}{N^2} \text{ since } N \gg 1 \tag{4.44}$$
$$|\lambda|_{max} = 8 \tag{4.45}$$

The condition number of the matrix $\boldsymbol{A}$ is therefore

$$\underbrace{K(\boldsymbol{A})}_{\text{Condition number}} = \frac{|\lambda|_{max}}{|\lambda|_{min}} = \frac{4N^2}{\pi^2} \sim n, \tag{4.46}$$

where n is the total number of unknowns in the 2D problem. Here appears the difficulty to achieve fast convergence with the different methods that will be detailed in the following sections: They are all penalized with ill-conditioned systems. Steady heat equation becomes more and more ill-conditioned as the number of points increases. Hence, many studies have been carried out to accelerate the simple iterative methods.

4.3 Direct vs Iterative methods to solve Ax = b

It was shown in Sec. 4.1 that the numerical resolution of elliptic PDEs turns out to handle a large linear system $\boldsymbol{Ax} = \boldsymbol{b}$. Two kinds of approaches can be considered:

- Direct resolution with Gauss pivoting and LU factorization.
- Iterative methods.

4.3.1 Direct resolution

When considering a large $n \times n$ matrix $\boldsymbol{A}$ as the one obtained in a discretized elliptic PDE, the direct resolution issues several limitations:

- **Storage**: Given the number of unknowns, the memory footprint to store the full matrix is tremendous. For a 2D problem with $N = 1,000$, $M = 1,000$, the total number of points is $n \approx 10^6$. The number of entries in the matrix $\boldsymbol{A}$ is then $n^2 = 10^{12}$. Accounting for 8 bytes to store one double-precision float number, the matrix storage would demands 8 TB!
- **Performance**: Gauss elimination and LU factorization are characterized by an algorithmic complexity $\mathcal{O}(n^3)$. For a 2D problem with $N = M$, we have $\mathcal{O}(N^6)$, which quickly becomes unafforadle. The previous result is valid for a full and plain matrix. For a banded matrix of bandwidth B, the complexity is $O(nB^2)$. In the 2D Laplace equation, it was shown that $B = N$, implying an algorithmic complexity $\mathcal{O}(N^4)$ which remains expansive computationally.
- **Miscellaneous** Gauss elimination and LU factorization become inaccurate with large round-off errors with a matrix of large condition number. Additionally, parallelizing the direct resolution of a linear system is difficult to implement.

Direct methods like iterative methods benefit from on-going improvements to increase their computational efficiency. The limitations of direct methods are too hard to overcome when the matrix is **dense**. However, when the matrix is **sparse**, like here, several remedies exist thanks to a compact storage, such as CSR (Compressed Sparse Row format). Parallel libraries such as MUMPS and SuperLU have become quite popular, showing that efficient parallelization is possible.

All this said, iterative approaches are usually and historically preferred for large matrices. Furthermore, their implementation allows **matrix-free** algorithms where the matrix $\boldsymbol{A}$ is not even stored.

4.3.2 Iterative methods

Also known as relaxation methods, the **principle of iterative methods is to build a sequence of solution fields that tends to x^*, the solution of original linear system $\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{b}$**:

$$\boldsymbol{x}_n \xrightarrow[n\to+\infty]{} \boldsymbol{x}^*. \tag{4.47}$$

The benefits of iterative methods are:

- **Storage**: The implementation does not require the storage of matrix A.
- **Accuracy**: The linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is an discrete approximation of the continuous PDE. There is no need then to find the solution $\boldsymbol{x}^*$ with exact accuracy. Given the remaining numerical truncation error, this over-resolution is time-consuming and unnecessary.
- **Convergence speed**: If the convergence speed of the iterative method is high enough, it will rapidly outperform direct methods.

Matrix splitting. The first general idea to derive an iterative method to solve the linear system is *matrix splitting*. The matrix $\boldsymbol{A}$ is split into two others as

Matrix Splitting

$$\boldsymbol{A} = \boldsymbol{A}_1 - \boldsymbol{A}_2 \text{ with any choice of } \boldsymbol{A}_1 \text{ and } \boldsymbol{A}_2 \tag{4.48}$$

The linear system is equivalently written as

$$\boldsymbol{A}_1 x = \boldsymbol{A}_2 x + \boldsymbol{b}. \tag{4.49}$$

The iterative method associated to a retained matrix splitting is then derived by evaluating the left-hand side at the $(k+1)^{th}$ iterate and the right-hand side at the k^{th} iterate. The iterative formula is then

Iterative method from matrix splitting

$$\boldsymbol{A}_1 \boldsymbol{x}_{k+1} = \boldsymbol{A}_2 \boldsymbol{x}_k + \boldsymbol{b}, \tag{4.50}$$

or

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_1^{-1} \boldsymbol{A}_2 \boldsymbol{x}_k + \boldsymbol{A}_1^{-1} \boldsymbol{b} \tag{4.51}$$

The iterations are initiated with an initial guess $\boldsymbol{x}^0$. The classical choice is $\boldsymbol{x}^0 = \boldsymbol{0}$. Such an iterative procedure is only interesting if i) the matrix $\mathbf{A}_1$ can easily be inversed compared to the $\boldsymbol{A}$; ii) And if the the convergence of the derived method is fast. The convergence properties are characterized by monitoring the evolution of the **error** and **residual** of the method, two key quantities that are now defined. The error at iteration k is the difference between the exact solution and the current iterate:

$$\varepsilon_k = \boldsymbol{x}^* - \boldsymbol{x}_k \tag{4.52}$$

The evolution of the error can be determined from the retained matrix splitting as,

$$\left.\begin{array}{c} \boldsymbol{A}_1\boldsymbol{x}^* = \boldsymbol{A}_2\boldsymbol{x}^* + \boldsymbol{b} \\ \boldsymbol{A}_1\boldsymbol{x}_{k+1} = \boldsymbol{A}_2\boldsymbol{x}_k + \boldsymbol{b} \end{array}\right\} \xrightarrow[\text{Subtracting}]{} \boldsymbol{A}_1\varepsilon_{k+1} = \boldsymbol{A}_2\varepsilon_k \tag{4.53}$$

Then,

$$\boxed{\varepsilon_{k+1} = \left(\boldsymbol{A}_1^{-1}\boldsymbol{A}_2\right)\varepsilon_k = \boldsymbol{G}\varepsilon_k}, \tag{4.54}$$

where $\boldsymbol{G} = \boldsymbol{A}_1^{-1}\boldsymbol{A}_2$ is the **gain matrix**. The error decrease rate is then entirely controlled by the chosen matrix splitting. That is why some iterative methods are much more interesting than others. The residual of the iterative method at iteration k is defined as

$$r_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k, \tag{4.55}$$

and denotes how far the iterate is from the solution $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^* = 0$. The residual and error are related as

$$r_k = \boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{A}\boldsymbol{x}_k = \boldsymbol{A}\varepsilon_k. \tag{4.56}$$

Hence, when the error of the iterative method decreases, so is the residual, and reciprocally. Given that the error cannot be computed without knowing first the searched solution $\boldsymbol{x}^*$, **the convergence of the iterates is monitored in practice by considering the norm of the residual:** $\|r_k\|$.

Convergence Convergence is achieved if $\varepsilon_k \xrightarrow[k\to\infty]{} 0$, implying that the residual also tends to zero. This is controlled by the spectral radius $\rho(\boldsymbol{G})$ of the gain matrix:

Convergence condition

The iterative method based on the chosen matrix splitting converges if

$$\rho(\boldsymbol{G}) = \rho(\boldsymbol{A}_1^{-1}\boldsymbol{A}_2) < 1 \tag{4.57}$$

where the spectral radius is the maximum eigenvalue amplitude: $\rho(\boldsymbol{G}) = |\lambda_i^G|_{max}$.

Demonstration

The average rate of error reduction is defined as

$$\overline{\Delta\varepsilon_n} \overset{\text{def}}{=} \left(\frac{\|\varepsilon_n\|}{\|\varepsilon_0\|}\right)^{1/n} \tag{4.58}$$

Since after each iteration $\varepsilon^n = \boldsymbol{G}\varepsilon^{n-1}$, the error at iteration n is a power function of the gain matrix: $\varepsilon_n = \boldsymbol{G}^n\varepsilon_0$, and

$$\overline{\Delta\varepsilon_n} = \left(\frac{\|\boldsymbol{G}^n\varepsilon_0\|}{\|\varepsilon_0\|}\right)^{1/n} \leq \|\boldsymbol{G}^n\|^{1/n} \tag{4.59}$$

Since the spectral radius obeys the following property,

$$\|\boldsymbol{G}^n\|^{1/n} \xrightarrow[n\to\infty]{} \rho(\boldsymbol{G}), \tag{4.60}$$

it yields that $\overline{\Delta\varepsilon_n} \leq \rho(\boldsymbol{G})$ for $n \gg 1$.Therefore,

$$\boxed{\|\varepsilon_n\| \leq \rho(\boldsymbol{G})^n \, \|\varepsilon_0\|} \tag{4.61}$$

The iterative method converges, *i.e.* the error ε_n tends to zero, if the spectral radius of the gain matrix is strictly lower than unity.

The previous demonstration outlines that the rate of convergence is also determined by $\rho(\boldsymbol{G})$ for the chosen splitting $\boldsymbol{A}_1$, $\boldsymbol{A}_2$.

Exercise 8

Starting the iterative process with $\boldsymbol{x}_0$ and the corresponding error ε_0, determine the number of iterations n to decrease the error norm by a factor 10^m.

Solution 8

One wishes $\|\varepsilon_n\| \leq 10^{-m} \, \|\varepsilon_0\|$. This condition is fulfilled if $\rho(\boldsymbol{G})^n \leq 10^{-m}$ according to the above demonstration. Therefore,

$$\boxed{n \geq \frac{-m}{\log_{10}(\rho(G))}} \tag{4.62}$$

4.4 Basic iterative methods

The matrix splitting should yield an easily inverse matrix $\boldsymbol{A}_1$ and result with a gain matrix of small spectral radius. The first attempts such as choosing $\boldsymbol{A}_1$ as the identity matrix or the diagonal part of $\boldsymbol{A}$ result in the Richardson and Jacobi methods, respectively. The Gauss-Seidel method is obtained by considering the diagonal and lower triangular parts of $\boldsymbol{A}$. These first methods are the most basic ones and have been derived during the 19^{th} century before any computer were ever built. They were then used until the mid-20^{th} century by carrying out the calculations manually.

Different parts of the matrix $\boldsymbol{A}$ are defined as

$$\boldsymbol{A} = \begin{pmatrix} \ddots & & -F \\ & D & \\ -E & & \ddots \end{pmatrix} = \boldsymbol{D} - \boldsymbol{E} - \boldsymbol{F}, \tag{4.63}$$

where $\boldsymbol{D}$ is the diagonal part of $\boldsymbol{A}$, $\boldsymbol{E}$ its upper triangular part and $\boldsymbol{F}$ its lower triangular part:

$$D_{i,j} = \begin{cases} A_{i,j} \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}, \quad E_{i,j} = \begin{cases} A_{i,j} \text{ if } i < j \\ 0 \text{ otherwise} \end{cases}, \quad F_{i,j} = \begin{cases} A_{i,j} \text{ if } i > j \\ 0 \text{ otherwise} \end{cases} \tag{4.64}$$

4.4.1 Richardson method

The simplest method, yet unstable for Laplace equation.

Let's recall the iterative process associated to a matrix splitting :

$$\boldsymbol{A}_1 \boldsymbol{x}_{k+1} = \boldsymbol{A}_2 \boldsymbol{x}_k + \boldsymbol{b} \tag{4.65}$$

The simplest choice for an easily inversable matrix is the identity matrix: $\boldsymbol{A}_1 = \boldsymbol{I}$. Then, $\boldsymbol{A}_2 = \boldsymbol{I} - \boldsymbol{A}$, and the Richardson method is given by

Richardson method

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b} \tag{4.66}$$

The iterative method converges if the corresponding gain matrix $\boldsymbol{G}_R = \boldsymbol{I} - \boldsymbol{A}$ is characterized by a spectral radius smaller than unity: $\rho(\boldsymbol{G}_R) < 1$. In fact, **the Richardson methods turns out to be a fixed-point method** applied to $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{b} - \boldsymbol{A}x$. Indeed, the fixed-point iterations are then, $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{F}(\boldsymbol{x}_k) = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$.

Application to the 2D Poisson problem. Let's consider the Richardson method, *i.e.* the fixed point method, applied to the discretized elliptic PDE seen in Sec. 4.2. The nth linear equation corresponding a pait (i, j) of 2D point indexes is given by

$$\phi_{i+1,j} + \phi_{i-1,j} - 4\phi_{i,j} + \phi_{i,j+1} + \phi_{i,j-1} = \underbrace{h^2 f_{i,j} + \text{BC}'s}_{b_{i,j}} \tag{4.67}$$

The corresponding fixed-point iterations are then

$$\phi_{i,j}^{k+1} = \phi_{i,j}^k + \left[b_{i,j} - (\phi_{i+1,j}^k + \phi_{i-1,j}^k - 4\phi_{i,j}^k + \phi_{i,j+1}^k + \phi_{i,j-1}^k)\right] \tag{4.68}$$

$$\boxed{\phi_{i,j}^{k+1} = -\phi_{i+1,j}^k - \phi_{i-1,j}^k + 5\phi_{i,j}^k - \phi_{i,j+1}^k - \phi_{i,j-1}^k + b_{i,j}} \tag{4.69}$$

As one can observe that the matrix $\boldsymbol{A}$ does not need to be stored to carry out the desired iterations. The stability of the method is deduced by first determining the eigenvalues of the gain matrix. Such analysis and the following ones are carried out for homogeneous Dirichlet conditions, which yield analytical results. The eigenvalues of the Richardson gain matrix $\boldsymbol{G}_R$ are

$$\lambda(\boldsymbol{G}_R) = \lambda(\boldsymbol{I} - \boldsymbol{A}) = 1 - \lambda_{nm}^A \text{ where } 1 \leq n, m \leq N - 1 \tag{4.70}$$

$$= 1 + 4\left[\sin^2(\frac{n\pi}{2N}) + \sin^2(\frac{m\pi}{2N})\right]. \tag{4.71}$$

The corresponding spectral radius is then

$$\boxed{\rho(G_R) \approx |\lambda|_{max} = 5 > 1}, \tag{4.72}$$

and we can deduce that the **Richardson method is unstable when applied to the steady heat equation**.

4.4.2 Jacobi method

A simple method, stable but slow when applied to Laplace equation

In the Jacobi method, the matrix splitting is carried out with the diagonal part of $\boldsymbol{A}$: $\boldsymbol{A}_1 = \boldsymbol{D}$ and $\boldsymbol{A}_2 = \boldsymbol{D} - \boldsymbol{A} = \boldsymbol{E} + \boldsymbol{F}$. This choice yields a latrix $\boldsymbol{A}_1$ that is easily inversed since

$$D_{i,j}^{-1} = \begin{cases} 1/A_{i,j} \text{ if } i = j \\ 0 \text{ otherwise} \end{cases} \tag{4.73}$$

The corresponding iterative method reads

$$\boldsymbol{D}\boldsymbol{x}_{k+1} = (\boldsymbol{D} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b} \tag{4.74}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{D}^{-1}(\boldsymbol{D} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{D}^{-1}\boldsymbol{b}, \tag{4.75}$$

yielding the Jacobi iterative method:

Jacobi method (Generic)

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{D}^{-1}\boldsymbol{b} \tag{4.76}$$

The corresponding gain matrix is then $\boldsymbol{G}_J = (\boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A})$.

Application to the 2D Poisson problem. Following the Jacobi method, its application to any linear system, and here specifically to the discretized Poisson equation, consists in first identifying the diagonal term in each linear equation,

$$\phi_{i+1,j} + \phi_{i-1,j} \underbrace{-4\phi_{i,j}}_{\text{Diagonal term}} + \phi_{i,j+1} + \phi_{i,j-1} = b_{i,j}, \tag{4.77}$$

and leaving it on the left-hand side evaluated at the new iterate $k+1$: Jacobi:

$$\underbrace{-4\phi_{i,j}^{k+1}}_{\text{diagonal term on LHS}} = -(\phi_{i+1,j}^{k} + \phi_{i-1,j}^{k} + \phi_{i,j+1}^{k} + \phi_{i,j-1}^{k}) + b_{i,j}. \tag{4.78}$$

The Jacobi method applied to the 2D poisson problem then reads

Jacobi method (2D Poisson)

$$\phi_{i,j}^{k+1} = \frac{1}{4}(\phi_{i+1,j}^{k} + \phi_{i-1,j}^{k} + \phi_{i,j+1}^{k} + \phi_{i,j-1}^{k}) - \frac{1}{4}b_{i,j} \tag{4.79}$$

Once again, the matrix A does not need to be stored. With homogeneous Dirichlet boundary conditions, the diagonal entries of $\boldsymbol{A}$ are equal to -4. The gain matrix of the Jacobi method is then

$$\boldsymbol{G}_J = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A} = \boldsymbol{I} + \frac{1}{4}\boldsymbol{A}. \tag{4.80}$$

The corresponding eigenvalues are

$$\lambda(G_J) = 1 + \frac{1}{4}\lambda_{nm}^{A} \tag{4.81}$$

$$\lambda(G_J) = \frac{1}{2}[cos(\frac{n\pi}{N}) + cos(\frac{m\pi}{M})] \text{ for } 1 \leq n, m \leq N-1, M-1 \tag{4.82}$$

For each eigenvalue, $|\lambda(G_J)| < 1$, hence

$$\boxed{|\rho(\boldsymbol{G}_J)| < 1}, \tag{4.83}$$

showing that the **Jacobi method converges when applied to the Poisson problem**. The rate of convergence is controlled by $\rho(G_J) = |\lambda|_{max}$:

$$|\lambda|_{max} = |\lambda_{11}| = \frac{1}{2}[cos(\frac{\pi}{N}) + cos(\frac{\pi}{M})] \tag{4.84}$$

If $N \gg 1$ and $M \gg$,

$$|\lambda|_{max} \simeq \frac{1}{2}(1 - \frac{\pi^2}{2M^2} + 1 - \frac{\pi^2}{2N^2} + ...) \tag{4.85}$$

$$|\lambda|_{max} = 1 - \frac{\pi^2}{4}(\frac{1}{M^2} + \frac{1}{N^2}) \xrightarrow[k\to\infty]{} 1 \tag{4.86}$$

The spectral radius tends to one as the N increases. The **Jacobi method converges then very slowly when applied to practical cases**. However, this method is a starting point for more advanced iterative schemes.

4.4.3 Gauss-Seidel methods

An improvement of the Jacobi method

4.4.3.1 Standard Gauss-Seidel

Let's have a look at the implementation of the Jacobi method:

Loop i=1 ,N - 1
  Loop j=1 , M - 1

$$\phi_{i,j}^{k+1} = \frac{1}{4}(\phi_{i+1,j}^{k} + \phi_{i,j+1}^{k} + \underbrace{\phi_{i-1,j}^{k} + \phi_{i,j-1}^{k}}_{\text{already available at } k+1}) - \frac{1}{4}b_{i,j}$$

While looping for increasing indexes i and j, we notice that the values $\phi_{i-1,j}^{k+1}$ and $\phi_{i,j-1}^{k+1}$ have already been computed when updating $\phi_{i,j}^{k+1}$. If these values are used as soon as they are available instead of waiting for the next iteration step, one can expect to accelerate the convergence of the sequence. This results in the Gauss-Seidel method which, for the 2D Poisson problem, is written as

Gauss-Seidel method (2D Poisson)

$$\phi_{i,j}^{k+1} = \frac{1}{4}(\phi_{i-1,j}^{k+1} + \phi_{i,j-1}^{k+1} + \phi_{i+1,j}^{k} + \phi_{i,j+1}^{k}) - \frac{1}{4}b_{i,j} \tag{4.87}$$

The corresponding generic Gauss-Seidel method consists in choosing the splitting $\boldsymbol{A}_1 = \boldsymbol{D} - \boldsymbol{E}$, the lower triangular and diagonal parts of $\boldsymbol{A}$, and $\boldsymbol{A}_2 = \boldsymbol{A}_1 - \boldsymbol{A} = \boldsymbol{F}$, the upper part of $\boldsymbol{A}$. Hence,

Gauss-Seidel method (Generic)

$$\boldsymbol{x}_{k+1} = (\boldsymbol{D} - \boldsymbol{E})^{-1}\boldsymbol{F}\boldsymbol{x}_k + (\boldsymbol{D} - \boldsymbol{E})^{-1}\boldsymbol{b}, \tag{4.88}$$

and the gain matrix of the Gauss-Seidel method is $\boldsymbol{G}_{GS} = (\boldsymbol{D} - \boldsymbol{E})^{-1}\boldsymbol{F}$

Convergence in 2D Poisson problem With homogeneous Dirichlet conditions, one can show that the eigenvalues $\boldsymbol{G}_{GS}$ are related the the ones of the Jacobi gain matrix:

$$\lambda(\boldsymbol{G}_{GS}) = [\lambda(\boldsymbol{G}_J)]^2 \tag{4.89}$$

Then, for $1 \leq n, m \leq N-1, M-1$, we have

$$\lambda(\boldsymbol{G}_{GS}) = \frac{1}{4}\left[\cos(\frac{n\pi}{N}) + \cos(\frac{m\pi}{M})\right]^2 \tag{4.90}$$

Therefore, the **Gauss-Seidel method converges when applied to the 2D Poisson problem. Regarding its rate of convergence, it is twice faster than Jacobi** to reach the same error threshold. However, similarly to the Jacobi method, it remains penalized for large N, M values as the spectral radius of the gain matrix gets closer to unity.

4.4.3.2 Backward Gauss-Seidel

An alternative iterative method can be built: The backward Gauss-Seidel method is derived by considering loops of decreasing values of i and j,

Loop i = N - 1, 1
  Loop j = M - 1, 1
    $\phi_{i,j}^{k+1} = \frac{1}{4}(\phi_{i+1,j}^{k} + \phi_{i,j+1}^{k} + \phi_{i-1,j}^{k} + \phi_{i,j-1}^{k}) - \frac{1}{4}b_{i,j}$

This time, the values $\phi_{i+1,j}^{k+1}$ and $\phi_{i,j+1}^{k+1}$ are the ones that are already available and the backward Gauss-Seidel reads

Backward Gauss-Seidel method (2D Poisson)

$$\phi_{i,j}^{k+1} = \frac{1}{4}(\phi_{i+1,j}^{k+1} + \phi_{i,j+1}^{k+1} + \phi_{i-1,j}^{k} + \phi_{i,j-1}^{k}) - \frac{1}{4}b_{i,j} \tag{4.91}$$

The generic Backward Gauss-Seidel corresponds to the splitting $\boldsymbol{A}_1 = \boldsymbol{D} - \boldsymbol{F}$, the diagonal and upper triangular part of $\boldsymbol{A}$, and $\boldsymbol{A}_2 = \boldsymbol{A}_1 - \boldsymbol{A} = \boldsymbol{E}$, the lower part of $\boldsymbol{A}$:

Backward Gauss-Seidel method (Generic)

$$\boldsymbol{x}_{k+1} = (\boldsymbol{D} - \boldsymbol{F})^{-1}\boldsymbol{E}\boldsymbol{x}_k + (\boldsymbol{D} - \boldsymbol{F})^{-1}\boldsymbol{b}, \tag{4.92}$$

The properties of the Backward Gauss-Seidel are identical to the regular Gauss-Seidel method.

4.4.3.3 Symmetric Gauss-Seidel

For a better repartition of numerical errors, one can alternate standard and backward Gauss-Seidel methods as

Symmeric Gauss-Seidel method

$$(\boldsymbol{D} - \boldsymbol{E})\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{b} \tag{4.93}$$

$$(\boldsymbol{D} - \boldsymbol{F})\boldsymbol{x}_{k+1} = \boldsymbol{E}\boldsymbol{x}_{k+\frac{1}{2}} + \boldsymbol{b} \tag{4.94}$$

This results with the gain matrix

$$\boldsymbol{G}_{SGS} = (\boldsymbol{D} - \boldsymbol{F})^{-1}\boldsymbol{E}(\boldsymbol{D} - \boldsymbol{E})^{-1}\boldsymbol{F} \tag{4.95}$$

The better repartition of numerical error thanks to the alternation in sweep directions achieves a convergence rate that is twice faster compared to the standard Gauss-Seidel method. However, since the cost per iteration is also twice larger, the global computational efficiency is identical to the standard method. The method remains nonetheless interesting as a preconditioning technique (this concept will be addressed in the next sections) because it leaves the preconditioned system symmetric, in opposition to standard Gauss-Seidel.

4.5 Over-relaxation methods

Jacobi (1850) and Gauss-Seidel (1874) methods date from the 19^{th} century, when computers did not exist and calculations where performed by hand. Over-relaxation methods were developed on the 1950s with the **general goal of accelerating convergence by minimizing the spectral radius** $\rho(\boldsymbol{A}_1^{-1}\boldsymbol{A}_2)$.
The principle of over-relaxation is to enhance the correction $\Delta\boldsymbol{x}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ determined by a retained approach. This correction is then amplified by a factor ω:

$$\Delta\boldsymbol{x}_k = \omega\overline{\Delta\boldsymbol{x}_k}, \tag{4.96}$$

where $\overline{\Delta\boldsymbol{x}_k} = \overline{\boldsymbol{x}_{k+1}} - \boldsymbol{x}_k$ and $\overline{\boldsymbol{x}_{k+1}}$ is the updated iterated returned by the unaccelerated iterative method. ω **is called the over-relaxation coefficient if** $\omega > 1$ **or the under-relaxation coefficient if** $\omega < 1$**, respectively.** The over-relaxation of an iterative methods can then be written as

Over-Relaxation formula

$$\boldsymbol{x}_{k+1} = \omega\overline{\boldsymbol{x}_{k+1}} + (1 - \omega)\boldsymbol{x}_k, \tag{4.97}$$

where ω is a free parameter to optimize. For the previous methods detailed before (Richardson, Jacobi, Gauss-Seidel), only the over-relaxation of the Gauss-Seidel method, named *Successive Over-Relaxation* (SOR), results in a significant enhancement. Optimal over-relaxation of the Richardson and Jacobi methods actually yield back to the original Jacobi method.

4.5.1 Richardson over-relaxation

The basic Richardson method is

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}, \tag{4.98}$$

which is unstable when applied to the 2D Poisson problem. The over-relaxed version, yield a *modified* Richardson method, given by

$$\boldsymbol{x}_{k+1} = \omega[(\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}] + (1 - \omega)\boldsymbol{x}_k \tag{4.99}$$

or

Over-relaxed Richardson

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \omega \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}\omega, \tag{4.100}$$

which corresponds to the gain matrix $\boldsymbol{G}_R(\omega) = \boldsymbol{I} - \omega \boldsymbol{A}$. The corresponding eigenvalues are then given by

$$\lambda_R(\omega) = 1 - \omega \lambda_{nm}^A \tag{4.101}$$

2D Poisson problem. In this specific problem wiht homogeneous Dirichlet conditions, one finds

$$\lambda_R(\omega) = 1 + 4\omega \left(\sin^2(\frac{n\pi}{2N}) + \sin^2(\frac{m\pi}{2M}) \right). \tag{4.102}$$

The relaxation scheme is clearly unstable for any $\omega > 0$. ω must then be negative and the iterations is carried out backwards. The magnitude of ω remains limited, since convergence requires $|\lambda_R(\omega)| < 1$, therefore

$$\boxed{\omega < 0 \text{ and } |\omega| \leq \frac{1}{4}} \tag{4.103}$$

The modified Richardson method performs then backward iterations with a relaxation coefficient smaller than 1 in magnitude. The method is then **under-relaxed** to enforce stability. The optimal choice minimizes all eigenvalues $|\lambda_R(\omega)| = |1 - \omega \lambda_{nm}^A|$ to determine the minimal spectral radius yielding the highest convergence rate. **The optimal relaxation coefficient is then** $|\omega| = \frac{1}{4}$. **This results in nothing else but the Jacobi method applied to the same Poisson problem!**

Remark

The negative sign of ω is due to the forward iterations being always unstable with the standard Richardson method. The correct direction of iterations could have been obtained with a different starting point

- *for the matrix splitting conventional definition. Defining $\boldsymbol{A} = \boldsymbol{A}_1 + \boldsymbol{A}_2$, with $\boldsymbol{A}_1 = -\boldsymbol{I}$ and $\boldsymbol{A}_2 = \boldsymbol{I} + \boldsymbol{A}$ would have given*

$$-\boldsymbol{A}_1 \boldsymbol{x}_{k+1} = \boldsymbol{A}_2 \boldsymbol{x}_k - \boldsymbol{b} \tag{4.104}$$
$$\boldsymbol{x}_{k+1} = -\boldsymbol{A}_1^{-1} A_2 \boldsymbol{x}_k + \boldsymbol{A}_1^{-1} \boldsymbol{b} \tag{4.105}$$
$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} + \boldsymbol{A})\boldsymbol{x}_k - \boldsymbol{b} \tag{4.106}$$
$$\tag{4.107}$$

- *or for the conventional fixed-point method that also defined the Richardson method. Considering instead*

$$\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = 0 \Rightarrow \boldsymbol{x} = \boldsymbol{x} + (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}), \tag{4.108}$$

would have given similarly

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} + \boldsymbol{A})\boldsymbol{x}_k - \boldsymbol{b} \tag{4.109}$$

Nevertheless, without under-relaxation the correctly oriented Richardson method remains unstable in the 2D Poisson problem.

4.5.2 Jacobi Over-relaxation

The standard Jacobi method is written as:

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{D}^{-1}A)\boldsymbol{x}_k + \boldsymbol{D}^{-1}\boldsymbol{b} \tag{4.110}$$

and its over-relaxed variant reads

$$\boldsymbol{x}_{k+1} = \omega[(I - D^{-1}A)\boldsymbol{x}_k + D^{-1}b] + (1-\omega)\boldsymbol{x}_k \tag{4.111}$$
$$\tag{4.112}$$

or

Over-relaxed Jacobi

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \omega\boldsymbol{D}^{-1}\boldsymbol{A})\boldsymbol{x}_k + \omega\boldsymbol{D}^{-1}\boldsymbol{b} \tag{4.113}$$

where the corresponding gain matrix can be expressed from the Jacobi gain matrix: $\boldsymbol{G}_J(\omega) = \omega\boldsymbol{G}_J + (1-\omega)\boldsymbol{I}$.

Stability. The eigen values of the gain matrix are And for the stability analysis:

$$\lambda^J(\omega) = \omega\lambda^J + (1-\omega). \tag{4.114}$$

The spectral radius of the gain matrix is then bounded as

$$\rho(\boldsymbol{G}_J(\omega)) \leq |1-\omega| + \omega|\lambda^J|. \tag{4.115}$$

Requiring $\rho(\boldsymbol{G}_J(\omega)) < 1$ gives the following range of possible values for the over-relaxation factor ω:

$$\boxed{0 \leq \omega \leq \frac{2}{1+\rho(\boldsymbol{G}_J)}} \tag{4.116}$$

Optimality. The optimal value ω is obtained by minimizing the method's specral radius. Let's consider a particular case where the eigenvalues of the considered linear system yields positive and negative value for the corresponding gain matrix of the standard Jacobi method. This condition is met for example in the Poisson problem. Let's denote by λ^J_{min} the lowest negative eigenvalue and by λ^J_{max} the highest positive eigenvalue. One can then bound the eigenvalues of the over-relaxed Jacobi method as

$$\omega\lambda^J_{min} + (1-\omega) \leq \lambda_J(\omega) \leq \omega\lambda^J_{max} + 1 - \omega. \tag{4.117}$$

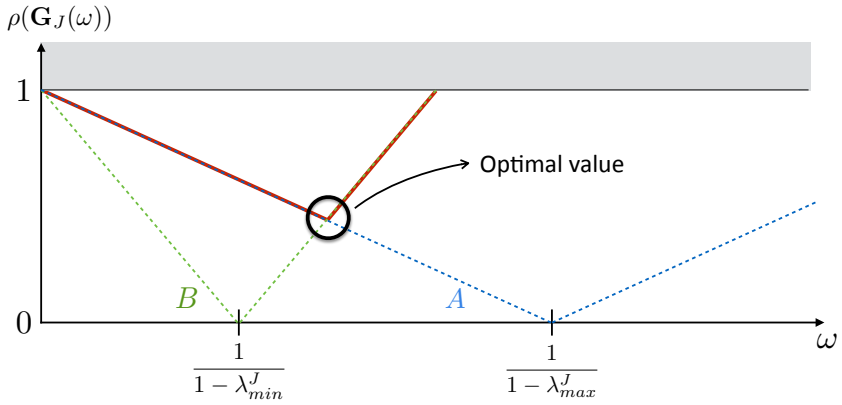


Figure 4.4 : *Determination of optimal over-relaxation coefficient for the Jacobi method*

Since those bounds are actually reached, the spectral radius of the over-relaxed method is

$$\rho(\boldsymbol{G}_J(\omega)) = \text{Max}(\underbrace{|\lambda^J_{max} - 1)\omega + 1|}_{A}, \underbrace{|(-\lambda^J_{min})\omega - 1|}_{B}) \tag{4.118}$$

The two curves corresponding to terms A and B are plotted as function of ω in Fig. 4.4.
The optimal value corresponds to the equality of both terms A and B:

$$-1 + \omega_{opt}(1 - \lambda^J_{min}) = 1 - \omega_{opt}(1 - \lambda^J_{max} \tag{4.119}$$

Therefore,

$$\boxed{\omega_{opt} = \frac{2}{2 - (\lambda^J_{min} + \lambda^J_{max})}} \tag{4.120}$$

2D Poisson problem. In this configuration with homogeneous boundary conditions, $\lambda^J_{max} = -\lambda^J_{min} = \cos\left(\frac{\pi}{M}\right)$. Therefore, Example: 2d heat equation

$$\boxed{\omega_{opt} = 1} \tag{4.121}$$

The optimal over-relaxed Jacobi method is the original method itself!

4.5.3 Gauss-Seidel Over-relaxation, a.k.a Successive Over-Relaxation (SOR)

This method is also known as Successive Over-Relaxation (SOR) and will present a considerable benefit compared to the two previous attempts with Richardson and Jacobi methods. For Gauss-Seidel method, we have:

$$\boldsymbol{x}_{k+1} = (\boldsymbol{D} - \boldsymbol{E})^{-1}\boldsymbol{F}\boldsymbol{x}_k + (\boldsymbol{D} - \boldsymbol{E})^{-1}\boldsymbol{b},$$

which can alternatively be written as

$$(\boldsymbol{D} - \boldsymbol{E})\boldsymbol{x}_{k+1} = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{b}$$

or

$$\boldsymbol{D}\boldsymbol{x}_{k+1} = \underbrace{\boldsymbol{E}\boldsymbol{x}_{k+1}}_{\text{Known values}} + \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{b} \tag{4.122}$$

The latter expression highlight the new components of $\boldsymbol{x}_{k+1}$ being computed on the left-hand side and the already known ones used on the right-hand side. The over-relaxation of the Gauss-Seidel method is given by

$$\boldsymbol{x}_{k+1} = \omega\overline{\boldsymbol{x}_{k+1}} + (1-\omega)\boldsymbol{x}_k \tag{4.123}$$

$$\text{with } \boldsymbol{D}\overline{\boldsymbol{x}_{k+1}} = \boldsymbol{E}\boldsymbol{x}_{k+1} + \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{b} \tag{4.124}$$

Pay attention to the last expression where the right-hand side is evaluated with the known components of the over-relaxed solution $\boldsymbol{x}_{k+1}$ and not $\overline{\boldsymbol{x}_{k+1}}$. Then,

$$\boldsymbol{D}\boldsymbol{x}_{k+1} = \omega(\boldsymbol{E}\boldsymbol{x}_{k+1} + \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{b}) + (1-\omega)\boldsymbol{D}\boldsymbol{x}_k, \tag{4.125}$$

and the SOR method is obtained by solving the linear system:

SOR (Generic)

$$(\boldsymbol{D} - \omega\boldsymbol{E})\boldsymbol{x}_{k+1} = [(1-\omega)\boldsymbol{D} + \omega\boldsymbol{F}]\boldsymbol{x}_k + \omega\boldsymbol{b}. \tag{4.126}$$

The matrix on the left-had side $\boldsymbol{D} - \omega\boldsymbol{E}$ is triangular and yields a costless linear system to solve through forward substitution. The corresponding gain matrix is

$$\boldsymbol{G}_{SOR}(\omega) = (\boldsymbol{D} - \omega\boldsymbol{E})^{-1}[(1-\omega)\boldsymbol{D} + \omega\boldsymbol{F}] \tag{4.127}$$

$$\boldsymbol{G}_{SOR}(\omega) = \boldsymbol{I} - \omega(\boldsymbol{D} - \omega\boldsymbol{E})^{-1}\boldsymbol{A}. \tag{4.128}$$

The SOR method corresponds to the following matrix splitting:

$$\boldsymbol{A} = \underbrace{\omega^{-1}(\boldsymbol{D} - \omega\boldsymbol{E})}_{\boldsymbol{A}_1} - \underbrace{\omega^{-1}[\omega\boldsymbol{F} + (1-\omega)\boldsymbol{D}]}_{\boldsymbol{A}_2} \tag{4.129}$$

One can demonstrate the following necessary stability condition:

$$\boxed{0 \leq \omega < 2} \tag{4.130}$$

To accelerate convergence through over-relaxation ($\omega > 1$), the range of practical interest is $1 < \omega < 2$.

2D Poisson problem Let's recall the standard Gauss-Seidel formula applied to the 2D Poisson problem:

$$\phi_{ij}^{k+1} = \frac{1}{4}(\phi_{i-1,j}^{k+1} + \phi_{i+1,j}^{k} + \phi_{i,j+1}^{k} + \phi_{i,j-1}^{k+1}) - \frac{1}{4}b_{i,j} \tag{4.131}$$

The SOR formula follows from

$$\begin{cases} \overline{\phi_{i,j}^{k+1}} = \dfrac{1}{4}(\phi_{i-1,j}^{k+1} + \phi_{i+1,j}^{k} + \phi_{i,j+1}^{k} + \phi_{i,j-1}^{k+1}) - \dfrac{1}{4}b_{i,j} \\ \phi_{i,j}^{k+1} = (1-\omega)\phi_{i,j}^{k} + \omega\overline{\phi_{i,j}^{k+1}} \end{cases} \tag{4.132}$$

or

SOR (2D Poisson)

$$\phi_{i,j}^{k+1} = (1-\omega)\phi_{i,j}^{k} + \omega\left[\frac{1}{4}(\phi_{i-1,j}^{k+1} + \phi_{i+1,j}^{k} + \phi_{i,j+1}^{k} + \phi_{i,j-1}^{k+1}) - \frac{1}{4}b_{i,j}\right] \tag{4.133}$$

When applying SOR to the studied 2D Poisson problem, the eigenvalues of the SOR gain matrix are known:

$$\lambda^{SOR}(\omega) = 1 - \omega + \omega\lambda^{SOR}(\omega)^{\frac{1}{2}}\lambda^{J} \tag{4.134}$$

$$\Rightarrow \lambda^{SOR}(\omega)^{1/2} = \frac{1}{2}(\omega\lambda^{J} \pm \sqrt{\omega^2\lambda_J^2 + 4(1-\omega)}), \tag{4.135}$$

where λ^J are the eigenvalues of the Jacobi gain matrix applied to the same problem.

Optimality (Poisson problem) . The optimal value of the over-relaxation coefficient is obtained by minimizing the spectral radius $\rho(\boldsymbol{G}_{SOR}) = |\lambda^{SOR}(\omega)|_{max}$. Since there is no solution to $\frac{d\rho(\boldsymbol{G}_{SOR})}{d\omega} = 0$, there is no local minimum and a the global one must be determined as illustrated in Fig. 4.5. The optimal point can be shown to be

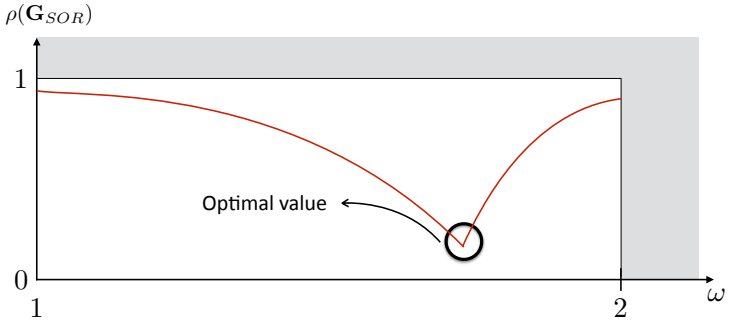


Figure 4.5 : *Determination of optimal over-relaxation coefficient for the SOR method*

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho(\boldsymbol{G}_J)^2}}, \tag{4.136}$$

where $\rho(\boldsymbol{G}_J)$ is the spectral radius of the Jacobi method. Selecting the optimal over-relaxation coefficient, the SOR method is much faster then the Jacobi and Gauss-Seidel methods. For the studied Poisson problem and $N = M$, a more practical formula can be obtained,

$$\rho(\boldsymbol{G}_J) = \cos\left(\frac{\pi}{N}\right) \rightarrow \omega_{opt} = \frac{2}{1 + \sin\left(\frac{\pi}{N}\right)} \tag{4.137}$$

For $N \gg 1$, the following useful approximation is used

$$\boxed{\omega_{opt} \approx 2\left(1 - \frac{\pi}{N} + \frac{\pi^2}{N^2} + \cdots\right)} \tag{4.138}$$

The optimal value is then slightly lower than 2. **For the general linear systems, there is no formula for** ω_{opt}, which must then be roughly estimated by trying different values.

Remark

- *A backward SOR can also be derived. It shares the same properties as the standard SOR method.*
- *Based on the symmetric Gauss-Seidel method, a Symetric SOR (SSOR) can be obtained as well. Similarly to the symmetric Gauss-Seidel method, its convergence rate is twice faster but, with a twofold cost per step, the global computational efficiency remains the same. The SSOR is nonetheless appreciated for its better error balancing and is a good symmetric preconditioning method.*

4.6 Matrix splitting interpreted as a Richardson method

In this section, all matrix splitting methods considered so far are shown to be simply linked to the Richardson method.

4.6.1 Preconditioning

When the linear system $\boldsymbol{Ax} = \boldsymbol{b}$ is ill-conditioned, *i.e.* when the condition number $\kappa(\boldsymbol{A}) = \frac{|\lambda^A_{max}|}{|\lambda^A_{min}|}$) is large, all numerical methods (direct or iterative) are penalized. A very common practice is then to precondition the system, which consists in applying a linear change of variables.

Right preconditioning

The linear system is transformed as

$$\boldsymbol{Ax} = \boldsymbol{b} \rightarrow \boldsymbol{AK}^{-1}\boldsymbol{Kx} = \boldsymbol{b}, \tag{4.139}$$

where $\boldsymbol{K}$ is the preconditioning matrix. Setting,

$$\begin{cases} \tilde{\boldsymbol{A}} = \boldsymbol{AK}^{-1} \\ \boldsymbol{y} = \boldsymbol{Kx} \end{cases} \tag{4.140}$$

yields the modified linear system:

$$\tilde{\boldsymbol{A}}\boldsymbol{y} = \boldsymbol{b} \tag{4.141}$$

$$\text{and } \boldsymbol{x} = \boldsymbol{K}^{-1}\boldsymbol{y} \tag{4.142}$$

Given the latter equation which determines $\boldsymbol{x}$, the preconditioning matrix must be yield a linear system, $\boldsymbol{Kx} = \boldsymbol{y}$, that is cheaply solved. Additionally, the retained matrix $\boldsymbol{K}$ must of course solve the initial problem, meaning that the condition number of the new system $\tilde{\boldsymbol{A}}\boldsymbol{y} = \boldsymbol{b}$ must be significantly reduced.

Left preconditioning

Alternatively, right preconditioning can be replaced by or combined with left preconditioning that considers

$$\boldsymbol{Ax} = \boldsymbol{b} \rightarrow \boldsymbol{K}^{-1}\boldsymbol{Ax} = \boldsymbol{K}^{-1}\boldsymbol{b} \tag{4.143}$$

Setting,

$$\begin{cases} \tilde{\boldsymbol{A}} = \boldsymbol{K}^{-1}\boldsymbol{A} \\ \tilde{\boldsymbol{b}} = \boldsymbol{K}^{-1}\boldsymbol{b} \end{cases} \tag{4.144}$$

yields a new linear system:

$$\tilde{\boldsymbol{A}}\boldsymbol{x} = \tilde{\boldsymbol{b}} \tag{4.145}$$

4.6.2 Matrix splitting

All the methods seen so far can be expressed as a matrix splitting, $\boldsymbol{A} = \boldsymbol{A}_1 - \boldsymbol{A}_2$, yielding a common iterative process $\boldsymbol{A}_1\boldsymbol{x}_{k+1} = \boldsymbol{A}_2\boldsymbol{x}_k + \boldsymbol{b}$. These iterations can be written as

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_1^{-1}\boldsymbol{A}_2\boldsymbol{x}_k + \boldsymbol{A}_1^{-1}\boldsymbol{b} \tag{4.146}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_1^{-1}(\boldsymbol{A}_1 - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{A}_1^{-1}\boldsymbol{b} \tag{4.147}$$

$$\boxed{\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A}_1^{-1}\boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{A}_1^{-1}\boldsymbol{b}} \tag{4.148}$$

This expression is nothing else than the Richardson method, $\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \tilde{\boldsymbol{A}})\boldsymbol{x}_k + \tilde{\boldsymbol{b}}$, applied to the preconditioned system $\tilde{\boldsymbol{A}}\boldsymbol{x} = \tilde{\boldsymbol{b}}$: $\boldsymbol{A}_1^{-1}\boldsymbol{A} = \boldsymbol{A}_1^{-1}\boldsymbol{b}$. Therefore, **all previous methods are identical to a Richardson iterative method applied to the preconditioned with the preconditioning matrix $\boldsymbol{K} = \boldsymbol{A}_1$.**

Each matrix splitting method is then equivalently identified as a specific preconditionner with different choices for $\boldsymbol{K}$:

- Jacobi preconditioner : $\boldsymbol{K} = \boldsymbol{D}$
- Gauss-Seidel preconditioner : $\boldsymbol{K} = \boldsymbol{D} - \boldsymbol{E}$
- Symmetric Gauss-Seidel preconditioner : $\boldsymbol{K} = (\boldsymbol{D} - \boldsymbol{E})\boldsymbol{D}^{-1}(\boldsymbol{D} - \boldsymbol{F})$
- SOR preconditioner : $\boldsymbol{K} = \boldsymbol{D} - \omega\boldsymbol{E}$
- SSOR preconditioner : $\boldsymbol{K} = (\boldsymbol{D} - \omega\boldsymbol{E})\boldsymbol{D}^{-1}(\boldsymbol{D} - \omega\boldsymbol{F})$

Since the Richardson method is identical to the simple fixed point method applied to the linear system, every methods considered so far are preconditioned fixed point method. To progress further, one must improve the underlying Richardson method.

4.7 Towards Krylov methods

4.7.1 Residuals in the Richardson method and Krylov subspaces

Given the conclusion in the previous section, once $\boldsymbol{A}$ is preconditioned, the only method at our disposal is the Richardson method,

$$\boldsymbol{x}_{k+1} = (I - A)\boldsymbol{x}_k + \boldsymbol{b}$$

In terms of the iteration residual, $\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$, one can write

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{r}^k$$

Multiplying by $-\boldsymbol{A}$ and adding $\boldsymbol{b}$ yields the following evolution of the residuals in the Richardson method:

$$\boxed{\boldsymbol{r}^{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{r}^k.} \tag{4.149}$$

Hence,

$$\left\|\boldsymbol{r}^{k+1}\right\| \leq \|\boldsymbol{I} - \boldsymbol{A}\| \left\|\boldsymbol{r}^k\right\|. \tag{4.150}$$

The iterative method converges to the solution, *i.e.* $\boldsymbol{r_k} \to 0$, if

$$\|\boldsymbol{I} - \boldsymbol{A}\| \leq 1 \tag{4.151}$$

All the methods seen so far (=preconditioned Richardson) have modified $\boldsymbol{A}$ so that $\|I - A\| < 1$ is satisfied. This restriction to have all eigenvalues contained in the unit ball must be elevated to progress further. Let's first notice that the Richardson residuals can be expressed as powers $\boldsymbol{A}$ multiplied by the initial residual $\boldsymbol{r}_0$:

$$\boldsymbol{r}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{r}_k \to \boxed{\boldsymbol{r}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})^{k+1}\boldsymbol{r}_0 = P_{k+1}(\boldsymbol{A})\boldsymbol{r}_0}, \tag{4.152}$$

where $P_{k+1}(\boldsymbol{A})$ is a particular polynomial fo degree $k+1$. The iterate $\boldsymbol{x}_k$ can also be written as a function of a polynomial $Q_k(\boldsymbol{A})$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{r}_0 + \boldsymbol{r}_1 + \boldsymbol{r}_2 + ... + \boldsymbol{r}_k \tag{4.153}$$

$$\boxed{\boldsymbol{x}_{k+1} = \sum_{j+0}^{k} (\boldsymbol{I} - \boldsymbol{A})^j \boldsymbol{r}_0 = Q_k(\boldsymbol{A})\boldsymbol{r}_0} \tag{4.154}$$

The Richardson method writes then each iterate $\boldsymbol{x}_{k+1}$ as a specific linear combination of vectors $\boldsymbol{r}_0$, $\boldsymbol{A}\boldsymbol{r}_0, \cdots, \boldsymbol{A}^k\boldsymbol{r}_0$. The iterate $\boldsymbol{x}_{k+1}$ is then an element of the vectorial subspace of dimension $k+1$ generated or spanned by those vectors:

$$\boxed{\boldsymbol{x}_{k+1} \in \text{span}\{\boldsymbol{r}_0,\, \boldsymbol{A}\boldsymbol{r}_0,\, \cdots,\, \boldsymbol{A}^k\boldsymbol{r}_0\} \underset{def.}{=} \mathcal{K}_{k+1}(\boldsymbol{A}, \boldsymbol{r}_0)}, \tag{4.155}$$

where the defined vectorial space $\mathcal{K}_{k+1}(\boldsymbol{A}, \boldsymbol{r}_0)$ is called the $(k+1)$-dimensional Krylov subspace. In conclusion of this section, the **simple Richardson method yields particular successive estimates $\boldsymbol{x}_k$ that lie in Krylov subspace of increasing dimension**.

4.7.2 Improving the Richardson method

4.7.2.1 Damping properties Richardson

To understand how $\boldsymbol{A}$'s eigenvalues impact the residual convergence, let's express $\boldsymbol{r}^0$ on the eigenvectors $\boldsymbol{w}_j$ bases of $\boldsymbol{A}$:

$$\boldsymbol{r}_0 = \sum_{j=1}^{n} \gamma_j \boldsymbol{w}_j, \tag{4.156}$$

with the weights γ_j. Then,

$$\boldsymbol{r}_k = \boldsymbol{P}_k(\boldsymbol{A})\boldsymbol{r}_0 = \sum_{j=1}^{n} \gamma_j P_k(\lambda_j^A)\boldsymbol{w}_j \tag{4.157}$$

The decrease in residual $\boldsymbol{r}_k$ is then linked to the decrease of initial $\boldsymbol{r}_0$ components due to the damping of each eigenvalue by the polynomial $P_k(\lambda_j^A)$. The improvement of the fixed-point method, that is the Richardson method, is to build another method with better damping properties for P_k.

4.7.2.2 First improving step: Modified Richardson

Let's introduce an iteration relaxation parameter α_k:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k \tag{4.158}$$

$$\boxed{\boldsymbol{r}_{k+1} = (\boldsymbol{I} - \alpha_k \boldsymbol{A})\boldsymbol{r}_k} \tag{4.159}$$

The obtained formula is a *modified* Richardson method where the simple Richardson method is retrieved with $\alpha_k = 1$. The residual can still be expressed as a polynomial of $\boldsymbol{A}$,

$$\boldsymbol{r}_{k+1} = P_{k+1}(\boldsymbol{A})\boldsymbol{r}_0 \text{ with } P_k(\boldsymbol{A}) = \prod_{j=1}^{k}(\boldsymbol{I} - \alpha_j \boldsymbol{A}), \tag{4.160}$$

and the iterate $\boldsymbol{x}_{k+1}$ still lies in the Krylov subspace $\mathcal{K}_{k+1}(\boldsymbol{A}, \boldsymbol{r}_0)$. By introducing additional degrees of freedom α_j, the convergence of the method does not require $\|I - A\| < 1$ anymore. The obtained polynomial $P_k(\boldsymbol{A})$ is much more general and corresponds to a family of polynomials with real roots. Nevertheless, the coefficients α_k remain to be determined. The next step is then to determining an optimal set of coefficients.

4.7.2.3 Generalization idea: Krylov methods

The simple and modified Richardson method always yield an iterate $\boldsymbol{x}_{k+1} \in \mathcal{K}_{k+1}(\boldsymbol{A}, \boldsymbol{r}_0)$. Let's keep this rule and then propose a general method where $\boldsymbol{x}_{k+1}$ is now expressed from an arbitrary polynomial of degree k as

$$\boldsymbol{x}_{k+1} = Q_k(\boldsymbol{A})\boldsymbol{r}_0 \tag{4.161}$$

Assuming without loss of generality that $\boldsymbol{0} = 0 \rightarrow \boldsymbol{r}_0 = \boldsymbol{b}$, the corresponding residual is given by

$$\boldsymbol{r}_{k+1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A}\boldsymbol{Q}_k(\boldsymbol{A}))\boldsymbol{r}_0 = \tilde{P}_{k+1}(\boldsymbol{A})\boldsymbol{r}_0 \tag{4.162}$$

where $\tilde{P}_{k+1}$ is a general polynomial of degree $k+1$ fulfilling $\tilde{P}_{k+1}(0) = 1$. **This general method combined with an optimal choice of $\boldsymbol{x}_{k+1}$ in the Krylov subspace $\mathcal{K}_{k+1}(\boldsymbol{A}, \boldsymbol{r}_0)$ results in a so-called Krylov iterative method**. Each choice of an optimality condition yields a different Krylov method. The most classical ones are: **Conjugate Gradient, GMRES and Bi-CGStab**. When carrying out n iterations of the Krylov method where n is the total number of unknowns, the vectorial Krylov subspace then spans the whole $\mathbb{R}^n$ space and the method becomes exact, yielding the exact solution $\boldsymbol{x}^*$. In practice though, such methods are interrupted at an iteration k much sooner than n, making them placed in the family of iterative methods.

4.7.3 Conjugate Gradient method

The conjugate gradient method is the one of the best known iterative method to solve a symmetric positive definite linear system, which makes it one of the preferred method to solve elliptic PDEs such as Poisson equations [2]. The method is built upon a specific criterion of optimality where the residual $\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$ is set orthogonal to the current Krylov subspace $\mathcal{K}_k(\boldsymbol{A}, r_0)$ such that the error norm $\|\boldsymbol{x}_k - \boldsymbol{x}^*\|_A$ based on the positive matrix $\boldsymbol{A}$ is minimized. The obtained iterative formula is given below:

[2] In fact, it is the opposite matrix $-\boldsymbol{A}$ that is symmetric definite positive in the Poisson problem

Conjugate Gradient

Compute $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$.

For $k = 0, 1, \cdots$, until convergence **DO:**

$$\begin{aligned} \alpha_k &= \langle \boldsymbol{r}_k, \boldsymbol{r}_k \rangle / \langle \boldsymbol{A}\boldsymbol{p}_k, \boldsymbol{p}_k \rangle \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{p}_k \\ \beta_k &= \langle \boldsymbol{r}_{k+1}, \boldsymbol{r}_{k+1} \rangle / \langle \boldsymbol{r}_k, \boldsymbol{r}_k \rangle \\ \boldsymbol{p}_{k+1} &= \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k \end{aligned}$$

End

Let us remember that Krylov methods were derived in this section with the initial desire to make the simple Richardson method more efficient. It was assumed that the linear system was the original one or a preconditioned one. As for Richardson, Krylov methods are strongly impacted by preconditioning which makes them even more efficient. It has been shown that the preconditioned Richardson yielded the Jacobi, Gauss-Seidel ad SOR methods. Since the conjugate gradient methods tackles symmetric systems, the preconditioned system must remain symmetric, which invites to use symmetric preconditionners. The Jacobi or symmetric Gauss-Seidel preconditionner enable to improve further the convergence rate of the conjugate gradient method. The general implementation of the preconditionned conjugate gradient method follows:

Preconditioned Conjugate Gradient

Choose preconditioning matrix $\boldsymbol{K}$

Compute $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, $\boldsymbol{z}_0 = \boldsymbol{K}^{-1}\boldsymbol{r}_0$ and $\boldsymbol{p}_0 = \boldsymbol{z}_0$.

For $k = 0, 1, \cdots$, until convergence **DO:**

$$\begin{aligned} \alpha_k &= \langle \boldsymbol{r}_k, \boldsymbol{z}_k \rangle / \langle \boldsymbol{A}\boldsymbol{p}_k, \boldsymbol{p}_k \rangle \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{p}_k \\ \boldsymbol{z}_{k+1} &= \boldsymbol{K}^{-1}\boldsymbol{r}_{k+1} \\ \beta_k &= \langle \boldsymbol{r}_{k+1}, \boldsymbol{z}_{k+1} \rangle / \langle \boldsymbol{r}_k, \boldsymbol{z}_k \rangle \\ \boldsymbol{p}_{k+1} &= \boldsymbol{z}_{k+1} + \beta_k \boldsymbol{p}_k \end{aligned}$$

End

4.8 Comparison of methods

Solving a linear system of n unknowns, the different iterative methods require different number of iterations to reduce the initial error by several orders of magnitude. Their algorithmic complexity in shown in Tab. 4.8. One can clearly see the significant benefit by considering successively the SOR and Conjugate gradient methods. The multigrid method has not been detailed here and the readers

are forwarded to reference textbooks for details. Such methods are the most efficient on structured meshes and yield an unbeaten linear complexity.

Methods	Complexity
Jacobi	$\mathcal{O}(n^2)$
Gauss-Seidel	$\mathcal{O}(n^2)$
SOR	$\mathcal{O}(n^{3/2})$
Preconditioned Conjugate Gradient	$\mathcal{O}(n^{5/4})$
Multigrid	$\mathcal{O}(n^{3/2})$

Table 4.1 : *Comparison of iterative methods*

Cheat Sheet: Iterative methods to solve the 2D Poisson equation

Solving the linear system obtained by using 2[nd]-order centered difference formula for

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f(x, y),$$

yields the linear system

$$\phi_{i+1,j} + \phi_{i-1,j} - 4\phi_{i,j} + \phi_{i,j+1} + \phi_{i,j-1} = h^2 f_{ij}.$$

- **Jacobi method**

$$\phi_{i,j}^{k+1} = \frac{1}{4}(\phi_{i+1,j}^{k} + \phi_{i-1,j}^{k} + \phi_{i,j+1}^{k} + \phi_{i,j-1}^{k}) - \frac{1}{4}h^2 f_{i,j}$$

- **Gauss-Seidel method**

$$\phi_{i,j}^{k+1} = \frac{1}{4}(\phi_{i-1,j}^{k+1} + \phi_{i,j-1}^{k+1} + \phi_{i+1,j}^{k} + \phi_{i,j+1}^{k}) - \frac{1}{4}h^2 f_{i,j}$$

- **SOR method**

$$\phi_{i,j}^{k+1} = (1-\omega)\phi_{i,j}^{k} + \omega\left[\frac{1}{4}(\phi_{i-1,j}^{k+1} + \phi_{i+1,j}^{k} + \phi_{i,j+1}^{k} + \phi_{i,j-1}^{k+1}) - \frac{1}{4}h^2 f_{i,j}\right]$$

with $\omega_{opt} \approx 2\left(1 - \frac{\pi}{N} + \frac{\pi^2}{N^2} + \cdots\right)$

Chapter 5

Hyperbolic and Parabolic Partial Differential Equations

Contents

Compared to Elliptic PDEs, hyperbolic and parabolic PDEs have the advantage of not requiring to couple all points in the domain for their resolution. Given the propagative nature of both parabolic and hyperbolic PDEs, they have a "time"-like direction. From that statement, it is natural to consider a *step-by-step* approach. Numerical resolutions of parabolic and hyperbolic PDEs will then rely on *Time Marching* methods. Most of the numerical methods for Parabolic and Elliptic PDEs use different methods of discretization for time-direction and space-directions. The "global" numerical method and its properties is then a consequence of the properties of both spatial and time discretizations.

5.1 Examples of Hyperbolic and Parabolic PDEs

5.1.1 Parabolic PDEs

Parabolic PDEs share the following properties:

- they feature one preferred direction, along which they imply an infinite propagation of information,
- they require both initial conditions and boundary conditions for other.

Example :

- 1D unsteady heat equation:

$$\frac{\partial T}{\partial t} = a \frac{\partial^2 T}{\partial x^2} \tag{5.1}$$

- 2D unsteady heat equation:

$$\frac{\partial T}{\partial t} = a\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \tag{5.2}$$

- 3D unsteady heat equation:

$$\frac{\partial T}{\partial t} = a\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right) \tag{5.3}$$

- 1D advection-diffusion equation:

$$\frac{\partial Y_k}{\partial t} + c\frac{\partial Y_k}{\partial x} = D\frac{\partial^2 Y_k}{\partial x^2} \tag{5.4}$$

Remark

The presence of a time variable is not mandatory. A purely spatial example of elliptic equation is:

$$u\frac{\partial T}{\partial x} = a\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right), \tag{5.5}$$

which reduces, under the physical assumption $\frac{\partial^2 T}{\partial x^2} \ll 1$ *to the purely spatial parabolic equation:*

$$u\frac{\partial T}{\partial x} = a\frac{\partial^2 T}{\partial y^2}. \tag{5.6}$$

In this parabolic equation, the particular direction is x.

5.1.2 Hyperbolic PDEs

For hyperbolic PDEs:

- at least one propagation direction appears,
- propagation speeds are finite,
- Initial condition and boundary conditions are required on other directions (then, they are not mandatory everywhere).

Example :

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \qquad \text{1D advection equation} \tag{5.7}$$

$$\frac{\partial u}{\partial t} + c_1\frac{\partial u}{\partial x} + c_2\frac{\partial u}{\partial y} = 0 \qquad \text{2D advection equation} \tag{5.8}$$

$$c_1\frac{\partial u}{\partial x} + c_2\frac{\partial u}{\partial y} = 0 \qquad \text{2D steady advection} \tag{5.9}$$

5.2 Convergence of numerical schemes: The Lax theorem

When deriving a finite-difference approximation of a PDE, the objective is to ensure that the numerical solution will converge to the exact solution as the time and space increments tend to zero. The definition of convergence is the following:

Convergence

A Finite difference approximation $P_{\Delta t,\Delta x} v = f$ is said to be convergent with respect to the PDE $Pu = f$ if, for a given initial condition v_i^0 that tends to $u(t = 0, x_i)$ as Δx tends to 0, the solution v_i^n tends to the solution $u(t^n, x_i)$ as Δt and Δx tend to 0.

In order to get convergence, several ingredients are required. The first ingredient is the consistency, which ensures that the error between the finite difference approximation and the exact PDE tends to zero as space and time increments tend to zero:

Consistency

Considering the PDE $Pu = f$, where P is the partial derivative operator, a finite difference scheme $P_{\Delta t,\Delta x} v = f$ is consistent with $Pu = f$ if for any smooth function $\phi(t, x)$:

$$\|P\phi - P_{\Delta t,\Delta x}\phi\| = O(\Delta x^p) + O(\Delta t^q) \tag{5.10}$$

where $p, q > 0$ are the order of consistency in space and time respectively.

Example : If we consider a first order approximation of the advection equation using forward Euler and upwind differencing for space derivative, we get:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0 \rightarrow \frac{u_j^{n+1} - u_j^n}{\Delta t} + a\frac{u_j^n - u_{j-1}^n}{\Delta x} = 0 \tag{5.11}$$

Lookign at the truncation error made by finite difference approximation of time and space derivatives, we obtain:

$$u_j^{n+1} = u_j^n + \left.\frac{\partial u}{\partial t}\right|_j^n \Delta t + O(\Delta t^2) \rightarrow \left.\frac{\partial u}{\partial t}\right|_J^n = \frac{u_j^{n+1} - u_j^n}{\Delta t} + O(\Delta t) \tag{5.12}$$

$$u_{j-1}^n = u_j^n - \left.\frac{\partial u}{\partial x}\right|_j^n \Delta x + O(\Delta x^2) \rightarrow \left.\frac{\partial u}{\partial x}\right|_J^n = \frac{u_j^n - u_{j-1}^n}{\Delta t} + O(\Delta x) \tag{5.13}$$

Looking at the difference between the exact PDE and its approximation, we finally get:

$$\|Pu - P_{\Delta t,\Delta x} u\| = O(\Delta t) + O(\Delta x) \tag{5.14}$$

Then this scheme is consistent of first order in time and space with the advection equation.

The second ingredient is the stability, which ensures that the solution remains bounded for any time increment (methods to demonstrate stability of a numerical approximation are presented in Section 5.3):

Stability

A finite difference scheme $P_{\Delta t,\Delta x} v = f$ is said to be *stable* if $P^{-1}_{\Delta t,\Delta x}$ is uniformly bounded, i.e., if it exists a strictly positive constant C such that for $n \geq 0$:

$$\left\| P^{-1}_{\Delta t,\Delta x} \right\| \leq C \leq \infty$$

Then, if consistency and stability are satisfied, the Lax theorem states the convergence of the numerical approximation:

Lax theorem

If a finite difference approximation $P_{\Delta t,\Delta x} v = f$ is consistent at order p in time and order q in space with the PDE $Pu = f$, and is stable, the finite difference approximation is convergent at order p in time and order q in space.

Demonstration

To demonstrate convergence, we look at the difference between the numerical solution u and the exact solution v:

$$\begin{aligned} u - v &= P^{-1}_{\Delta t,\Delta x} P_{\Delta t,\Delta x} u - P^{-1}_{\Delta t,\Delta x}(P_{\Delta t,\Delta x} v) \\ &= P^{-1}_{\Delta t,\Delta x}(P_{\Delta t,\Delta x} u - Pu) - P^{-1}_{\Delta t,\Delta x}(P_{\Delta t,\Delta x} v - Pu) \\ &= P^{-1}_{\Delta t,\Delta x}(P_{\Delta t,\Delta x} u - Pu) - P^{-1}_{\Delta t,\Delta x}(f - f) \end{aligned}$$

Taking the norm of the difference, we get:

$$\begin{aligned} \|u - v\| &= \left\| P^{-1}_{\Delta t,\Delta x}(P_{\Delta t,\Delta x} u - Pu) \right\| \\ &\leq \left\| P^{-1}_{\Delta t,\Delta x} \right\| \|P_{\Delta t,\Delta x} u - Pu\| \end{aligned}$$

Using stability ($\left\| P^{-1}_{\Delta t,\Delta x} \right\| \leq C \leq \infty$) and consistency ($\|P\phi - P_{\Delta t,\Delta x}\phi\| = O(\Delta x^p) + O(\Delta t^q)$) , we finally get:

$$\begin{aligned} &\|u - v\| \leq C\left(O(\Delta x^p) + O(\Delta t^q)\right) \\ &\|u - v\| \underset{\Delta t,\Delta x \to 0}{\to} 0 \end{aligned}$$

Example : Let us consider the numerical solution of the 1D linear advection equation:

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

- the use of Forward Euler for the time part and Upwind scheme for the spatial part is consistent with the original equation at order 1 in time and space. Furthermore, as it will be demonstrated later in this chapter, this combination is also stable. As a consequence of Lax theorem, this scheme is thus convergent, as seen in Fig. 5.1.
- If we replaced the scheme for the spatial part by a centered scheme, we obtain an unstable scheme. Even if the resulting scheme is consistent at order 1 in time and 2 in space with the advection equation, it cannot converge to the exact solution as seen in Fig. 5.2.

5.3 Stability Analysis of discrete PDE

To achieve convergence, stability is mandatory and is thus an essential step in the derivation of a convergent (and thus useful) finite difference approximation of a PDE. There are several ways to analyse and demonstrate stability. Before presenting them, we first introduce two important numerical parameters: the **Fourier number** and the Courant-Friedrichs-Levy or **CFL number**.
The **Fourier Number** is related to diffusion equations and reads:

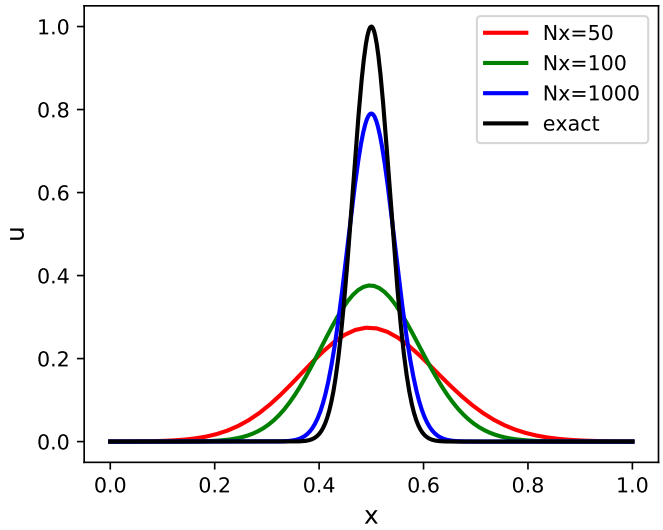


Figure 5.1 : *Solution of the 1d linear advection equation using Forward Euler+Upwind after one revolution: exact solution (black), and numerical solutions using* 50 *(red),* 100 *(green) and* 1000 *(blue) points.*

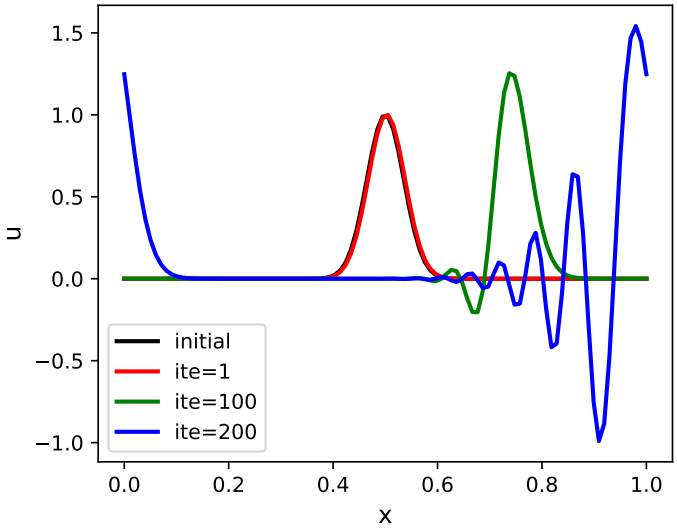


Figure 5.2 : *Solution of the 1d linear advection equation using Forward Euler+Centered: initial solution (black), and numerical solutions after* 1 *(red),* 100 *(green) and* 200 *(blue) iterations.*

Fourier Number

$$Fo = \frac{a\Delta t}{\Delta x^2}.$$

where a is the diffusion coefficient. It can be interpreted as the ratio of the time step Δt and the characteristic propagation time of information through a mesh cell ($\Delta x^2/a$). The **CFL number** is related to advection equations and reads

CFL Number

$$\mathcal{C} = \frac{|u|\,\Delta t}{\Delta x}.$$

where u is the advection velocity. It can be interpreted as the ratio of time step Δt and the propagation time of information through a mesh cell ($\Delta x/u$).

5.3.1 Fourier space analysis

To analyse the stability of a numerical scheme, the Fourier transform is very useful as it gives insight on the spectral properties on the scheme. The Fourier transform can either be used on the continuous equation or on the discrete data obtained by discretization. In the following, we first look at the exact solution of a reference advection-diffusion equation, and we then look its discrete counterpart.

5.3.1.1 Behaviour of the exact solution

Let us consider the advection-diffusion equation of a scalar u:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = d^2\frac{\partial^2 u}{\partial x^2} \tag{5.15}$$

If we take the fourier transform $\hat{u}(k,t) = \int_{-\infty}^{\infty} u(x,t)e^{-ikx}\mathrm{d}x$, we get:

$$\frac{\mathrm{d}\hat{u}}{\mathrm{d}t} = (-dk^2 - iak)\hat{u}(k,t) \tag{5.16}$$

This equation is a first order equation, which solution is:

$$\hat{u}(k,t) = e^{-dk^2t - iakt}\hat{u}_0 = \sigma(k)\hat{u}_0 \tag{5.17}$$

where $\sigma(k)$ is the amplification factor of the exact solution. Now, let us consider a unique mode as an initial condition $\hat{u}(k,0) = \sigma(k-\beta)$, where β is the wave number of the initial mode, which corresponds to $u(x,t) = \frac{1}{2\pi}e^{i\beta x}$. The solution then writes:

$$\hat{u}(k,t) = e^{-d\beta^2t - ia\beta t}\hat{u}_0 \tag{5.18}$$

Going back to the physical space, we can get the following possible behaviour:

- pure advection for $d = 0$: $u(x,t) = \frac{e^{i\beta(x-at)}}{2\pi}$, for which the initial mode travels at constant velocity with no gain or loss, see Fig. 5.3.
- pure diffusion for $a = 0$: $u(x,t) = \frac{e^{i\beta x}}{2\pi}e^{-d\beta^2t}$, for which the initial mode is stationary and decaying in time, see Fig. 5.4.

These two references behaviours are used to be compared with the analytical ones.

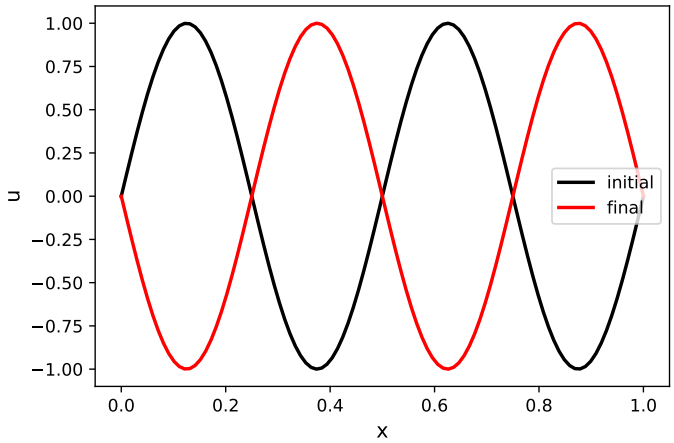


Figure 5.3 : *Exact solution of the 1d linear advection equation for an initial sinusoidal perturbation.*

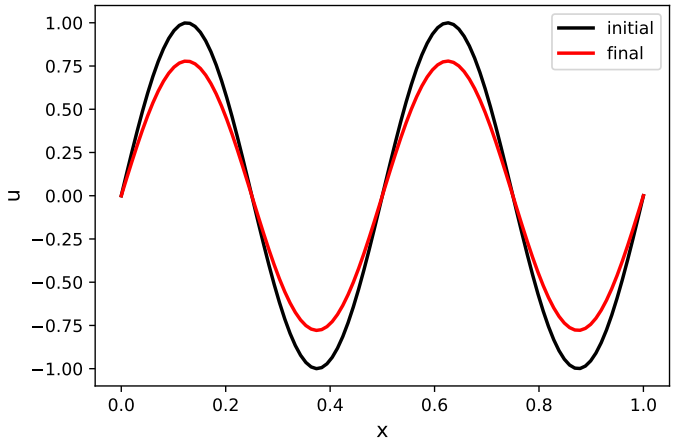


Figure 5.4 : *Exact solution of the 1d linear diffusion equation for an initial sinusoidal perturbation.*

5.3.1.2 Behavior of the numerical solution

In the same spirit as in the previous section, we will use Fourier transform to analyze the solution. This time, since numerical solution is in form of discrete data, the discrete Fourier transform will be used[1]:

$$\hat{u}^n(k) = \sum_{j=0}^{N_x-1} u_j^n e^{-ij\Delta x k} \Delta x \tag{5.19}$$

Let us consider that a discretisation in time and space of a linear equation leads to a linear system to be solved:

$$\sum_l a_l u_{j+l}^{n+1} = \sum_l b_l u_{j+l}^n \tag{5.20}$$

[1] Here i refers to the imaginary number.

where l belongs to the stencil of the scheme.
Applying the Discrete Fourier Transform leads to:

$$\left[\sum_l a_l e^{ikl\Delta x}\right] \hat{u}_j^{n+1} = \left[\sum_l b_l e^{ikl\Delta x}\right] \hat{u}_j^n(k) \tag{5.21}$$

$$\frac{\hat{u}^{n+1}}{\hat{u}^n} = \frac{\sum_l b_l e^{ihl\Delta x}}{\sum_l a_l e^{ihl\Delta x}} = \sigma^*(k) \tag{5.22}$$

$\sigma^\star(k)$ is the amplification factor of the numerical scheme. At this point, it is worth noticing that this amplification factor will be different from the exact solution, and comparing exact and numerical ones will give access to the numerical error made by the scheme. This parameter is also used to investigate the stability of the scheme.

Example : Discretizing the pure advection equation with forward Euler+upwind leads to the following scheme:

$$u_j^{n+1} = u_j^n - \mathcal{C}(u_j^n - u_{j-1}^n) \tag{5.23}$$

In this case, l is equal to 0 (local point j) or -1 (upwind point $j-1$). The coefficient are then $a_0 = 1$, $b_0 = 1 - \mathcal{C}$, and $b_{-1} = \mathcal{C}$.

5.3.1.3 Von Neumann Stability Analysis

A first way to evaluate the stability of a numerical method is to study its response to a unique harmonic perturbation, and to check whether it amplifies or damp the perturbation allows to discriminate between unstable and stable. This method is the *so-called* **Von Neumann stability analysis method**.

Von Neumann stability analysis

Consider a unique harmonic mode:

$$u_j^n = \exp\left(i\beta x_j\right) = \exp(i\beta j\Delta x), \tag{5.24}$$

Then, by computing the amplification factor σ as:

$$\sigma^* = \frac{u_j^{n+1}}{u_j^n}, \tag{5.25}$$

the stability of the numerical method is guaranteed as long as $|\sigma^*| \leq 1$, the method being unstable otherwise. Note that the existence of σ^* is a consequence of the linearity of the discretized PDE.

Example : Consider the Forward Euler - Centered Difference Scheme (5.6.1.1) for the heat equation (5.144) with

$$u_j^n = \exp(i\beta j\Delta x). \tag{5.26}$$

This yields

$$\phi_j^{n+1} = \phi_j^n + Fo\phi_j^n \left(\exp(i\beta\Delta x) - 2 + \exp(-i\beta\Delta x)\right) \phi_j^{n+1} = \phi_j^n \underbrace{\left(1 + 2\,Fo\left(\cos(\beta\Delta x) - 1\right)\right)}_{\sigma}. \tag{5.27}$$

Stability condition expresses here as

$$-1 \leq 1 + 2\,Fo\,(\cos(\beta\Delta x) - 1) \leq 1, \tag{5.28}$$

$$-2 \leq 2\,Fo\,\underbrace{(\cos(\beta\Delta x) - 1)}_{\leq 0} \leq 0, \tag{5.29}$$

$$0 \leq Fo \leq \frac{1}{1 - cos(\beta\Delta x)}. \tag{5.30}$$

The left-hand-side condition being always verified, the stability condition reads

$$Fo \leq \frac{1}{1 - cos(\beta\Delta x)}. \tag{5.31}$$

In the worst case, $\cos(\beta\Delta x) = -1$. Hence the **stability condition for Forward Euler – 2nd-order FD space with 1D Heat equation**:

$$\boxed{Fo \leq \frac{1}{2}} \tag{5.32}$$

In what follows, it will be clear that such a restriction on the time step $\left(\Delta t \leq \frac{\Delta x^2}{2a}\right)$, and similar ones when using explicit methods to solve *parabolic* PDEs, is penalizing and must be elevated.

Exercise 9

Characterize the stability of the discretization of the advection equation using Forward Euler and:

- 1st-order upwind,
- 1st-order downwind,
- 2nd-order centered.

Solution 9

1st-order upwind: Recalling equation (5.7) and assuming $c > 0$, we get:

$$u_j^{n+1} = u_j^n - \mathcal{C}\left(u_j^n - u_{j-1}^n\right), \tag{5.33}$$

with

$$u_j^n = e^{i\beta j\Delta x}, \tag{5.34}$$

one gets

$$u_j^{n+1} = u_j^n \underbrace{\left(1 - \mathcal{C} + \mathcal{C}e^{-i\beta\Delta x}\right)}_{\sigma} \tag{5.35}$$

$$\sigma = 1 - \mathcal{C}\left(1 - e^{-i\beta\Delta x}\right) = 1 - 2i\mathcal{C}\sin\left(\frac{\beta\Delta x}{2}\right)e^{-\frac{i\beta\Delta x}{2}} \tag{5.36}$$

$$\sigma = \left(1 - 2\mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right)\right) - i\mathcal{C}\sin(\beta\Delta x) \tag{5.37}$$

$$|\sigma|^2 = \left(1 - 2\mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right)\right)^2 + \mathcal{C}^2\sin^2(\beta\Delta x) \tag{5.38}$$

$$= 1 - 4\mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right) + 4\mathcal{C}^2\sin^4\left(\frac{\beta\Delta x}{2}\right) + \mathcal{C}^2\sin^2(\beta\Delta x) \tag{5.39}$$

$$= 1 - 4\mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right) + 4\mathcal{C}^2\sin^4\left(\frac{\beta\Delta x}{2}\right) + 4\mathcal{C}^2\sin^2\left(\frac{\beta\Delta x}{2}\right)\cos^2\left(\frac{\beta\Delta x}{2}\right) \tag{5.40}$$

$$= 1 - 4\mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right) + 4\mathcal{C}^2\sin^4\left(\frac{\beta\Delta x}{2}\right) + 4\mathcal{C}^2\sin^2\left(\frac{\beta\Delta x}{2}\right)\left(1 - \sin^2\left(\frac{\beta\Delta x}{2}\right)\right) \tag{5.41}$$

$$= 1 - 4\mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right) + 4\mathcal{C}^2\sin^2\left(\frac{\beta\Delta x}{2}\right) \tag{5.42}$$

Therefore, the stability condition $|\sigma|^2 \leq 1$ can be expressed as:

$$\mathcal{C}^2 - \mathcal{C} \leq 0, \tag{5.43}$$

$$\mathcal{C}(\mathcal{C} - 1) \leq 0. \tag{5.44}$$

Eventually, the stability condition for Forward Euler – 1$^{\text{st}}$-order Upwind scheme is:

$$\boxed{\mathcal{C} \leq 1} \tag{5.45}$$

This stability limit is known as the **Courant-Friedrich-Lewy**, or **CFL-condition**.

1$^{\text{st}}$-order downwind:

$$u_j^{n+1} = u_j^n - \mathcal{C}\left(u_{j+1}^n - u_j^n\right) \tag{5.46}$$

The Von-Neumann analysis reads

$$u_j^{n+1} = u_j^n\left(1 - \mathcal{C}e^{i\beta\Delta x} + \mathcal{C}\right), \tag{5.47}$$

$$\sigma = 1 - \mathcal{C}\left(e^{i\beta\Delta x} - 1\right) = 1 - 2i\mathcal{C}e^{\frac{i\beta\Delta x}{2}}\sin\left(\frac{\beta\Delta x}{2}\right) \tag{5.48}$$

$$\sigma = 1 + 2\mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right) - 2i\mathcal{C}\sin\left(\frac{\beta\Delta x}{2}\right)\cos\left(\frac{\beta\Delta x}{2}\right) \tag{5.49}$$

$$|\sigma|^2 = 1 + 4\mathcal{C}^2\sin^4\left(\frac{\beta\Delta x}{2}\right) + 4\mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right) + 4\mathcal{C}^2\sin^2\left(\frac{\beta\Delta x}{2}\right)\cos^2\left(\frac{\beta\Delta x}{2}\right) \tag{5.50}$$

$$|\sigma|^2 = 1 + 4\mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right) + 4\mathcal{C}^2\sin^2\left(\frac{\beta\Delta x}{2}\right) > 1 \tag{5.51}$$

Forward Euler – 1$^{\text{st}}$-order downwind is then an **unstable** scheme.

2$^{\text{nd}}$-order centered:

$$u_j^{n+1} = u_j^n - \frac{\mathcal{C}}{2}\left(u_{j+1}^n - u_{j-1}^n\right) \tag{5.52}$$

$$u_j^{n+1} = u_j^n \left(1 - \frac{\mathcal{C}}{2}\left(e^{i\beta\Delta x} - e^{-i\beta\Delta x}\right)\right) \tag{5.53}$$

$$\sigma = 1 - i\mathcal{C}\sin\left(\frac{\beta\Delta x}{2}\right) \tag{5.54}$$

$$\sigma^2 = 1 + \mathcal{C}\sin^2\left(\frac{\beta\Delta x}{2}\right) \geq 1 \tag{5.55}$$

Forward Euler – 2nd-order centered is then an **unstable** scheme.

5.3.1.4 Stability Analysis using the Modified Wavenumber

This method is similar to Von-Neumann. It also assumes periodic Boundary Conditions. Modified wavenumber analysis is also used for FD. The main difference with Von Neumann analysis is that it is based on the semi-discrete scheme, i.e. the time-like dimension is not discretized.

Modified wave number analysis

Considering the following semi-discretisation in space:

$$\frac{\mathrm{d}u_j(t)}{\mathrm{d}t} + \sum_l b_l u_{j+l}(t) = 0,$$

the spatially-discretized solution being sought as $u_j(t) = \psi(t)e^{i\beta j\Delta x}$, the spectral semi-discrete equation becomes a linear ODE:

$$\frac{\mathrm{d}\psi(t)}{\mathrm{d}t} = \left(\sum_l b_l e^{i\beta l\Delta x}\right)\psi(t) = \lambda\psi(t)$$

Then, the stability of the numerical scheme will first depend on the real part of λ: if $Re(\lambda) > 0$, the scheme is unstable. If $Re(\lambda) \leq 0$, the stability of the full scheme will depend on the choice of the ODE solver, as seen in Section 3.2.

Example: 1D Heat Equation : The following problem is considered here:

$$\frac{\partial\phi}{\partial t} = a\frac{\partial^2\phi}{\partial x^2}. \tag{5.56}$$

With a 2nd-order centered scheme, one gets:

$$\frac{d\phi_j}{dt} = \frac{a}{\Delta x^2}\left(\phi_{j+1} - 2\phi_j + \phi_{j-1}\right) \tag{5.57}$$

with

$$\phi_j = \psi(t)e^{i\beta j\Delta x} \tag{5.58}$$

Then,

$$\frac{d\phi_j}{dt} = \frac{a}{\Delta x^2}\left(e^{+i\beta\Delta x} - 2 + e^{-i\beta\Delta x}\right)\phi_j \tag{5.59}$$

$$\frac{d\psi}{dt} = -\frac{2a}{\Delta x^2}\left(1 - \cos\left(\beta\Delta x\right)\right)\psi(t) = -a\beta'^2\psi(t) \tag{5.60}$$

where

$$\beta'^2\Delta x^2 = 2\left(1 - cos\left(\beta\Delta x\right)\right) \tag{5.61}$$

is the modified wavenumber seen in FD lectures. The previously found eigenvalues $\lambda = \frac{2a}{\Delta x^2}\cos(\beta\Delta x) - 1$ are retrieved using β'. Each space-discretization scheme yields a different β', already known. The corresponding eigenvalues are $\boxed{\lambda = -a\beta'^2}$, they are then all real negative numbers for the 1D heat equation.

- Euler Stability with centered 2nd-order: In this case, eigenvalues are given by

$$\lambda = \frac{2a}{\Delta x^2}\left(\cos\left(k\Delta x\right) - 1\right). \tag{5.62}$$

Hence the following limitation:

$$\Delta t \leq \frac{2}{|\lambda|} \leq \frac{\Delta x^2}{2a} \tag{5.63}$$

and eventually, one retrieves

$$\boxed{Fo \leq \frac{1}{2}}. \tag{5.64}$$

- Euler Stability with centered 4th-order:

$$\phi_i^{n+1} = \phi_i^n + \frac{Fo}{12}\left(-\phi_{i-2}^n + 15\phi_{i-1}^n - 30\phi_i^n + 16\phi_{i+1}^n - \phi_{i+2}^n\right) \tag{5.65}$$

We have seen that the 4th-order for $\frac{\partial^2\phi}{\partial x^2}$ gives

$$k'^2\Delta x^2 = \frac{1}{6}\cos\left(2k\Delta x\right) - \frac{8}{3}\cos\left(k\Delta x\right) + \frac{5}{2} \tag{5.66}$$

then

$$\Delta t \leq \frac{2}{|\lambda|} = \frac{2}{\frac{a}{\Delta x^2}\left|\frac{1}{6}\cos(2k\Delta x - \frac{8}{3}\cos\left(k\Delta x\right) + \frac{5}{2})\right|} \tag{5.67}$$

The worst case corresponds to $k\Delta x = \pi$, *i.e.* $\frac{a\Delta t}{\Delta x^2} \leq \frac{2}{\frac{1}{6}+\frac{8}{3}+\frac{5}{2}} = \frac{3}{8}$ Eventually, the Fourier condition reads:

$$\boxed{Fo \leq \frac{3}{8}} \tag{5.68}$$

Example: 1D Advection Equation : Consider the following problem:

$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x}, \tag{5.69}$$

with

$$u_j = \psi(t)e^{ikx_j}. \tag{5.70}$$

This gives

$$\frac{d\psi}{dt} = -ick\psi(t) \tag{5.71}$$

The eigenvalues are then $\boxed{\lambda = ick'}$ for 1D advection, with k' the modified wavenumber of $\frac{\partial u}{\partial x}$

- Centered 2nd-order scheme: For this scheme,

$$k'\Delta x = \sin(k\Delta x) \tag{5.72}$$

$$\lambda = -\frac{ic}{\Delta x}\sin(k\Delta x) \tag{5.73}$$

For a Centered 2nd-order scheme space-discretization, one retrieves the **unstability** of Euler, RK2 and AB2, the **stability** of LeapFrog for $\mathcal{C} \leq 1$ and of RK4 for $\mathcal{C} \leq 2.83$.

- Centered 4th-order scheme: In this case, the space discretization reads:

$$\left.\frac{\partial u}{\partial x}\right|_i = \frac{u_{i-2} - 8u_{i-1} + 8u_{i+1} - u_{i+2}}{12h}. \tag{5.74}$$

This yields

$$k'\Delta x = \frac{4}{3}\sin(k\Delta x) - \frac{1}{6}\sin(2k\Delta x) \tag{5.75}$$

and

$$\lambda = -ick' \tag{5.76}$$

Let us find $k'_{\max}$ to characterize the stability, using $X = k\Delta x$:

$$\frac{dk'}{dX} = \frac{1}{\Delta x}\left(\frac{4}{3}\cos(X) - \frac{1}{3}\cos(2X)\right) = 0 \tag{5.77}$$

$$4\cos(X) = \cos(2X) = 2\cos^2(X) - 1 \tag{5.78}$$

setting $Y = \cos(X)$ gives

$$2Y^2 - 4Y - 1 = 0, \tag{5.79}$$

which, since $|Y| \leq 1$, reduces to one solution:

$$Y = 1 - \frac{1}{2}\sqrt{6}. \tag{5.80}$$

Eventually,

$$X = \arccos(Y) \approx 1.8 \tag{5.81}$$

and

$$k'_{\max} = k'(X) = \frac{1.372}{h} \tag{5.82}$$

This yields

$$|\lambda|_{\max} = \frac{1.372}{\Delta x} \tag{5.83}$$

$$\boxed{\begin{array}{ll}\text{RK3:} & \mathcal{C} \leq 1.26 \\ \text{RK4:} & \mathcal{C} \leq 2.06\end{array}} \tag{5.84}$$

- Padé scheme (4^{th}-order): Padé scheme consists in computing the approximate derivatives $\frac{\partial u}{\partial x}\Big|_j$ by solving equation:

$$\left.\frac{\partial u}{\partial x}\right|_{j+1} + 4\left.\frac{\partial u}{\partial x}\right|_{j} + \left.\frac{\partial u}{\partial x}\right|_{j-1} = \frac{3}{\Delta x}\left(u_{j+1} - u_{j-1}\right). \tag{5.85}$$

This yields

$$k'\Delta x = \frac{3\sin(k\Delta x)}{2 + \cos(k\Delta x)}, \tag{5.86}$$

and

$$\lambda = -ick'. \tag{5.87}$$

λ is then pure imaginary. One can now evaluate $k_{\max}$ using

$$\frac{dk'}{dx} = 0 = \frac{3}{\Delta x}\frac{\left[\cos(x)\left(2 + \cos(x)\right) + \sin^2(x)\right]}{\left(2 + \cos(x)\right)^2}, \tag{5.88}$$

with $X = k\Delta x$. The maximum is then obtained for:

$$\cos(X) = -\frac{1}{2}, \tag{5.89}$$

which corresponds to

$$X = \frac{2\pi}{3}, \tag{5.90}$$

and

$$\boxed{k'_{\max} = \frac{\sqrt{3}}{h}} \tag{5.91}$$

Eventually, one finds for RK3 and RK4 time integration:

$$\boxed{\begin{array}{lll}\text{Padé - RK3:} & \Delta t \leq \dfrac{\sqrt{3}}{|\lambda|_{\max}} & \mathcal{C} \leq 1.0 \\ \text{Padé - RK4:} & \Delta t \leq \dfrac{2.83}{|\lambda|_{\max}} & \mathcal{C} \leq 1.66\end{array}} \tag{5.92}$$

Example: 1D Advection-Diffusion Equation : Recall the 1D formulation of an Advection-Diffusion Problem:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}. \tag{5.93}$$

The corresponding modified wavenumber equation reads

$$\frac{du_j}{dt} = -ick'_1 u_j - \nu k'^2_2 u_j. \tag{5.94}$$

Using a cenetered 2^{nd}-order scheme for both terms:

$$\lambda = -\frac{2\nu}{\Delta x^2}\left(1 - \cos(k\Delta x)\right) - ic\frac{\sin(k\Delta x)}{\Delta x}. \tag{5.95}$$

Considering Forward-Euler stability condition, one finds:

$$\mathcal{C}^2 \leq 2Fo \leq 1 \tag{5.96}$$

Thus, adding diffusion makes Forward-Euler **stable**. This is the principle of *artificial diffusion*, which will be discussed later on.

5.3.1.5 Limitations to Fourier-space Analysis

The Fourier space analysis suffers the following limitations:

- it is restricted to linear PDE with fixed coeffs,
- it can only be performed on uniform meshes,
- the use of Fourier modes assumes the periodic *Boundary Conditions*. Some counter-examples exist that show the impact of *Boundary Conditions* treatment on stability.

5.3.2 Stability Analysis of the semi-discrete PDE

It is also possible to investigate the stability of a numerical scheme without the use of a Fourier space analysis, but by directly looking the linear system generated by the scheme. Before actually tackling the stability analysis, a quick mathematical preamble is necessary.

Eigenvalues of a tridiagonal matrix

For a $(N-1)\times(N-1)$ tridiagonal matrix

$$M = \begin{pmatrix} b & c & & 0 \\ a & \ddots & \ddots & \\ & \ddots & \ddots & c \\ 0 & & a & b \end{pmatrix}, \tag{5.97}$$

denoted by $B[a,b,c]$, the eigenvalues are

$$\lambda_j = b + 2\sqrt{ac}\,\cos\left(\frac{j\pi}{N}\right) \quad \text{for} \quad 1 \leq j \leq N-1 \tag{5.98}$$

For the demonstration, see Appendix A.1.

5.3.2.1 Method of Lines (MOL)

Method of Lines (MOL)

The semi-discrete PDE, as known as *MOL*, is obtained by discretizing along all directions except for the time-like variable:

$$Pu = 0 \Rightarrow \frac{\mathrm{d}}{\mathrm{d}t} u_j(t) + P_{\Delta x} u_j(t) = 0 \tag{5.99}$$

The PDE is then turned into a set of ODEs. Any ODE method can then be applied. The MOL enables us to use what we have learnt on ODE numerical methods. This gives a good insight on the most appropriate methods to chose to resolve the PDE given a spatial discretization. However, not all discretized PDE can be interpreted using MOL (*e.g.* Lax Wendroff, to be described later).

Example: 1D Heat Equation : Consider the following Cauchy Problem for a physical domain $\Omega = [0, L]$:

$$\begin{cases} \dfrac{\partial \phi}{\partial t} = a \dfrac{\partial^2 \phi}{\partial x^2} & \text{for } (x,t) \in \Omega \times \mathbb{R}^+ \\ \phi(0,t) = \phi(L,t) = 0 & \text{for } t \in \mathbb{R}^+ \\ \phi(x,0) = g(x) & \text{for } x \in \Omega \end{cases} \tag{5.100}$$

Let $(x_j)_{j=0..N}$ be a uniform mesh of $N+1$ points, with $x_0 = 0$ and $x_N = L$. Using a 2^{nd}-order centered formula for space discretization, one gets:

$$\frac{d\phi_j}{dt} = a \left(\frac{\phi_{j+1} - 2\phi_i + \phi_{j-1}}{\Delta x^2} \right) \quad \text{for } j = 1, ..., N-1. \tag{5.101}$$

This yields a system of $N-1$ ODEs:

$$\frac{d}{dt} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_{N-1} \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_{N-1} \end{bmatrix}, \tag{5.102}$$

where

$$\boldsymbol{A} = \frac{a}{\Delta x^2} \begin{pmatrix} -2 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -2 \end{pmatrix} = \frac{a}{\Delta x^2} B[1, -2, 1]. \tag{5.103}$$

The system can then be solved using RK, multi-step or other ODE methods. The stability of the ODE method is determined by the eigenvalues of $\boldsymbol{A}$:

$$\lambda_j = \frac{a}{\Delta x^2} \left(-2 + 2\cos\left(\frac{j\pi}{N} \right) \right) \quad \text{with } 1 \leq j \leq N-1 \tag{5.104}$$

Thus, all λ_j are real negative numbers. Let us study the stability for different choices of ODE numerical methods.

1. Forward Euler: the method reads

$$\phi_j^{n+1} = \phi_j^n + Fo \left(\phi_{j+1}^n - 2\phi_j^n + \phi_{j-1}^n \right), \tag{5.105}$$

which is stable if $\Delta t \leq \frac{2}{|\lambda_j|}$ for all $j \in \{1..N-1\}$. Hence

$$\Delta t \leq \frac{2}{|\lambda|_{\max}}, \quad \text{and} \tag{5.106}$$

$$|\lambda|_{\max} \approx \frac{4a}{\Delta x^2}, \quad \textit{i.e.} \tag{5.107}$$

$$\Delta t \leq \frac{\Delta x^2}{2a}, \quad \text{and eventually} \tag{5.108}$$

$$\boxed{Fo \leq \frac{1}{2}}, \tag{5.109}$$

which is in agreement with the previously found stability limit.

2. RK2: this two-stage method reads:

$$\begin{cases} \phi_j^{n+1/2} = \phi_j^n + \dfrac{Fo}{2}\left(\phi_{j+1}^n - 2\phi_j^n + \phi_{j-1}^n\right) \\ \phi_j^{n+1} = \phi_j^n + Fo\left(\phi_{j+1}^{n+1/2} - 2\phi_j^{n+1/2} + \phi_{j-1}^{n+1/2}\right) \end{cases} \tag{5.110}$$

The stability condition then writes:

$$\Delta t \leq \frac{2}{|\lambda|_{\max}}, \tag{5.111}$$

hence again

$$\boxed{Fo \leq \frac{1}{2}}. \tag{5.112}$$

3. Backward Euler: as an implicit method, it is unconditionnaly stable
4. Cranck-Nicolson: also known as trapezoidal – unconditionnaly stable
5. RK4: For this method, one finds:

$$\Delta t \leq \frac{2.78}{|\lambda|_{\max}}, \tag{5.113}$$

which gives

$$\boxed{Fo \leq 0.70}. \tag{5.114}$$

Example: 1D Advection Equation : Consider the following Cauchy Problem for a physical domain $\Omega = [0, L]$:

$$\begin{cases} \dfrac{\partial u}{\partial t} + c\dfrac{\partial u}{\partial x} = 0 & \text{for } (x,t) \in \Omega \times \mathbb{R}^+ \\ u(0,t) = 0 & \text{for } t \in \mathbb{R}^+ \text{ (set on left side only)} \\ u(x,0) = g(x) & \text{for } x \in \Omega \end{cases} \tag{5.115}$$

Let $(x_i)_{i=0..N}$ be a uniform mesh of $N+1$ points, with $x_0 = 0$ and $x_N = L$. Using a 2nd-order centered formula for space discretization, one gets:

$$\frac{du_i}{dt} = -c\left(\frac{u_{i+1} - u_{i-1}}{2\Delta x}\right) \quad \text{for } i = 1, ..., N-1. \tag{5.116}$$

This yields a system of $N-1$ ODEs[2]:

$$\frac{d}{dt}\begin{bmatrix} u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} = -\frac{c}{2\Delta x} B[-1,0,1] \begin{bmatrix} u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}. \tag{5.117}$$

Again, the system can then be solved using RK, multi-step or other ODE methods. The stability of the ODE method is determined by the eigenvalues of $\left(-\frac{c}{2\Delta x} B[-1,0,1]\right)$:

$$\lambda_j = -\frac{ic}{\Delta x}\cos\left(\frac{\pi j}{N}\right) \quad \text{for } 1 \leq j \leq N \tag{5.118}$$

Here, all λ_j are **pure imaginary**. For this reason, every ODE method that does not include a part of imaginary axes in its stability domain is doomed to fail.

[2] In reality, a 1st-order upwind discretization has to be used at the right-side boundary.

Example : Forward Euler, RK2, AB2 will fail.

1. Leap Frog: The update is computed as:

$$u_i^{n+1} = u_i^{n-1} - \frac{\mathcal{C}}{2}\left(u_{i+1}^n - u_{i-1}^n\right) \tag{5.119}$$

The stability condition writes:

$$\Delta t \leq \frac{1}{|\lambda|_{\max}} \quad \text{with } |\lambda|_{\max} = \frac{c}{\Delta x} \tag{5.120}$$

$$\boxed{\mathcal{C} \leq 1} \tag{5.121}$$

2. RK3: the stability condition can be proved to be:

$$\Delta t \leq \frac{\sqrt{3}}{|\lambda|_{\max}} \tag{5.122}$$

The CFL condition is thus:

$$\boxed{\mathcal{C} \leq \sqrt{3}}. \tag{5.123}$$

3. RK4: the stability condition reads:

$$\Delta t \leq \frac{2.83}{|\lambda|_{\max}}, \tag{5.124}$$

which yields the following CFL condition:

$$\boxed{\mathcal{C} \leq 2.83}. \tag{5.125}$$

5.3.3 Stability of Multidimensionnal Problems

Multidimensionnal problems can be tackled using generalizations of the 1D methods studied previously, using multidimensionnal Fourier modes: $e^{ik_x x_j}, e^{ik_y y_\ell}, e^{ik_z z_p}$
In the following, results for different equations and discretizations are condensed.

Heat Equation: The 2D version of this PDE reads:

$$\frac{\partial \phi}{\partial t} = a\left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}\right) \tag{5.126}$$

Using a Forward-Euler + centered 2$^{\text{nd}}$-order space discretization, the stability condition on the Fourier number is:

$$\boxed{Fo_{\text{2D}} \leq \frac{1}{4}} \tag{5.127}$$

The 3D heat equation is given by:

$$\frac{\partial \phi}{\partial t} = a\left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2}\right) \tag{5.128}$$

The same discretization (Forward-Euler + centered 2$^{\text{nd}}$-order space) would yield:

$$\boxed{Fo_{\text{3D}} \leq \frac{1}{6}} \tag{5.129}$$

Advection Equation: The 2D version of this PDE reads:

$$\frac{\partial \phi}{\partial t} + c_1 \frac{\partial \phi}{\partial x} + c_2 \frac{\partial \phi}{\partial y} = 0 \tag{5.130}$$

The stability condition corresponding to a Forward-Euler + 1^{st}-order upwind discretization in each direction gives the following CFL condition:

$$\mathcal{C}_1 + \mathcal{C}_2 \leq 1, \tag{5.131}$$

where

$$\mathcal{C}_d = \frac{|c_d|\Delta t}{\Delta x}. \tag{5.132}$$

For a 3D formulation, which reads:

$$\frac{\partial \phi}{\partial t} + c_1 \frac{\partial \phi}{\partial x} + c_2 \frac{\partial \phi}{\partial y} + c_3 \frac{\partial \phi}{\partial z} = 0, \tag{5.133}$$

one obtains the 3D CFL criterion:

$$\mathcal{C}_1 + \mathcal{C}_2 + \mathcal{C}_3 \leq 1. \tag{5.134}$$

5.4 Characterization of numerical errors

5.4.1 Numerical diffusion and dispersion

Using a convergent scheme, the exact solution is retrieved when time and space step tend to zero under the appropriate stability condition. If convergence is not fully achieved, a numerical error is made, and it is important to characterize which type of influence this error has on the numerical solution.
To analyse this error, the strategy consists in comparing exact and numerical solutions, by looking the amplification factor σ.

Numerical diffusion and dispersion

Numerical diffusion means that the loss of amplitude compared to the exact solution, meaning that the rate of decay is not matched:

$$\frac{|\sigma^*(k)|}{|\sigma(k)|} \neq 1$$

Numerical dispersion means that the propagation speed is not matched, i.e, the solution is travelling slower or faster:

$$\frac{arg(\sigma^*(k))}{arg(\sigma(k))} \neq 1$$

In general, diffusion and dispersion depend on wave number k.

5.4.1.1 Application to the 1D linear advection equation

In the case of the advection equation, the solution is a pure propagation at a fixed velocity without diffusion of the solution

$$\hat{u}(k,t) = e^{-iakt}\hat{u}_0 \tag{5.135}$$

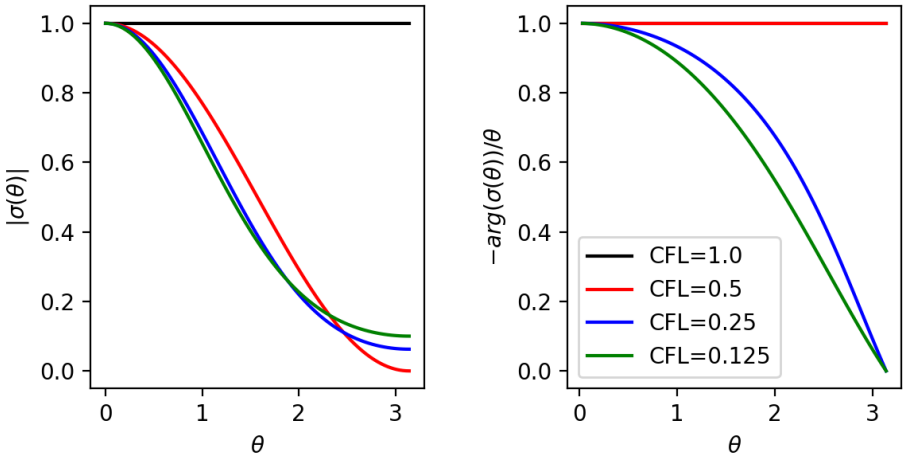


Figure 5.5 : *Diffusion and Dispersion errors of the Euler+Upwind scheme applied to the advection equation at different CFL numbers for a final time* $t_f = \dfrac{\Delta x}{a}$.

In order to compare the results of different schemes at different CFL numbers, and thus different time steps, it is required to fix the final time at which the comparison is performed. Here, we arbitrarily choose $t_f = \dfrac{\Delta x}{a}$, which is the time required to cross a cell at $CFL = 1$. At this final time, we get:

$$\hat{u}(k, t_f) = e^{-iak\frac{\Delta x}{a}}\hat{u}(k, 0) \quad \Rightarrow \quad \hat{u}(k, t_f) = e^{-i\theta}\hat{u}(k, 0) \tag{5.136}$$

where $\theta = k\Delta x$ is the node wave number. Here we get the exact amplication factor, which presents no diffusion ($|e^{-i\theta}| = 1$)and no dispersion ($arg(e^{-i\theta} = -\theta$).
Now, let us consider the discretised solution of the advection equation using the Euler+Upwind scheme. Using the Discrete Fourier Transform, we get for one iteration, i.e. one time step:

$$\hat{u}^{n+1} = (1 - \mathcal{C})\hat{u}^n + \mathcal{C}e^{-i\theta}\hat{u}^n$$

and then, for multiple time steps:

$$\hat{u}^{n+l} = (1 - \mathcal{C} + \mathcal{C}e^{-i\theta})^l\hat{u}^n$$

We thus get for the amplification factor after l time steps:

$$\Rightarrow \sigma(\theta) = (1 - \mathcal{C} + \mathcal{C}e^{-i\theta})^l = (1 - \mathcal{C}(1 - cos\theta) - i\mathcal{C}sin\theta)^l$$

Taking the norm and the argument of the amplification factor, we get the diffusion and dispersion properties of this scheme, plotted in Fig. 5.5. We first see that the diffusion and dispersion errors clearly depend on the frequency of interest. Here the error of Euler+Upwind scheme is increasing together with the frequency of interest. We can clearly see that the CFL number has a great impact on the diffusion and dispersion errors.
The influence of theses numerical errors on the solution is shown in Fig. 5.6, where the initial solution consists in a pure sinusoidal mode $u(t = 0, x) = sin(\omega x)$, where $\omega = 4\pi$. The solution is plotted at the same final time $t_f = 0.5$ at which initial and final solution must overlap. The numerical diffusion is clearly highlighter by the loss of amplitude of the sinusoidal wave. The dispersion is also highlighted by the space shift of the numerical solution, showing here a lower propagation speed than the expected one. This is coherent with Fig. 5.5, which shows a dispersion error below 1 for $\mathcal{C} = 0.25$, i.e. a lower propagation speed.

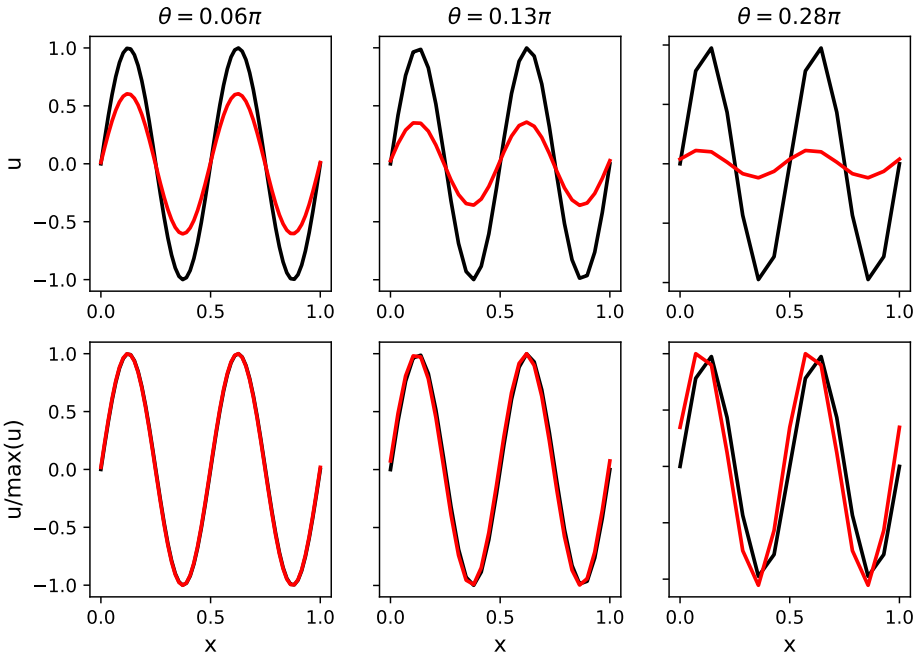


Figure 5.6 : *Numerical solution of the Euler+Upwind scheme applied to the advection equation for an initial sinusoidal wave at* $\mathcal{C} = 0.25$*. Final solution* u *(upper part) and final solution normalized by its maximum value (lower part), with different normalized wave numbers* $\theta = 0.01\pi$ *(left),* 0.13π *(center) and* 0.28π *(right).*

5.4.2 Modified equation

Another way to analyse the numerical error of a numerical scheme is to look at the modified equation of the numerical scheme. The principle of the modified equation is to write a continuous equation that represents the evolution of the numerical solution. To do so, we need to keep higher order terms in the Taylor expansions that have been neglected. For example, to approximate a first order derivative in space, we discard higher order terms:

$$u(t,x+\Delta x) = u(t,x) + \left.\frac{\partial u}{\partial x}\right|_x^t \Delta x + \underbrace{\sum_{k=2}^{\infty} \left.\frac{\partial^k u}{\partial x^k}\right|_x^t \frac{\Delta x}{k!}}_{\text{Numerical error}}$$

In the discarded part, the leading term is the first element of the sum:

$$\sum_{k=2}^{\infty} \left.\frac{\partial^k u}{\partial x^k}\right|_x^t \frac{\Delta x}{k!} = \underbrace{\left.\frac{\partial^2 u}{\partial x^2}\right|_x^t \frac{\Delta x}{2}}_{\text{Leading term of the numerical error}} + \sum_{k=3}^{\infty} \left.\frac{\partial^k u}{\partial x^k}\right|_x^t \frac{\Delta x}{k!}$$

The numerical error will then be mainly caused by this first term. Let us get back to the downwind discretisation of the first order derivative:

$$\left.\frac{\partial u}{\partial x}\right|_j^n = \frac{u_{j+1}^n - u_j^n}{\Delta x}$$

If we look at the discarded error terms and we reintroduce the leading one, we get:

$$\frac{u_\sigma^{n+1} - u_j^n}{\Delta x} = \left.\frac{\partial u}{\partial x}\right|_j^n + \left.\frac{\partial^2 u}{\partial x^2}\right|_\sigma^n \frac{\Delta x}{2}$$

This give a continuous representation of the numerical scheme, with an additional term representing the numerical error.

5.4.2.1 Application to forward Euler + Upwind

Let us consider the 1D linear advection solved using Forward Euler + Upwind scheme:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a\frac{u_j^n - u_{j-1}^n}{\Delta x} = 0$$

For the time derivative, we get:

$$\underbrace{\frac{u_j^{n+1} - u_j^n}{\Delta t}}_{\text{numerical approximation}} = \underbrace{\left.\frac{\partial u}{\partial t}\right|_j^n}_{\text{real solution}} + \underbrace{\left.\frac{\partial^2 u}{\partial t^2}\right|_j^n \frac{\Delta t}{2} + O(\Delta t^2)}_{\text{numerical error}}$$

For the space derivative, we get:

$$a\frac{(u_j^n - u_{j-1}^n)}{\Delta x} = a\left.\frac{\partial u}{\partial x}\right|_j^n - \frac{\partial^2 u}{\partial x^2}\left(\frac{a\Delta x}{2}\right) + O(\Delta x^2)$$

Combining both equations:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = -\frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} + a\frac{\Delta x}{2}\frac{\partial^2 u}{\partial x^2}$$

The right hand side is the numerical error $e = -\frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} + a\frac{\Delta x}{2}\frac{\partial^2 u}{\partial x^2}$. We still have to reformulate the second order time derivative to give a physical meaning to this error. To do so we take the time derivative of this equation:

$$\frac{\partial^2 u}{\partial t^2} = -a\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial t}\right) + \frac{\partial e}{\partial t} = a^2\frac{\partial^2 u}{\partial x^2} + \frac{\partial e}{\partial t} - a\frac{\partial e}{\partial x}$$

We then get an implicit equation of the numerical error:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = e = \frac{-\Delta t}{2}\frac{a^2\partial^2 u}{\partial x^2} + \frac{\Delta x}{2}\frac{\partial^2 u}{\partial x^2} - \frac{\Delta t}{2}\left(\frac{\partial e}{\partial t} - a\frac{\partial e}{\partial x}\right)$$

The last term of this equation is not explicitely determined. However, we know that the numerical error is of order one in time and space. As this last term is multiplied by Δt, this term ends to be of second order, and then to be negligible compared to the order of the numerical error. We finally get the modified equation of the upwind scheme

Modified equation of the Forward Euler+Upwind scheme

$$\frac{\partial u}{\partial t} + \frac{a\partial u}{\partial x} = \frac{1}{2}a\Delta x(1 - a\frac{\Delta t}{\Delta x})\frac{\partial^2 u}{\partial x^2}$$

The RHS acts as a numerical diffusion with diffusion coefficient:

$$D = \frac{1}{2}a\Delta x\left(1 - a\frac{\Delta t}{\Delta x}\right)$$

If this coefficient is positive, the scheme is stable, and if this coefficient is negative the scheme is unstable. We then obtain the stability limit of the scheme:

$$D > 0(\text{stable}) \text{ if } a > 0 \text{ and } a\frac{\Delta t}{\Delta x} = \mathcal{C} \leq 1$$
$$D < 0(\text{unstable}) \text{ if } a < 0 \text{ or } \mathcal{C} > 1$$

5.4.2.2 Forward Euler + Centered scheme

Applying the same methodology we get for the Forward Euler+Centered scheme:

Modified equation of the Forward Euler+Centered scheme

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = \underbrace{-a^2\frac{\Delta t}{2}\frac{\partial^2 u}{\partial x^2}}_{\text{anti-diffusive term} \Rightarrow \text{unstable}} \quad \underbrace{-a\frac{\Delta x^2}{6}\frac{\partial^3 u}{\partial x^3}}_{\text{dispersive term}}$$

As the diffusion coefficient is here always negative, the scheme is unconditionally unstable. It also exhibit a third order derivative term, which corresponds to dispersion error.

5.5 Lax Wendroff and Lax-Friedrichs schemes

5.5.1 Artificial diffusion

To get a scheme at second order in space we require a centered scheme for the space derivative. Forward Euler leads to an unstable scheme because of negative diffusion. A possible solution to get a second order accuracy in space is to add a positive diffusion to counterbalance it:

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+a\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}=D\frac{u_{j+1}^n-2u_j^n+u_{j-1}^n}{\Delta x^2} \tag{5.137}$$

The question is then how to taylor the numerical diffusion in order to get stability while preserving consistency, in order to achieve convergence. The first solution is to add a constant diffusion D_{artif}, but this is however not consistent with the advection equation, but with the advection diffusion equation:

$$\frac{\partial u}{\partial t}+a\frac{\partial u}{\partial x}=D_{artif}\frac{\partial^2 u}{\partial x^2} \tag{5.138}$$

5.5.2 Lax Wendroff scheme

Let us tailor D such that it exactly matches the negative diffusion, i.e. $D=a^2\frac{\Delta t}{2}$. We then get the Lax-Wendroff scheme:

Lax Wendroff scheme

$$u_i^{n+1}=u_i^n-a\frac{\Delta t}{2\Delta x}\left[u_{i+1}^n-u_{i-1}^n\right]+\frac{a^2\Delta t^2}{2\Delta x^2}\left[u_{i+1}^n-2u_i^n+u_{i-1}^n\right] \tag{5.139}$$

c
The modified equation of the Lax-Wendroff scheme is the following:

$$\frac{\partial u}{\partial t}+a\frac{\partial u}{\partial x}=\frac{1}{8}a\Delta x^2(\mathcal{C}^2-1)\frac{\partial^3 u}{\partial x^3} \tag{5.140}$$

The term on the right hand side is a third order derivative, it is thus a dispersive term. This scheme is second order in time and space. The amplification factor of the Lax-Wendroff scheme is:

$$\sigma(\theta)=\frac{1}{2}\mathcal{C}(1+i)e^{-i\theta}+1-\mathcal{C}^2-\frac{1}{2}\mathcal{C}(1-i)e^{-i\theta} \tag{5.141}$$

5.5.3 Lax-Friedrichs scheme

The principle of the Lax-Friedrichs scheme is to replace u_i^n by $\frac{u_{j+1}^{n+1}+u_{j-1}^n}{2}$. So we get:

$$\frac{u_i^{n+1}-\frac{u_{i+1}^n+u_{i-1}^n}{2}}{\Delta t}+a\frac{u_{i+1}^n-u_{i-1}^n}{2\Delta x}=0 \tag{5.142}$$

Lax Friedrichs scheme

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}+a\frac{u_{i+1}^n-u_{i-1}^n}{2\Delta x}=\frac{1}{2\Delta t}\left[u_{i+1}^n-2u_i^n+u_{i-1}^n\right] \tag{5.143}$$

The modified equation is :

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = \frac{\Delta x^2}{2\Delta t}(1-\mathcal{C}^2)\frac{\partial^2 u}{\partial x^2}$$

meaning that the numerical error made by the scheme is mainly numerical diffusion.

5.6 Consistent and stable discretisations

5.6.1 Diffusion equations

5.6.1.1 1D Diffusion Equation

Recall the (parabolic) 1D diffusion equation:

$$\frac{\partial \phi}{\partial t} = a\frac{\partial^2 \phi}{\partial x^2} \tag{5.144}$$

It can be discretized as (ϕ_i^n), where i denotes the space index and n the time index.
Using a Forward Euler time discretization and a 2nd-order centered space discretization, the numerical method writes:

1D diffusion: Forward Euler+ Centered

$$\phi_i^{n+1} = \phi_i^n + Fo\left(\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n\right).$$

Replacing the Forward Euler time discretization by Backward Euler yields the following implicit method:

1D diffusion: Backward Euler+ Centered

$$\phi_i^{n+1} = \phi_i^n + Fo\ \left(\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}\right)$$

Using a trapezoidal method instead yields the so-called *Crank-Nicolson* (implicit) method:

1D diffusion: Cranck-Nicolson + Centered

$$\phi_i^{n+1} = \phi_i^n + \frac{1}{2}\left[Fo\ \left(\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n\right) + Fo\ \left(\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}\right)\right]$$

A last example of discretization for the 1D diffusion equation can be a Forward Euler time discretization and a 4th-order centered finite difference discretization in space. The resulting scheme reads:

1D diffusion: Forward Euler+ 4th order-Centered

$$\phi_i^{n+1} = \phi_i^n + \frac{Fo}{12}\left(-\phi_{i-2}^n + 16\phi_{i-1}^n - 30\phi_i^n + 16\phi_{i+1}^n - \phi_{i+2}^n\right) \tag{5.145}$$

5.6.1.2 2D diffusion equations

Recall the 2D diffusion equation:

$$\frac{\partial \phi}{\partial t} = a\left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}\right). \tag{5.146}$$

This equation can also be discretized using Forward Euler in time and a centered 2nd-order scheme in space:

2D Diffusion: Forward Euler+Centered

$$\phi_{i,j}^{n+1} = \phi_{i,j}^{n} + Fo_x\left(\phi_{i+1,j}^{n} - 2\phi_{i,j}^{n} + \phi_{i-1,j}^{n}\right) + Fo_y\left(\phi_{i,j+1}^{n} - 2\phi_{i,j}^{n} + \phi_{i,j-1}^{n}\right) \tag{5.147}$$

Here, one can chose two different mesh sizes along x and y directions. Hence the two different Fourier Numbers defined as:

$$Fo_x = \frac{a\Delta t}{\Delta x^2}, \tag{5.148}$$

$$Fo_y = \frac{a\Delta t}{\Delta y^2}. \tag{5.149}$$

Similarly to previous equation, one can blend different space and time methods to build other numerical schemes.

5.6.1.3 Non-linear diffusion equations

TO BE DONE

5.6.2 Advection equations

5.6.2.1 1D Advection Equation

Recall the (hyperbolic) 1D linear advection equation:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0. \tag{5.150}$$

This equation describes the advection of field u at speed c. We assume $c > 0$, corresponding to an advection from left to right.
Using a backward Euler time discretization and a 2nd-order centered finite differences space discretization, one gets:

1D advection: Backward Euler+Centered

$$u_i^{n+1} = u_i^n - \frac{\mathcal{C}}{2}\left(u_{i+1}^{n+1} - u_{i-1}^{n+1}\right).$$

Another possibility consists in applying a Forward Euler time discretization combined with a 1st-order upwind spatial discretization:

1D advection: Forward Euler+Upwind

$$\begin{aligned} u_i^{n+1} &= u_i^n - \mathcal{C}\left(u_i^n - u_{i-1}^n\right) && \text{if } c > 0, \\ u_i^{n+1} &= u_i^n - \mathcal{C}\left(u_{i+1}^n - u_i^n\right) && \text{if } c < 0. \end{aligned}$$

5.6.2.2 2D advection equation

The 2D advection equation reads:

$$\frac{\partial u}{\partial t} = c_x \frac{\partial u}{\partial x} + c_y \frac{\partial u}{\partial y}. \tag{5.151}$$

This equation can be discretized using Forward Euler in time and a 1st-order upwind scheme in space. This can be written, for $c_x > 0$ and $c_y > 0$, as

2D advection: First order Euler+Upwind

$$u_{i,j}^{n+1} = u_{i,j}^n - \mathcal{C}_x\left(u_{i,j}^n - u_{i-1,j}^n\right) - \mathcal{C}_y\left(u_{i,j}^n - u_{i,j-1}^n\right). \tag{5.152}$$

Again, two different *Courant numbers* appear:

$$\mathcal{C}_x = \frac{c_x \Delta t}{\Delta x} \tag{5.153}$$

$$\mathcal{C}_y = \frac{c_y \Delta t}{\Delta y} \tag{5.154}$$

5.6.3 Advection-Diffusion equation

The following equation is met when studying the *thin layer approximation* for Navier-Stokes equations:

$$u \underbrace{\frac{\partial \phi}{\partial x}}_{①} + v \underbrace{\frac{\partial \phi}{\partial y}}_{②} = a \underbrace{\frac{\partial^2 \phi}{\partial y^2}}_{③}. \tag{5.155}$$

The different terms of this equation can be discretized as follows:

① the preferred direction of this parabolic equation appears here. One can discretize this term using Forward Euler,

② this term is the advection term of the advection-diffusion equation. A 1st-order upwind scheme can be used,

③ the diffusion term can be again discretized following a 2nd-order centered scheme.

Eventually, this yields the following update:

2D advection-diffusion: Forward Euler+Upwind +Centered

$$\phi_{i+1,j} = \phi_{i,j} - \frac{v}{u}\frac{\Delta x}{\Delta y}\left(\phi_{i,j} - \phi_{i,j-1}\right) + \frac{a}{u}\frac{\Delta x}{\Delta y^2}\left(\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}\right) \tag{5.156}$$

5.6.4 Non-linear transport equations

TO BE DONE

Chapter 6

Implicit methods for multidimensional parabolic equations

Contents

6.1 Introduction

Implicit methods are sometimes preferred to explicit methods when the desired physical fidelity Δt_{phys} is strongly greater than the explicit time step imposed for stability. Here are some situations in which implicit methods are useful:

- Stiff source terms: because the time scale of the source term is too short, explicit methods are too expensive. As an example, combustion with detailed chemistry can lead to highly stiff source terms for intermediate species.

- Advection: implicit methods are not commonly used for accurate transient computations. However they can be of interest for low Mach flows when solved using compressible solver, because of the difference between the material waves u and and the acoustics waves $u \pm c$, the accuracy being limited to the material waves.

- Steady-state computations: by enabling to use very large time step, implicit methods allow to reach steady state quickly, the transient part being resolved poorly. It also avoids to deal with the change of nature of the PDE, e.g., the steady state of the parabolic unsteady heat equation is an elliptic equation, using a time-marching method.

- Unsteady diffusion (parabolic PDEs): for unsteady parabolic equation the limiting parameter is the Fourier number $Fo = a\frac{\Delta t}{\Delta x^2}$. In 1D, the stability limit of the explicit Euler method associated with a centered scheme in space is $Fo < \frac{1}{2}$. The time step is thus constraint by $\Delta t \leq \frac{\Delta x^2}{a}$, which is extremely penalising with mesh-refinement and is also a common

problem to all parabolic PDEs with explicit methods. The solution is then to use implicit methods.

Remark

Looking at the time step limitation due to stiffness as seen for ODEs, we get for the 1D unsteady diffusion with Euler+Centered:

$$\lambda_j = \frac{2a}{\Delta x^2}\left(\cos\left(\frac{j\pi}{N}\right) - 1\right)$$

The minimum and maximum eigenvalues are then $|\lambda_{min}| \approx \frac{\pi^2 a}{N^2 \Delta x^2}$ *and* $|\lambda_{max}| = \frac{4a}{\Delta x^2}$*. The condition number is thus:*

$$\frac{|\lambda_{max}|}{|\lambda_{min}|} = \frac{4N^2}{\pi^2} \gg 1$$

The problem thus becomes very stiff with mesh refinement, i.e., when increasing N*.*

6.2 Implicit methods applied to the unsteady heat equation

6.2.1 1D heat equation

Let us recall the 1D unsteady heat equation:

$$\frac{\partial \phi}{\partial t} = a\frac{\partial^2 \phi}{\partial x^2}$$

The simpler implicit method that can be envisaged to discretise this equation is to use a centered scheme for the space derivative associated with Backward Euler method:

$$\phi_i^{n+1} = \phi_i^n + Fo\left(\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}\right)$$

This scheme is first order in time and unconditionally stable. Rearranging the terms to write, we get:

1D unsteady heat equation: Backward Euler+Centered

$$-Fo\phi_{i-1}^{n+1} + (1+2Fo)\phi_i^{n+1} - Fo\phi_{i+1}^{n+1} = \phi_i^n$$

This is a linear system with a tridiagonal matrix [1]:

$$\begin{bmatrix} 1+2Fo & -Fo & & & & \\ -Fo & 1+2Fo & -Fo & & & \\ & \ddots & \ddots & \ddots & & \\ & & -Fo & 1+2Fo & -Fo & \\ & & & -Fo & 1+2Fo & \end{bmatrix} \begin{bmatrix} \\ \phi^{n+1} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \phi^n \\ \\ \end{bmatrix} \tag{6.1}$$

To solve this system, an efficient method is the Thomas algorithm of complexity $\mathcal{O}(N)$:

[1] Beware: first and last lines must be adapted properly to account for the boundary conditions

Thomas algorithm

The Thomas algorithm solves tridiagonal linear systems using Gaussian elimination in forward and backward sweeps. Considering the following tridiagonal system:

$$\begin{bmatrix} b_0 & c_0 & & & & \\ a_0 & b_1 & c_1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & a_{N-3} & b_{N-2} & c_{N-2} & \\ & & & a_{N-2} & b_{N-1} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} d_0 \\ \vdots \\ d_{N-1} \end{bmatrix}$$

the forward sweep consists in performing Gaussian elimination from the first to the last line:

First line: $b_0 x_0 + c_0 x_1 = d_0 \rightarrow x_0 = \dfrac{d_0 - c_0 x_1}{b_0}$

Second line: $\left(b_1 - a_1 \dfrac{c_0}{b_0}\right) x_1 + c_1 x_2 = d_1 - a_1 \dfrac{d_0}{b_0} \Rightarrow b_1' x_1 + c_1 x_2 = d_1'$

...

This first sweep leads to the following bidiagonal system:

$$\begin{bmatrix} b_0' & c_0 & & & \\ & b_1' & c_1 & & \\ & \ddots & \ddots & \ddots & \\ & & & b_{N-2}' & c_{N-2} \\ & & & & b_{N-1}' \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} d_0' \\ \vdots \\ d_{N-1}' \end{bmatrix}$$

which last line is solved in a straightforward manner:

$$x_{N-1} = \frac{d_{N-1}'}{b_{N-1}'}$$

and the backward sweep will end the resolution of the linear system:

$$x_{N-2} = \frac{d_{N-2}' - c_{N-2} x_{N-1}}{b_{N-1}'}$$

To achieve second order in time while ensuring unconditional stability, another discretization consists in using the Trapezoidal method for the time derivative:

$$\phi_i^{n+1} = \phi_i^n + \frac{Fo}{2}\left(\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}\right) + \frac{Fo}{2}\left(\phi_{i+1}^{n} - 2\phi_i^{n} + \phi_{i-1}^{n}\right) \tag{6.2}$$

which leads to the so-called Crank-Nicolson method, second order in time and space:

Crank-Nicolson method

$$-\frac{Fo}{2}\phi_{i-1}^{n+1} + (1+Fo)\phi_i^{n+1} - \frac{Fo}{2}\phi_{i+1}^{n+1} = \frac{Fo}{2}\phi_{i-1}^{n} + (1-Fo)\phi_i^{n} + \frac{Fo}{2}\phi_{i+1}^{n}$$

The Crank-Nicolson can also take advantage of the Thomas algorithm to solve the resulting tridiagonal system.

Remark

Similar 1D implicit schemes for advection can be found, which are solved using the Thomas algorithm for 3-points stencil.

6.2.2 2D unsteady heat equation

Let us now recall the 2D unsteady heat equation:

$$\frac{\partial \phi}{\partial t} = a\left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}\right)$$

Again, the simplest approach is to use Backward Euler for the time derivative:

2D unsteady heat equation: Backward Euler+Centered

$$-Fo\phi_{i+1,j}^{n+1} - Fo\phi_{i-1,j}^{n+1} + (1+4Fo)\phi_{i,j}^{n+1} - Fo\phi_{i,j+1}^{n+1} - Fo\phi_{i,j-1}^{n+1} = \phi_{i,j}^{n}$$

or also the trapezoidal rule leading to the Crank-Nicolson method:

2D unsteady heat equation: Backward Euler+Centered

$$\begin{aligned}-\frac{Fo}{2}\phi_{i+1,j}^{n+1} - \frac{Fo}{2}\phi_{i-1,j}^{n+1} + (1+2Fo)\phi_{i,j}^{n+1} - \frac{Fo}{2}\phi_{i,j+1}^{n+1} - \frac{Fo}{2}\phi_{i,j-1}^{n+1} = \\ \frac{Fo}{2}\phi_{i+1,j}^{n} + \frac{Fo}{2}\phi_{i-1,j}^{n} + (1-2Fo)\phi_{i,j}^{n} + \frac{Fo}{2}\phi_{i,j+1}^{n} + \frac{Fo}{2}\phi_{i,j-1}^{n}\end{aligned}$$

For both Backward Euler and Crank-Nicolson methods, the linear system is much larger than in the 1D case and is not tridiagonal anymore. To solve this system, there are three solutions:

- the direct resolution, which is way too expensive;
- the use of iterative methods at each time step, which it is still expensive,
- the use of approximate factorisation such as Alternate Direction Implicit method.

6.3 ADI method

The ADI (Alternate Direction Implicit) method has been proposed by Peaceman and Rachford (1955) to solve the 2D heat equation::

ADI method

The ADI method method is based on two steps:

$$\phi_{i,j}^{n+1/2} = \phi_{i,j}^{n} + \frac{a\Delta t}{2}\left(\left.\frac{\partial^2\phi}{\partial x^2}\right|_{ij}^{n+1/2} + \left.\frac{\partial^2\phi}{\partial y^2}\right|_{ij}^{n}\right)$$
$$\phi_{i,j}^{n+1} = \phi_{i,j}^{n+1/2} + \frac{a\Delta t}{2}\left(\left.\frac{\partial^2\phi}{\partial x^2}\right|_{ij}^{n+1/2} + \left.\frac{\partial^2\phi}{\partial y^2}\right|_{ij}^{n+1}\right)$$

The first step is implicit in the direction x and explicit in the direction y, the second step is explicit in the direction x and implicit in the direction y, hence the name ADI.

This method is particularly interesting because:

- it is 2[nd]order accurate in time, and in fact approximates the Crank-Nicolson method,
- it is unconditionally stable,
- The problem is decomposed into 1D implicit problems that are much cheaper to solve numerically

To summarise, the ADI method has all the benefits of the Crank-Nicolson method with no drawback.

Remark

The direct application of Peacemond and Rachford formula in 3D is NOT unconditionally stable and is thus useless. To generalize the ADI formula to 3D equation, two approaches are available: approximate factorization, and splliting/fractional step methods.

6.3.1 Space discretization

Let us introduce the vectorial notation of a space discretization operator:

$$\left[\left.\frac{\partial^2\phi}{\partial x^2}\right|_{i,j}\right] = A_x\left[\phi_{i,j}\right], \qquad \left[\left.\frac{\partial^2\phi}{\partial y^2}\right|_{i,j}\right] = A_y\left[\phi_{i,j}\right]$$

Example : For the 2[nd]-order centered formula, we get $\left(A_x\phi\right)|_{i,j} = \frac{\phi_{i+1,j}-2\phi_{i,j}+\phi_{i-1,j}}{\Delta x^2}$

Then, let us rewrite the update formula of the ADI method:

$$\left\{\begin{array}{lll} \phi^{n+1/2} & = \phi^{n} & + \frac{a\Delta t}{2}\left(A_x\phi^{n+1/2} + A_y\phi^{n}\right) \\ \phi^{n+1} & = \phi^{n+1/2} & + \frac{a\Delta t}{2}\left(A_x\phi^{n+1/2} + A_y\phi^{n+1}\right) \end{array}\right.$$

or

$$\left\{\begin{array}{ll} (I - \frac{a\Delta t}{2}A_x)\phi^{n+1/2} & = (I + \frac{a\Delta t}{2}A_y)\phi^{n} \\ (I - \frac{a\Delta t}{2}A_y)\phi^{n+1} & = (I + \frac{a\Delta t}{2}A_x)\phi^{n+1/2} \end{array}\right.$$

or

$$\left\{\begin{array}{ll} \phi^{n+1/2} & = (I - \frac{a\Delta t}{2}A_x)^{-1}(I + \frac{a\Delta t}{2}A_y)\phi^{n} \\ \phi^{n+1} & = (I - \frac{a\Delta t}{2}A_y)^{-1}(I + \frac{a\Delta t}{2}A_x)\phi^{n+1/2} \end{array}\right.$$

Therefore the ADI method is finally written as:

$$\phi^{n+1} = (I - \frac{a\Delta t}{2}A_y)^{-1}(I + \frac{a\Delta t}{2}A_x)(I - \frac{a\Delta t}{2}A_x)^{-1}(I + \frac{a\Delta t}{2}A_y)\phi^n$$

or

$$\phi^{n+1} = (I - \frac{a\Delta t}{2}A_y)^{-1}(I - \frac{a\Delta t}{2}A_x)^{-1}(I + \frac{a\Delta t}{2}A_x)(I + \frac{a\Delta t}{2}A_y)\phi^n$$

as $(I + \frac{a\Delta t}{2}A_x)$ and $(I - \frac{a\Delta t}{2}A_x)^{-1}$ commute.
Using a Centered 2nd order space discretization, we get:

2nd order in time and space ADI method

$$\left\{ \begin{array}{rcl} -\frac{Fo}{2}\phi_{i-1,j}^{n+1/2} + (1+Fo)\phi_{i,j}^{n+1/2} - \frac{Fo}{2}\phi_{i+1,j}^{n+1/2} & = & \frac{Fo}{2}\phi_{i,j-1}^{n} + (1-Fo)\phi_{i,j}^{n} + \frac{Fo}{2}\phi_{i,j+1}^{n} \\ -\frac{Fo}{2}\phi_{i,j-1}^{n+1} + (1+Fo)\phi_{i,j}^{n+1} - \frac{Fo}{2}\phi_{i,j+1}^{n+1} & = & \frac{Fo}{2}\phi_{i-1,j}^{n+1/2} + (1-Fo)\phi_{i,j}^{n+1/2} + \frac{Fo}{2}\phi_{i+1,j}^{n+1/2} \end{array} \right.$$

This scheme requires:

- 1st step: loop over $j \in [0, M-1]$ and solve M tridiagonal problems using Thomas algorithm in x direction;
- 2nd step: loop over $i \in [0, N-1]$ and solve N tridiagonal problems using Thomas algorithm in x direction;

This scheme is thus far more efficient than solving the full implicit scheme seen previously.

6.3.2 Properties of ADI

There are several possible ADI formula, here we study the one that has been given previously.

6.3.2.1 Accuracy

Summing the two steps of the ADI methods, we get:

$$\phi_{i,j}^{n+1} = \phi_{i,j}^{n} + a\Delta t \left.\frac{\partial^2\phi}{\partial x^2}\right|_{i,j}^{n+1/2} + \frac{a\Delta t}{2}\left(\left.\frac{\partial^2\phi}{\partial y^2}\right|_{i,j}^{n+1} + \left.\frac{\partial^2\phi}{\partial y^2}\right|_{i,j}^{n} \right)$$

Therefore:

$$\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^{n}}{\Delta t} = a \left.\frac{\partial^2\phi}{\partial x^2}\right|_{i,j}^{n+1/2} + \frac{a}{2}\left(\left.\frac{\partial^2\phi}{\partial y^2}\right|_{i,j}^{n+1} + \left.\frac{\partial^2\phi}{\partial y^2}\right|_{i,j}^{n} \right)$$

Concerning the left hand side, considering that this approximation is a centered approximation around time $t_{n+1/2}$, we get:

$$\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^{n}}{\Delta t} = \frac{\partial \phi_{i,j}(t_{n+1/2})}{\partial t} + \mathcal{O}(\Delta t^2)$$

Concerning the second term of the right hand side, we get:

$$\left.\frac{\partial^2\phi}{\partial y^2}\right|_{i,j}^{n+1} = \left.\frac{\partial^2\phi}{\partial y^2}\right|_{i,j}^{n+1/2} - \Delta t \frac{\partial}{\partial t}\left.\left(\frac{\partial^2\phi}{\partial y^2}\right)\right|_{i,j}^{n+1/2} + \mathcal{O}(\Delta t^2)$$

$$\left.\frac{\partial^2\phi}{\partial y^2}\right|_{i,j}^{n} = \left.\frac{\partial^2\phi}{\partial y^2}\right|_{i,j}^{n+1/2} + \Delta t \frac{\partial}{\partial t}\left.\left(\frac{\partial^2\phi}{\partial y^2}\right)\right|_{i,j}^{n+1/2} + \mathcal{O}(\Delta t^2)$$

We finally get:

$$\frac{\partial \phi_{i,j}(t_{n+1/2})}{\partial t} = a \left(\left. \frac{\partial^2 \phi}{\partial x^2} \right|_{i,j}^{n+1/2} + \left. \frac{\partial^2 \phi}{\partial y^2} \right|_{i,j}^{n+1/2} \right) + \mathcal{O}(\Delta t^2)$$

The scheme is then 2^{nd} order accurate in time.

Remark

- *In practice, to balance numerical errors in all directions, the first implicit direction alternates between x and y at each iteration,*
- *Not all ADI methods are 2^{nd} order accurate in time.*

6.3.3 Stability

Through the modified wave number analysis, we obtain $A_x\phi = -(k'_x)^2\phi$ and $A_y\phi = -(k'_y)^2\phi$. Applying it to the ADI method, we get:

$$(1 + \frac{a\Delta t}{2}(k'_x)^2)(1 + \frac{a\Delta t}{2}(k'_y)^2)\phi^{n+1} = (1 - \frac{a\Delta t}{2}(k'_x)^2)(1 - \frac{a\Delta t}{2}(k'_y)^2)\phi^n$$

Then:

$$\phi^{n+1} = \sigma \phi^n$$

with:

$$\sigma = \frac{(1 - \frac{a\Delta t}{2}(k'_x)^2)(1 - \frac{a\Delta t}{2}(k'_y)^2)}{(1 + \frac{a\Delta t}{2}(k'_x)^2)(1 + \frac{a\Delta t}{2}(k'_y)^2)}$$

As $|\sigma| \leq 1$ is always verified, the ADI method shown here is unconditionally stable.

Remark

Not all ADI formula are unconditionally stable.

6.3.4 Boundary conditions

The boundary conditions to be applied on ϕ^n and ϕ^{n+1} are clear, as these are two solutions of the equations, but $\phi^{n+1/2}$ is not necessarily a solution on which the boundary conditions are to be applied directly. The two steps can be written:

$$\begin{cases} \phi_{i,j}^{n+1/2} - \frac{a\Delta t}{2} \left. \frac{\partial^2 \phi}{\partial x^2} \right|_{i,j}^{n+1/2} = \phi_{i,j}^{n} + \frac{a\Delta t}{2} \left. \frac{\partial^2 \phi}{\partial y^2} \right|_{i,j}^{n} \\ \phi_{i,j}^{n+1/2} + \frac{a\Delta t}{2} \left. \frac{\partial^2 \phi}{\partial x^2} \right|_{i,j}^{n+1/2} = \phi_{i,j}^{n+1} - \frac{a\Delta t}{2} \left. \frac{\partial^2 \phi}{\partial y^2} \right|_{i,j}^{n} \end{cases}$$

By summation, we get:

$$\phi_{i,j}^{n+1/2} = \frac{1}{2}\left(\phi_{i,j}^{n} + \phi_{i,j}^{n+1}\right) + \frac{a\Delta t}{2} \left(\left. \frac{\partial^2 \phi}{\partial y^2} \right|_{i,j}^{n} - \left. \frac{\partial^2 \phi}{\partial y^2} \right|_{i,j}^{n+1} \right)$$

The boundary conditions on $\phi^{n+1/2}$ must fulfil this relationship:

- if the boundary condition is uniform or linear, i.e. $\frac{\partial^2}{\partial y^2} = 0$, we get: $\phi_{i,j}^{n+1/2} = \frac{1}{2}\left(\phi_{i,j}^n + \phi_{i,j}^{n+1}\right)$
- if the boundary condition is neither uniform nor linear, we introduce a 1st order error if $\phi_{i,j}^{n+1/2} = \frac{1}{2}\left(\phi_{i,j}^n + \phi_{i,j}^{n+1}\right)$ or $\phi_{i,j}^{n+1/2} = \phi_{i,j}^n$.

Remark

All ADI formula require such attention on the treatment of boundary conditions to preserve the order of accuracy in time.

6.3.5 Steady state

When the implicit scheme is used to compute the steady state, it must be verified that the steady numerical solution fulfils the steady PDE, i.e. $\phi^{n+1} = \phi^n$:

$$(I - \frac{a\Delta t}{2}A_x)(I - \frac{a\Delta t}{2}A_y)\phi^n = (I + \frac{a\Delta t}{2}A_x)(I + \frac{a\Delta t}{2}A_y)\phi^n$$

$$(I - \frac{a\Delta t}{2}A_x - \frac{a\Delta t}{2}A_y + \frac{a^2\Delta t^2}{4}A_xA_y)\phi^n = (I + \frac{a\Delta t}{2}A_x - \frac{a\Delta t}{2}A_y + \frac{a^2\Delta t^2}{4}A_xA_y)\phi^n$$

We finally get:

$$a(A_x + A_y)\phi^n = 0 \Leftrightarrow a\left(\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2}\right) = 0$$

Remark

- *Some ADI formula leaves a truncation error $\mathcal{O}(\Delta t^\alpha)$ on the right hand side, the steady state that is computed is then affected by the chosen time step.*
- *Looking back to the previously introduced "modified Richardson method" and its generalisation with a preconditionner (see "revisited Richardson"), the ADI method can also be used to solve Elliptic PDEs where Δt is the coefficient of the modified Richardson and $K = (I - \frac{a\Delta t}{2}A_y)(I - \frac{a\Delta t}{2}A_x)$ is the preconditioning matrix corresponding to the ADI shown here.*

6.4 Generalization

A full implicit method can be written as $A\left[\phi_{i,j}\right] = B + \mathcal{O}(\Delta t^\alpha, \Delta x^\beta)$, which requires the inversion of the matrix A. To avoid a cumbersome inversion, the two possible approaches are the approximate factoring and the splitting methods.

6.4.1 Approximate Factoring

In approximate factoring, the objective is to modify A without modifying the order of accuracy, for example usin a succession of 1D implicit formula.

Example : Considering the Crank-Nicolson method:

$$\phi^{n+1} = \phi^n + \frac{a\Delta t}{2}A_x(\phi^{n+1} + \phi^n) + \frac{a\Delta t}{2}A_y(\phi^{n+1} + \phi^n) + \Delta t\mathcal{O}(\Delta t^2, \Delta x^2)$$

and rearranging the equation:

$$\left[I - \frac{a\Delta t}{2}A_x - \frac{a\Delta t}{2}A_y\right]\phi^{n+1} = \left[I + \frac{a\Delta t}{2}A_x + \frac{a\Delta t}{2}A_y\right]\phi^n$$

Inversing the matrix on the left hand side is difficult because it is not a tridiagonal matrix. In order to recover easily inversible matrices such as $I - \frac{a\Delta t}{2}A_x$, a factoring is performed on both right and left hand sides:

$$\left[I - \frac{a\Delta t}{2}A_x\right]\left[I - \frac{a\Delta t}{2}A_y\right]\phi^{n+1} - \frac{a^2\Delta t^2}{4}A_xA_y\phi^{n+1} = \left[I + \frac{a\Delta t}{2}A_x\right]\left[I + \frac{a\Delta t}{2}A_y\right]\phi^n - \frac{a^2\Delta t^2}{4}A_xA_y\phi^n + \Delta t\mathcal{O}(\Delta t^2, \Delta x^2)$$

Therefore:

$$\left[I - \frac{a\Delta t}{2}A_x\right]\left[I - \frac{a\Delta t}{2}A_y\right]\phi^{n+1} = \left[I + \frac{a\Delta t}{2}A_x\right]\left[I + \frac{a\Delta t}{2}A_y\right]\phi^n \underbrace{- \frac{a^2\Delta t^2}{4}A_xA_y\underbrace{(\phi^n - \phi^{n+1})}_{\mathcal{O}(\Delta t)}}_{\mathcal{O}(\Delta t^3)} + \Delta t\mathcal{O}(\Delta t^2, \Delta x^2)$$

Here the truncation error is then not penalized by neglicted the $\mathcal{O}(\Delta t^3)$-term. We thus get the approximate factoring of 2D Crank-Nicolson:

Approximate factoring of 2D Crank-Nicolson

$$\left[I - \frac{a\Delta t}{2}A_x\right]\left[I - \frac{a\Delta t}{2}A_y\right]\phi^{n+1} = \left[I + \frac{a\Delta t}{2}A_x\right]\left[I + \frac{a\Delta t}{2}A_y\right]\phi^n$$

Other formula can also be derived in the same spirit:

- Similar factoring of Crank-Nicolson in 3D, which leads to an unconditionally stable method,
- Factoring of 2D or 3D Backward Euler,
- Application to advection problems.
- ...etc...

Remark

For all approximate factoring, one must characterize:

- *the order of accuracy of the approximation,*
- *the stability,*
- *the consistency of the boundary conditions to apply at intermediate steps,*
- *the consistency with the steady state.*

6.4.2 Splitting methods

When several terms appear o the right hand side, the splitting methods, aka Fractional step methods, treat each term individually and separately with fractional steps:

Example :

- 1^{st} step: Backward Euler of the full time step Δt to solve $\frac{\partial \phi}{\partial t} = a\frac{\partial^2 \phi}{\partial x^2}$,
- 2^{nd} step: Backward Euler of the full time step Δt to solve $\frac{\partial \phi}{\partial t} = a\frac{\partial^2 \phi}{\partial y^2}$,

When they are applied to spatial derivative operators in different directions, they are named LOD methods where LOD stands for Local One Dimensional.

Chapter 7

Numerical resolution of incompressible fluids

Contents

7.1 Fundamental equations for fluid dynamics

Here we consider continuous phases that can be either liquid or gaseous. The equations are limited to single phase flows.
The first equations states the conversation of mass:

Mass conservation

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = 0$$

where ρ is the fluid density and u_i the i^{th} component of the fluid velocity.

The second equation states the conservation of momentum, and is derived from the second Newton's law:

Momentum equation

$$\frac{\partial \rho u_j}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_i} = -\frac{\partial P}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho g_j$$

where P is the thermodynamical pressure, τ is the shear stress tensor and g is the acceleration due to volumetric forces such as gravity.

P must satisfy an equation of state that depends on the fluid that is considered:

- Perfect gases: $P = \rho \frac{R}{W} T$, where R is the gas constant and W the molar mass of the specified gas,
- Real gases: for instance the Wan der Walls equation can be used: $\left(P + \frac{a}{V_m}\right)(V_m - b) = RT$ where a is the cohesion term, b is the molar covolume and V_m is the molar volume,
- Liquid: for instance we can use the Murnagham-Tait equation of state: $P = \frac{K_0}{n}\left(\left(\frac{\rho}{\rho_0}\right)^n - 1\right) + P_0$ where P_0 is the reference pressure, K_0 is the bulk modulus, and n a material parameter.

The stress tensor τ depends on the fluid rheology:

- Newtonian fluids: $\tau_{ij} = 2\mu S_{ij}$ where $S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{1}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}$ is the shear tensor and μ is the dynamic viscosity of the fluid.
- Non-Newtonian fluids: for these fluids, the stress tensor is determined by a non-linear relationship with the shear tensor S_{ij}, i.e., $\mu = \mu(S_{ij})$.

Remark

The momentum equation associated with a Newtonian fluid assumption leads to the so-called Navier-Stokes equation.

The last conservation equation is derive from the first principle of thermodynamic: the energy conservation. Many forms can be envisaged depending on the considered energy

Energy conservation

$$\frac{\partial \rho(e + \frac{1}{2}u_j^2)}{\partial t} + \frac{\partial \rho u_i (e + \frac{1}{2}u_j^2)}{\partial x_i} = -\frac{\partial P u_j}{\partial x_j} - \frac{\partial q_j}{\partial x_j} + \frac{\partial \tau_{ij} u_i}{\partial x_j} + \rho g_j u_j$$

where e is the internal energy of the fluid and q the heat flux.

To close the heat flux, the Fourier's law is used: $q_i = -\lambda \frac{\partial T}{\partial x_i}$ where T is the temperature and λ is the thermal conductivity of the fluid.
The final system of equations is then:

Governing equations for a fluid

$$\begin{cases} \dfrac{\partial \rho}{\partial t} & + \dfrac{\partial \rho u_i}{\partial x_i} & = 0 & & & \\ \dfrac{\partial \rho u_j}{\partial t} & + \dfrac{\partial \rho u_i u_j}{\partial x_i} & = -\dfrac{\partial P}{\partial x_j} & + \dfrac{\partial \tau_{ij}}{\partial x_j} & + \rho g_j & \\ \underbrace{\dfrac{\partial \rho(e + \frac{1}{2}u_j^2)}{\partial t}}_{\text{time evolution}} & \underbrace{+ \dfrac{\partial \rho u_i (e + \frac{1}{2}u_j^2)}{\partial x_i}}_{\text{inertia}} & = \underbrace{-\dfrac{\partial P u_j}{\partial x_j}}_{\text{pressure}} & \underbrace{+ \dfrac{\partial \tau_{ij} u_i}{\partial x_j}}_{\text{viscous term}} & \underbrace{- \dfrac{\partial q_j}{\partial x_j}}_{\text{heat fluxes}} & + \underbrace{\rho g_j u_j}_{\text{volume forces}} \\ & \underbrace{\qquad\qquad\qquad\qquad}_{\text{hyperbolic part}} & & \underbrace{\qquad\qquad\qquad\qquad}_{\text{parabolic part}} & & \end{cases}$$

Starting from this set of equations, the behaviour of the system depends on the chosen asssumption:

- Hyperbolic or parabolic character depending on the balance between the inertia and viscous terms,
- Purely hyperbolic in the case of negligible viscosity and conductivity, which leads to the so-called Euler equations,
- Elliptic in the zero Reynolds number limit (Stokes flow).

7.2 Incompressible flows

There are situations in which the variations of density may be considered as negligible, for instance for liquid flows, for which the density can be neglected; or for gaseous flows at Low-Mach number for which the acoustics may be assumed to be infinitely fast compared to the material waves. In the following, we present the equations under the assumption of constant density:

$$\rho = \rho_0 = \text{cst} \Rightarrow \frac{\partial \rho}{\partial t} = 0$$

Using this assumption, the equations of continuity and momentum become (under the newtonian fluid assumption):

Governing equations for incompressible flows

$$\left\{ \begin{array}{rcl} \sum_{i=1}^{N_{dim}} \dfrac{\partial u_i}{\partial x_i} & = & 0 \\ \dfrac{\partial u_j}{\partial t} + \sum_{i=1}^{N_{dim}} \dfrac{\partial u_i u_j}{\partial x_i} & = & -\dfrac{1}{\rho_0}\dfrac{\partial P}{\partial x_j} + \nu \sum_{i=1}^{N_{dim}} \dfrac{\partial^2 u_j}{\partial x_i^2} \end{array} \right.$$

Here it can be seen that to solve the velocity field, the energy equation is not required anymore, as we already have $N_{dim}+1$ equations and $N_{dim}+1$ variables, the velocity components u_i and the pressure P. However, we do not have an independant equation for P. Let us derive the equation of P by taking the divergence of the momentum equation:

$$\underbrace{\frac{\partial}{\partial t}\left(\sum_{j=1}^{N_{dim}} \frac{\partial u_j}{\partial x_j}\right)}_{=0} + \sum_{i=1}^{N_{dim}}\sum_{j=1}^{N_{dim}} \frac{\partial}{\partial x_j}\left(\frac{\partial u_i u_j}{\partial x_i}\right) = -\frac{1}{\rho_0}\sum_{j=1}^{N_{dim}} \frac{\partial}{\partial x_j}\left(\frac{\partial P}{\partial x_j}\right) + \nu \sum_{i=1}^{N_{dim}}\sum_{j=1}^{N_{dim}} \frac{\partial^2}{\partial x_i^2}\underbrace{\left(\frac{\partial u_j}{\partial x_j}\right)}_{=0}$$

and we finally get the following elliptic equation on the pressure:

Pressure equation

$$-\frac{1}{\rho_0}\sum_{i=1}^{N_{dim}} \frac{\partial^2 P}{\partial x_i \partial x_i} = \sum_{i=1}^{N_{dim}}\sum_{j=1}^{N_{dim}} \frac{\partial}{\partial x_j}\left(\frac{\partial u_i u_j}{\partial x_i}\right)$$

Remark

- *The equations can also be rigorously obtained by an asymptotic analysis in the low-Mach number limit,*
- *the solution does not depend on the absolute value of P but only on its gradients,*
- *P is no longer the thermodynamical pressure.*

7.3 Solution methods for incompressible Navier-Stokes equations

To solve the Navier-Stokes equations in the incompressible limit, the mai issue is the pressure-velocity coupling that must ensure the divergence-free condition. Four methods are available in the literature:

- Pressure correction,
- Fractional step or projection,
- Artificial compressibility,
- Stream function and vorticity.

7.3.1 Pressure-correction methods

In pressure-correction methods, the objective is to find the pressure field that ensures a divergence-free updated velocity. Let us consider a semi-discretization using Forward Euler for the time derivative:

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = -\sum_{j=1}^{N_{dim}} \frac{\delta u_i^n u_j^n}{\delta x_j} - \frac{1}{\rho_0}\frac{\delta P^n}{\delta x_i} + \nu \sum_{j=1}^{N_{dim}} \frac{\delta^2 u_i^n}{\delta x_j^2}$$

where i is the velocity component index and $\frac{\delta}{\delta x_j}$ is the numerical approximation of the partial derivative $\frac{\partial}{\partial x_j}$. To find P^n such that the divergence-free condition is ensured, i.e., $\sum_{i=1}^{N_{dim}} \frac{\partial u_i}{\partial x_i} = 0$, we take the numerical divergence of this equation:

$$\sum_{i=1}^{N_{dim}} \frac{\frac{\delta u_i^{n+1}}{\delta x_i} - \frac{\delta u_i^n}{\delta x_i}}{\Delta t} = -\sum_{i=1}^{N_{dim}}\sum_{j=1}^{N_{dim}} \frac{\delta}{\delta x_i}\left(\frac{\delta u_i^n u_j^n}{\delta x_j}\right) - \frac{1}{\rho_0}\sum_{i=1}^{N_{dim}} \frac{\delta^2 P^n}{\delta x_i^2} + \nu \sum_{i=1}^{N_{dim}}\sum_{j=1}^{N_{dim}} \frac{\delta^2}{\delta x_j^2}\left(\frac{\delta u_i^n}{\delta x_i}\right)$$

Considering that the initial field at instant n must be divergence-free, and that the final field at instant $n+1$ must also be divergence-free, we finally get:

$$\frac{1}{\rho_0}\sum_{i=1}^{N_{dim}} \frac{\delta^2 P^n}{\delta x_i^2} = -\sum_{i=1}^{N_{dim}}\sum_{j=1}^{N_{dim}} \frac{\delta}{\delta x_i}\left(\frac{\delta u_i^n u_j^n}{\delta x_j}\right)$$

which is in fact the discretized pressure equation. Let us summarize the algorithm:

1. Start with a divergence-free velocity field
2. Compute the advective and viscous term in the velocity equation
3. solve the Poisson equation to get the pressure
4. update the velocity equation

7.3.2 Fractional step of projection methods

This type of method can be seen as a splitting method or an approximate factorization. Here we present a three-step strategy:

Fractional step methods

step 1: for $i = 1, N_{dim}$ $\quad u_i^* = u_i^n - \Delta t \sum_{j=1}^{N_{dim}} \frac{\delta u_i^n u_j^n}{\delta x_j}$ hyperbolic equation

step 2: for $i = 1, N_{dim}$ $\quad u_i^{**} = u_i^* + \nu \Delta t \sum_{j=1}^{N_{dim}} \frac{\delta^2 u_i^*}{\delta x_j^2}$ parabolic equation

step 3: for $i = 1, N_{dim}$ $\quad u_i^{n+1} = u_i^{**} - \frac{\Delta t}{\rho_0} \frac{\delta P}{\delta x_i}$ require P

To find P, we take the divergence of the third step:

$$\sum_{i=1}^{N_{dim}} \frac{\delta u_i^{n+1}}{\delta x_i} = \sum_{i=1}^{N_{dim}} \frac{\delta u_i^{**}}{\delta x_i} - \frac{\Delta t}{\rho_0} \sum_{i=1}^{N_{dim}} \frac{\delta^2 P}{\delta x_i^2}$$

Here we want the updated velocity field u_i^{n+1} to be divergence-free, but the initial one u_i^{**} is not divergence-free in this case. We get finally the elliptic equation on the pressure P:

Discrete Pressure equation for fractional step

$$\sum_{i=1}^{N_{dim}} \frac{\delta^2 P}{\delta x_i^2} = \frac{\rho_0}{\Delta t} \sum_{i=1}^{N_{dim}} \frac{\delta u_i^{**}}{\delta x_i}$$

Remark

There exist many fractional step methods depending on the way the equation is splitted

7.3.3 Artificial compressibility method

The main issue in incompressible flows is that we do not have an evolution equation for the pressure. The idea behind artificial compressibility methods is to modify the divergence-free condition in order to get an evolution equation:

$$\frac{1}{\beta} \frac{\partial P}{\partial t} + \sum_{i=1}^{N_{dim}} \frac{\partial \rho_0 u_i}{\partial x_i} = 0$$

In this equation β is the artificial compressibility parameter. When β tends to infinity, we recover the original divergence-free condition, so β must be chosen as large as possible. The resulting set of equations is:

Artificial compressibility method

$$\frac{\partial P}{\partial t} + \beta\rho_0 \sum_{i=1}^{N_{dim}} \frac{\partial u_i}{\partial x_i} = 0$$
$$\frac{\partial u_j}{\partial t} + \sum_{i=1}^{N_{dim}} \frac{\partial u_i u_j}{\partial x_i} = -\frac{1}{\rho_0}\frac{\partial P}{\partial x_j} + \nu \sum_{i=1}^{N_{dim}} \frac{\partial^2 u_j}{\partial x_i^2}$$

and is not requiring to solve a Poisson equation anymore. However it introduces high stiffness, which leads to the use of implicit methods to solve the first equation.

7.3.4 Stream function and vorticity method

This method is only of interest for 2D flows, and is applicable in the case of constant fluid properties. Let us define the stream function Ψ and the vorticity ω:

$$\frac{\partial \Psi}{\partial x} = u_x, \quad \frac{\partial \Psi}{\partial y} = -u_y$$
$$\omega = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}$$

Rewriting the conservation equation in terms of Ψ and ω, we get:

Stream function-vorticity formulation

$$\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} = -\omega$$
$$\frac{\partial \omega}{\partial t} + u_x\frac{\partial \omega}{\partial x} + u_y\frac{\partial \omega}{\partial y} = \nu\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right)$$

Compared to the initial set of equations, the problem is reduced to 2 equations. It is still requiring to solve an elliptic equation but there is no issue of pressure-velocity coupling anymore.

Remark

In 3D, the stream function and vorticity have 3 components, it then leads to a system of 6 equations in place of the initial 4 equations, so this formulation is useless (especially with variable properties).

Part II

Exercises

Chapter 8

Introduction to Python

In this workshop, an introduction is given to Python programming. The three main elements of interest here are the use of arrays and control statements, and the use of plotting capacities in Python.

8.1 Definition and plotting of a function

8.1.1 First-dimensional case

We first write a code that evaluates the function $f(x) = x^3 + x^2 + x + 1$

1. Define a 1D array x containing N_x points between -1 and 1 (including first and end points).
2. Define the 1D array f which contains the evaluation of $f(x)$ at each point of array x.
3. Plot the function f versus x. Add labels to x and y axes, and add the title '$f(x) = x^3 + x^2 + x + 1$'.
4. Change line style to red dashed line, and line width to 3.

8.1.2 Two-dimensional case

The function now depends on two variables: $g(x, y) = x^2 + y^2 - 2xy$.

1. Construct the mesh point positions x and y of the 2d space $[-1, 1]^2$ using the function mgrid.
2. Define the 2D array g which contains the evaluation of $g(x, y)$ at each point of array (x, y).
3. Plot the function g versus x and y using a surface plot. Add labels to x and y axes, and add the title '$g(x) = x^2 + y^2 - 2xy$'.
4. Synthesise the 1D and 2D cases in a figure with two sub-plots.

8.2 Evaluation of the integral of a function

To evaluate the integral of a function the two most classical solution are the rectangle method and the trapezoidal rule. Considering that the integral F of a function $f(x)$ over an interval $[-1, 1]$ can be decomposed into integrals over $N_x - 1$ sub-intervals $[x_{i-1}, x_i]$:

$$F = \int_{-1}^{1} f(x)\mathrm{d}x = \sum_{k=1}^{N_x - 1} \int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \tag{8.1}$$

The integral over each interval can be evaluated by considering either a constant function (rectangle method) or a linear function (trapezoidal rule) between end points. The final formulas are then:

$$\text{Rectangle Method :} \quad F \approx \sum_{k=1}^{N_x-1} f(x_i)\Delta x \tag{8.2}$$

$$\text{Trapezoidal Rule :} \quad F \approx \sum_{k=1}^{N_x-1} \frac{f(x_i)+f(x_i+1)}{2}\Delta x \tag{8.3}$$

1. Implement both formulas to evaluate the integral of the function $f(x)$.
2. To evaluate the order of each method, evaluate the error for different number of points.
3. Plot the results in a log-log representation.

8.3 Root-finding algorithms

In this section, we want to find the root x^* of the equation $f(x) = 0$. Three method are considered : the fixed point iteration, the Newton method and the secant method. All methods end iterations when a sufficient accuracy is achieved on the equation to be solved. The desired accuracy is determined by a user-defined threshold. The convergence of the algorithm can strongly depend on the initial guess x^0 used to start the iterations.

1. Fixed point iteration transforms the problem into $x = g(x)$ and performs the following iterations
$$x^{n+1} = g(x_n)$$
Implement a fixed point iteration with $g(x) = f(x) + x$ starting from an initial guess x^0.
2. For a given guess for the solution x^n, the Newtons' method performs the following iterations:
$$x^{n+1} = x^n - f(x^n)/f'(x^n) \tag{8.4}$$
Implement the Newton method for the function $f(x)$ starting from an initial guess x^0.
3. The secant method is given by
$$x^{n+1} = x^n - f(x^n)\frac{x^n - x^{n-1}}{f(x^n) - f(x^{n-1})}$$
Implement the secant method for the function $f(x)$ starting from two initial guesses x^0 and x^1.
4. These methods are characterised by different rates of convergence α where $|e^{n+1}| \approx C|e^n|^\alpha$ and with $e^n = x^n - x^*$ the error at the n$^{\text{th}}$ iteration. Plot $\ln(e^{n+1}/e^n)$ and determine α for each method, knowing that $\ln(|e^{n+1}/e^n|) = \alpha \ln(|e^n/e^{n-1}|)$.

Chapter 9

Finite differences

The first objective of this workshop is to implement and evaluate finite differences methods. To do so, the following analytical solution is investigated:

$$f(x) = \frac{\sin(x)}{x^3} \tag{9.1}$$

This function has the following first and second order derivatives:

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \frac{x\cos(x) - 3\sin(x)}{x^4}, \qquad \frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} = \frac{-x^2\sin(x) - 6x\cos(x) + 12\sin(x)}{x^5} \tag{9.2}$$

The derivation of finite difference formulae is based on the Taylor-Expansion of a function around at a specific position:

$$f(x+h) = f(x) + \sum_{k=1}^{\infty} \frac{h^k}{k!} \left.\frac{\mathrm{d}^k f}{\mathrm{d}x^k}\right|_x \tag{9.3}$$

where h is the uniform spacing between each grid points.

9.1 First order derivatives

We first evaluate finite difference formula for the first order derivative $\frac{\mathrm{d}f}{\mathrm{d}x}$. Considering the following first and second order methods:

$$1^{st} \text{ order:} \quad \frac{\mathrm{d}f}{\mathrm{d}x} = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h) \tag{9.4}$$

$$2^{nd} \text{ order:} \quad \frac{\mathrm{d}f}{\mathrm{d}x} = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2) \tag{9.5}$$

The objective is to write a python code in order to evaluate their accuracy:

1. In a module file, define two separated functions that evaluate the finite difference approximation of $\frac{\mathrm{d}f}{\mathrm{d}x}$ at first order and at second order. The input arguments of each function are the point where to evaluate the derivative x_0, the spacing h and the function to evaluate $func$.
2. In a main script file, import all functions of the previously defined module.
3. Define the functions f(x) and $\frac{\mathrm{d}f}{\mathrm{d}x}$.
4. Define the point $x_0 = 4$ where all derivatives are evaluated.

5. For different spacings h, evaluate the error made by the two finite difference methods.
6. Plot errors of each method with the adequate labels and titles, and conclude on the order of accuracy.
7. Using Taylor tables, write a fourth order approximation of $\frac{\mathrm{d}f}{\mathrm{d}x}$ of stencil 4Δ.
8. Implement and evaluate this fourth order method. Plot the error.

9.2 Second order derivatives

For a second order derivative $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$, we consider the two following second order methods with stencils $2h$ and $4h$:

$$2^{nd} \text{ order (2h):} \quad \frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \mathcal{O}(h^2) \tag{9.6}$$

$$2^{nd} \text{ order (4h):} \quad \frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = \frac{f(x+2h) - 2f(x) + f(x-2h)}{4h^2} + \mathcal{O}(h^2) \tag{9.7}$$

9. In your finite difference module file, add these two formula.
10. In your main script, define the function $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$.
11. For different spacing h, evaluate the error made by the two finite difference methods.
12. Plot errors of each method will the adequate labels and titles, and conclude on the order of accuracy.
13. Using Taylor tables, write a fourth order approximation of $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$ of stencil 4Δ.
14. Implement and evaluate this fourth order method. Plot the error.

Chapter 10

Ordinary Differential Equations

10.1 A first simple example of ODE

Let us consider the following initial value problem:

$$\frac{\mathrm{d}Y(t)}{\mathrm{d}t} = F(t, Y(t)) \tag{10.1}$$

$$Y(t=0) = Y_0 \tag{10.2}$$

Considering a time-marching method, we use a discretized time vector $t_n = n\Delta t$. By integrating in time over the time interval $[t_n, t_{n+1}]$, we obtain the following update formula:

$$Y(t_{n+1}) = Y_n + \int_{t_n}^{t_{n+1}} F(\tau, Y(\tau))\mathrm{d}\tau \tag{10.3}$$

The time evolution of the right hand side is required to close the equation. The simpler strategy consists in considering that $F(\tau, Y(\tau)) = F(t_n, Y(t_n))$, which leads to the following scheme know as the Forward Euler's method:

$$Y(t_{n+1}) = Y_n + \Delta t F(t_n, Y(t_n)) \tag{10.4}$$

As a first introduction, we will investigate the following scalar ODE:

$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = -y \tag{10.5}$$

1. Find the analytical solution of $y(t)$ with the initial condition $y(t=0) = y_0$

2. Implement the Forward Euler's method. The initial condition is $y_0 = 1$. Choose an initial time step $\Delta t = 0.1s$ and a final time $T_{max} = 4s$.

3. Compare the results of the Euler's method to the analytical solution.

4. For different time steps, evaluate the error of this method compared to the analytical solution, and plot it. What is the order of accuracy?

5. What happen when the time step is larger than $1s$? Find K such that the forward Euler's method is written in the form $y^n = K^n y_0$. What is the constraint on K for the solution to be bounded?What is the consecutive constraint on Δt?

10.2 Harmonic oscillator

The first problem is devoted to the study of harmonic oscillators, such as spring-mass systems:

$$\ddot{x} + \omega^2 x = 0 \tag{10.6}$$

where ω is the characteristic wave number of the system. The objective is to solve this equation using ODE solvers, and to compare the results in terms of errors and stability. The analytical solution is:

$$x(t) = A\cos(\omega t + \varphi) \tag{10.7}$$

where $A = \sqrt{x_0^2 + \frac{u_0^2}{\omega^2}}$ is the amplitude and $\varphi = \arctan(-\frac{u_0}{x_0\omega})$ is the initial phase lag, which depend on the initial position $x_0 = x(t=0)$ and the initial velocity $u_0 = \dot{x}(t=0)$.
To solve the ODE, we will use a time-marching method. To do so, the equation is classically splitted into two equations using $u(t) = \dot{x}$. Considering $y_0 = x$ and $y_1 = u$:

$$\begin{cases} \dot{y_0} = y_1 \\ \dot{y_1} = -\omega^2 y_0 \end{cases} \tag{10.8}$$

10.2.1 Explicit method

1. In a module file, implement the Forward Euler method which arguments will be the initial solution y_0, the initial time t_0, the final time t_{end}, the time step Δt, and returns two vectors: the solution and time arrays.

2. In a main program file, define the initial solution with $x_0 = 1$ and $\dot{x}_0 = 0$, and the parameter $\omega = 4$. Call the module function and plot the results.

3. Check the solution behaviour while changing the time step.

4. The purpose is to get the best estimate of $x(t = 10)$. Evaluate the global error made by the Forward Euler method as a function of the time step Δt.

10.2.2 Implicit methods

1. In your module file for ODE solvers, implement the Backward Euler and Trapezoidal methods.

2. For a given time step $\Delta t = 0.1\ s$, compare on the same plot the time evolution of the solution obtained by every solvers.

3. Compare the errors made by each scheme as a function of the time step, and comment on the stability limits of each method.

10.2.3 High order explicit methods

Here, the objective is to investigate the accuracy and the stability of high order explicit methods. The method of interest are the family of Runge-Kutta methods (RK2, RK3, RK4), the Leapfrog method, and the Adams-Bashforth method.

1. Find the formulation of these schemes on internet and implement them.

2. Comment on the accuracy and the stability limits.

10.3 Evolution of a population

The evolution of a population can be written as:

$$\dot{P} = -P(1 - P) \tag{10.9}$$

where P is the population normalized by the maximum population. The analytical solution of this equation is:

$$P(t) = \frac{1}{1 - Ae^t} \tag{10.10}$$

where $A = 1 - 1/P_0$ and $P_0 = P(T = 0)$. Now you will solve this equation. As the equation is non-linear, we can either use Backward Euler and Trapezoidal methods by inverting a non-linear system or use the linearized version of theses schemes for which the Jacobian $\frac{\partial F}{\partial Y}$ is required as an argument.

1. Solve the equation using Forward Euler method. Analyse the stability of the method for the given equation.
2. Implement the full Backward Euler and Trapezoidal schemes.
3. Implement the linearized versions of theses two schemes (see Appendix).
4. Compare the time evolution of this equation obtained by the five solvers you have implemented.

Appendix

Non-linear solvers

We consider ODE of the general form:

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = F(t, Y(t)) \tag{10.11}$$

The Forward Euler's method is written in general form:

$$Y_{n+1} = Y_n + \Delta t F(t_n, Y_n)) \tag{10.12}$$

The Forward Euler's method is written in general form:

$$Y_{n+1} = Y_n + \Delta t F(t_{n+1}, Y_{n+1}) \tag{10.13}$$

while the trapezoidal scheme is written:

$$Y_{n+1} = Y_n + \Delta t \frac{F(t_n, Y_n) + F(t_{n+1}, Y_{n+1})}{2} \tag{10.14}$$

This two schemes end up in a system to be solved in order to find Y_{n+1}. If the RHS is linear in Y, it can be easily solved using *solve* in *numpy.linalg*. In the non-linear case, It can be done using non-linear solvers such as *fsolve* in *scipy.optimize*., a linearisation can be used:

$$F(t_{n+1}, Y_{n+1}) = F(t_{n+1}, Y_n) + \frac{\partial F}{\partial Y}(Y_{n+1} - Y_n) \tag{10.15}$$

where $\frac{\partial F}{\partial Y}$ is the Jacobian of F with respect to the solution vector Y. This linearisation is obviously exact when the RHS is a linear function of Y. Using this linearisation, the two schemes are then rewritten. The Linearised Forward Euler's method is written:

$$\left(\mathbb{1} - \Delta t \frac{\partial F}{\partial Y}\right) Y_{n+1} = \left(\mathbb{1} - \Delta t \frac{\partial F}{\partial Y}\right) Y_n + \Delta t F(t_{n+1}, Y_n)) \tag{10.16}$$

while the linearized trapezoidal scheme is written:

$$\left(\mathbb{1} - \frac{\Delta t}{2} \frac{\partial F}{\partial Y}\right) Y_{n+1} = \left(\mathbb{1} - \frac{\Delta t}{2} \frac{\partial F}{\partial Y}\right) Y_n + \Delta t \frac{F(t_n, Y_n) + F(t_{n+1}, Y_n)}{2} \tag{10.17}$$

Solving linear and non-linear systems: solve and fsolve

- To solve linear system of equations, the function *solve* in *numpy.linalg* can be used. The syntax is $x = \text{solve}(A, b)$ for solving $Ax = b$, where A is the $n \times n$ array and b a vector of size n.
- To solve non-linear system of equations, the function *fsolve* in *scipy.optimize* can be used. The syntax is $x = \text{fsolve}(G, x_0, args = (a, b, c))$ for solving $G(x, a, b, c) = 0$ with initial guess $x = x_0$.

Chapter 11

Elliptic equations

The goal of the present workshop is to solve Elliptic equations of the form:

$$\nabla^2 \phi = f(\mathbf{x}) \tag{11.1}$$

with various boundary conditions (Dirichlet, Neumann or mixed) as typically met when solving the steady heat equation. This equation is known as the Poisson equation if the right hand side (RHS) is not zero and as the Laplace's equation if the RHS is zero.

11.1 Poisson's equation

In this first problem, we will solve the following Poisson's equation on $\Omega = [-1, 1]^2$ with Dirichlet boundary conditions on $\partial\Omega$:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f(x, y) = 2(x^2 + y^2 - 2) \qquad \text{if } (x, y) \notin \partial\Omega \tag{11.2}$$

$$\phi(x, y) = 0 \qquad \text{if } (x, y) \in \partial\Omega \tag{11.3}$$

The space is discretized into $(N_x+1)\times(N_y+1)$ points. The finite difference of the Poisson equation is:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f(x_i, y_j) = f_{i,j} \tag{11.4}$$

where Δx and Δy are the spacing in x and y directions. In the following, we will consider the same discretization in both directions, i.e. $N_x = N_y = N$ and $\Delta x = \Delta y = h$.
A first method is to directly solve this linear system using for instance Gaussian elimination, which should become too expensive as it will be shown in class. This is why we are interested in iterative solvers which aim at an estimation of the solution at a given tolerance. In addition, unlike direct solver, there is no need to rearrange your solution vector for the iterative solver, as the matrix will never be stored. In this first workshop about Elliptic PDEs, the goal is to implement the Jacobi and Gauss-Seidel methods.
First, we have to set up basis functions:

1. In a python code, generate two arrays x and y representing the discretisation of the 2D space Ω. Build a function that evaluate the RHS f using the inputs x and y, and plot $f(x, y)$ in a surface or contour plot.

2. Implement the Jacobi method for a given number of iterations.

3. Implement a function that computes the residual of the Poisson's equation, i.e. $R = Ax - b$. Modify your code such that the exit condition for the Jacobi iterations will be controlled by the error on the equation, i.e. the residual R.

4. Using the function $time.clock()$, measure the time needed to solve the problem with a discretization $N_x = N_y = 50$.

5. Implement the Gauss-Seidel method and compare the execution time to the one of the Jacobi's method for different numbers of discretization points.

11.2 Laplace equation: treatment of boundary conditiions

In this second problem, the objective is to take into account boundary conditions in the case of the Laplace equation, i.e. with a null RHS. The problem of interest is depicted in Fig.11.1. A square plate of size $L = 10cm$ made of aluminium of thermal conductivity $\lambda = 240\ W/m/K$ received heat fluxes $|\varphi| = 2.4\ MW/m^2$ from the left side. All other sides are at fixed temperature $T_{up} = T_{right} = 300K$.

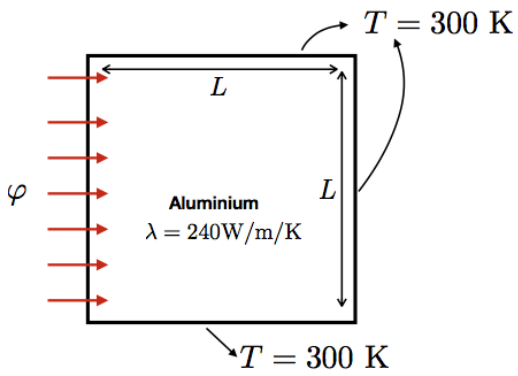


Figure 11.1 : *Aluminium plate with imposed temperature at upper, lower and right sides, and imposed flux from the left side.*

The objective is to analyse the temperature distribution of the plate. The temperature must fulfil the Laplace equation in the inner part of the domain:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{11.5}$$

which is in 2^{nd} order accurate discretized form:

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} = 0 \tag{11.6}$$

Contrary to the previous case, the boundary conditions are not simply set to zero: Dirichlet boundary conditions are imposed on upper, lower and right sides, while Neumann boundary condition is imposed on the left side. The imposed flux condition is:

$$\varphi = -\lambda \nabla T \cdot \mathbf{n} \tag{11.7}$$

where $\mathbf{n}$ is the outward surface normal. To impose the flux, a finite difference formula for the first order derivative is required, which has one constraint: it must be non-centered as it has points on one side only. Finite difference formula that fullfil it are the following:

First order	*Second order*	(11.8)
$\left.\frac{\partial T}{\partial x}\right\|_i = \frac{T_{i+1} - T_i}{\Delta x}$	$\left.\frac{\partial T}{\partial x}\right\|_i = \frac{-3T_i + 4T_{i+1} - T_{i+2}}{2\Delta x}$	(11.9)
$\left.\frac{\partial T}{\partial x}\right\|_i = \frac{T_i - T_{i-1}}{\Delta x}$	$\left.\frac{\partial T}{\partial x}\right\|_i = \frac{3T_i - 4T_{i-1} + T_{i-2}}{2\Delta x}$	(11.10)

The objective is the following: determine what is the maximum temperature?

Appendix

Let us consider the following discretized form:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \tag{11.11}$$

Jacobi method

The Jacobi method is based on the splitting of A into a diagonal matrix and a non-diagonal matrix. The final algorithm is the following:

$$\phi_{i,j}^{(k+1)} = \frac{1}{\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}} \left(\frac{\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k)}}{\Delta x^2} + \frac{\phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k)}}{\Delta y^2} - f_{i,j} \right) \tag{11.12}$$

The iterative process starts with an estimate solution $\phi_{i,j}^{(0)}$ up to the user.

Gauss-Siedel method

In the Gauss-Siedel method, the only difference is that the iterations use the results of the previous calculations in the iteration:

$$\phi_{i,j}^{(k+1)} = \frac{1}{\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}} \left(\frac{\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k+1)}}{\Delta x^2} + \frac{\phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k+1)}}{\Delta y^2} - f_{i,j} \right) \tag{11.13}$$

This method has the advantage of not requiring to store the old and new solutions during each iteration.
The goal of the present workshop is to solve elliptic partial differential equations of the form:

$$\nabla^2 \phi = f(\mathbf{x}) \tag{11.14}$$

with more advanced iterative methods than the basic Jacobi and Gauss-Seidel methods. The multi-block problem is left as homework.

11.3 Poisson's equation with SOR and conjugate gradient methods

Let's consider the finite difference formulation of the Poisson equation seen in Workshop 3:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f(x_i, y_j) = f_{i,j} \tag{11.15}$$

where Δx and Δy are the spacing in x and y directions.The objective is to implement two methods: the SOR method that is a overrelaxed modification of the Gauss-Seidel method, and the Conjugate Gradient method (a Krylov projection method for symmetric systems).

1. First, modify the Gauss-Seidel Algorithm from Workshop 3 to implement the SOR method.

2. Evaluate the influence of the parameter ω on the convergence rate of the method. Find the best choice for you problem and compare SOR to Jacobi and Gauss-Siedel methods.

3. Following the Appendix, implement the Conjugate Gradient method. To simplify the implementation, you will have to generate a function that compute the Laplacian of a given 2D scalar field in a matrix-free manner.

4. Compare the Conjugate Gradient method to SOR, Jacobi and Gauss-Siedel, and comment on the convergence rate.

11.4 Multi-block resolution

Here, a domain decomposition with a multi-block structure mesh is illustrated with the study the cooling of the Vulcain engine, main stage engine of the Arian V space launcher. The hot exhaust gases in the nozzle downstream the engine apply incredible heat loads on the structure which must be cooled (in fact, by using the cryogenic hydrogen fed into the combustion chamber) to sustain the tremendous heat flux. A cross-section of the nozzle geometry and its cooling channels is represented in Figure 11.2a. Instead of solving this complex problem, making use of the symmetry of the problem helps to restrict to the geometry in Figure 11.2b, with the represented boundary conditions. To handle such a geometry, we decomposed the problem into structured blocks that will be solve iteratively. Each is connected to each other by a continuity condition. The sole differences between each block are the boundary condition. The objective is to compute the temperature distribution in all blocks, which is done using the Laplace equation with the appropriate boundary conditions.

- Start by dealing with two blocks. For each iteration in the global iterative method, the temperature fields in the first block and then the second block are updated sequently with the solver of your choice (Jacobi, Gauss-Seidel, SOR...). The interface shared by the two blocks requires a specific attention: It is yet an undefined boundary condition for each part of the domain. The most simple choice is to set a Dirichlet boundary condition for one block (fixing the temperature values from other block solution) and a Neumann boundary condition for the other block (fixing the temperature derivative from the former block solution).

- Discretizating the geometry in Figure 11.2b, compute the maximum temperature in the domain using a multi-block decomposition. The thermal conductivity of the copper is 366 $W.m^{-1}.K^{-1}$.

Appendix

Let us consider the following discretized form:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \tag{11.16}$$

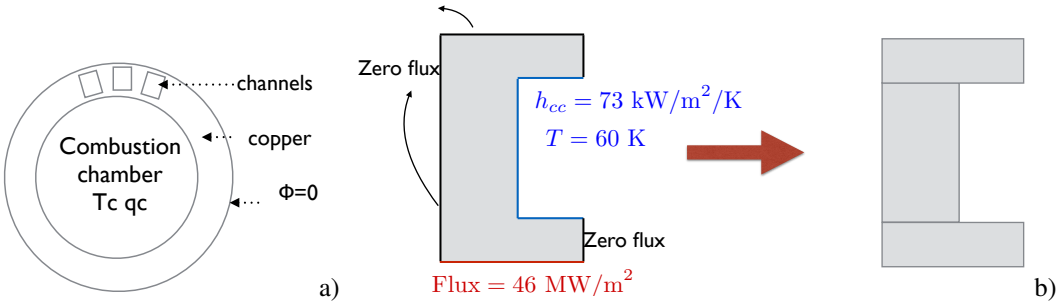


Figure 11.2 : *Cross-section of the Vulcain nozzle: The nozzle made of copper is cooled through channels (full (a) and simplified (b) representations).*

Jacobi method

The Jacobi method is based on the splitting of A into a diagonal matrix and a non-diagonal matrix. The final algorithm is the following:

$$\phi_{i,j}^{(k+1)} = \frac{1}{\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}} \left(\frac{\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k)}}{\Delta x^2} + \frac{\phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k)}}{\Delta y^2} - f_{i,j} \right) \tag{11.17}$$

The iterative process starts with an estimate solution $\phi_{i,j}^{(0)}$ up to the user.

Gauss-Siedel method

In the Gauss-Siedel method, the only difference is that the iterations use the results of the previous calculations in the iteration:

$$\phi_{i,j}^{(k+1)} = \frac{1}{\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}} \left(\frac{\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k+1)}}{\Delta x^2} + \frac{\phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k+1)}}{\Delta y^2} - f_{i,j} \right) \tag{11.18}$$

This method has the advantage of not requiring to store the old and new solutions during each iteration.

Succesive OverRelaxation (SOR) method

The SOR method is an overrelaxation of the Gauss-Seidel method:

$$\phi_{i,j}^{(k+1)} = (1-\omega)\phi_{i,j}^{(k)} + \frac{\omega}{\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}} \left(\frac{\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k+1)}}{\Delta x^2} + \frac{\phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k+1)}}{\Delta y^2} - f_{i,j} \right) \tag{11.19}$$

where $\omega \in]0, 2[$.

Conjugate gradient method

Let us first define the residual of iteration k:

$$r_{i,j}^{(k)} = f_{i,j} - \frac{\phi_{i+1,j}^{(k)} - 2\phi_{i,j}^{(k)} + \phi_{i-1,j}^{(k)}}{\Delta x^2} + \frac{\phi_{i,j+1}^{(k)} - 2\phi_{i,j}^{(k)} + \phi_{i,j-1}^{(k)}}{\Delta y^2} \tag{11.20}$$

We also define the application of the RHS to a given vector $p_{i,j}^{(k)}$:

$$(A \cdot p)_{i,j}^{(k)} = \frac{p_{i+1,j}^{(k)} - 2p_{i,j}^{(k)} + p_{i-1,j}^{(k)}}{\Delta x^2} + \frac{p_{i,j+1}^{(k)} - 2p_{i,j}^{(k)} + p_{i,j-1}^{(k)}}{\Delta y^2} \tag{11.21}$$

Let us consider $p_{i,j}^{(0)} = r_{i,j}^{(0)}$. The iterations of Conjugate Gradient method are then the following:

$$\alpha^{(k)} = \frac{\sum_{i,j} r_{i,j}^{(k)} r_{i,j}^{(k)}}{\sum_{i,j} p_{i,j}^{(k)} (A \cdot p)_{i,j}^{(k)}} \tag{11.22}$$

$$\phi_{i,j}^{(k+1)} = \phi_{i,j}^{(k)} + \alpha^{(k)} p_{i,j}^{(k)} \tag{11.23}$$

$$r_{i,j}^{(k+1)} = r_{i,j}^{(k)} - \alpha^{(k)} (A \cdot p)_{i,j}^{(k)} \tag{11.24}$$

$$\beta^{(k)} = \frac{\sum_{i,j} r_{i,j}^{(k+1)} r_{i,j}^{(k+1)}}{\sum_{i,j} r_{i,j}^{(k)} r_{i,j}^{(k)}} \tag{11.25}$$

$$p_{i,j}^{(k+1)} = r_{i,j}^{(k+1)} + \beta^{(k)} p_{i,j}^{(k)} \tag{11.26}$$

Chapter 12

Hyperbolic and parabolic equations: explicit methods

The goal of the present workshop is to first solve simple parabolic and hyperbolic problems where analytical solutions are known: 1D unsteady heat equation and 1D advection. The stability criteria are illustrated with these examples using different numerical methods. The last problem is left as homework and considers a parabolic problem of paramount physical importance: the thin boundary layer in the presence of pressure gradient.

12.1 1D Unsteady diffusion

Let's consider the unsteady heat equation for the scaled temperature profile $\hat{T} = \frac{T-T_w}{T_c^0-T_w}$:

$$\frac{\partial \hat{T}}{\partial t} = a\frac{\partial^2 \hat{T}}{\partial x^2},$$

with the corresponding boundary conditions

$$\begin{cases} \hat{T}(x=0,t) & = 0 \\ \hat{T}(x=L,t) & = 0 \end{cases},$$

and initial profile,

$$\hat{T}(x,0) = \sin\left(\frac{\pi x}{L}\right).$$

The length of the domain is $L = 1$ cm and the thermal diffusivity is $a = 10^{-4}$ m^2/s. The analytical solution of the problem is known and is given by:

$$\hat{T}(x,t) = \sin\left(\frac{\pi x}{L}\right) e^{-\frac{\pi^2}{L^2}at}$$

- Solve the discrete parabolic PDE using forward Euler for time integration and the centered-2$^{\text{nd}}$-order difference formula for space discretization.
- Check the importance of setting the time step by selecting the value of the Fourier number for the sake of numerical stability.
- Compare the numerical solution $\hat{T}(x = L/2, t)$ with the analytical solution.
- Multiply the number of spatial points by 10, what is the critical issue of using explicit methods to tackle such a parabolic PDE?

12.2 1D advection

Advection of a profile with constant velocity c is described by the following PDE:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0.$$

The initial profile is set as

$$u(x, t = 0) = u_0(x) = \exp\left(-\frac{(x - x_0)^2}{\delta^2}\right),$$

with $x_0 = 1$ m and $\delta = 0.2$ m. The domain length is $L = 12$ m and the advection velocity is set to $c = 10$ m/s. Since the solution is conveyed from left to right, a boundary condition is required only on the left side of the domain, a particularity of the hyperbolic nature of the PDE. This boundary condition is set as

$$u(x = 0, t) = 0.$$

The analytical solution is a translation of the initial profile:

$$u(x, t) = u_0(x - ct).$$

- Check the conditional stability of the discretized problem with forward Euler for time integration and 1st-order upwind for space discretization.

- Implement a second-order centered space discretization instead. What do you observe?
- Implement the second-order centered space discretization along with the leap-frog scheme.
- Implement the second-order centered space discretization along with the third-order Runge-Kutta method RK3.
- Choose any combination and test it. Is the stability behavior consistent with the theory?
- For all tested methods, plot on the same figure the different solutions and the analytical one after half a residence time in the computational domain, *i.e.* $t = L/(2c)$. What do you think of the quality of your results? Solutions exhibit strong numerical diffusion and/or dispersion.

12.3 Thin boundary layer in the presence of pressure gradient

Boundary layers determine many flow characteristics in fluid mechanics such as head losses in pipes, drag on moving objects, stall of a plane, viscous heating around supersonic vehicles ... Boundary layers are typically very thin (if they remain attached) and are characterized by the presence of viscous effects that cannot be neglected. For laminar thin boundary layers, the bidimensional flow over a flat plate is described by the velocity components (u, v), and is governed by (i) the continuity equation (a.k.a. local balance equation of mass) which is written for incompressible flows as

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{12.1}$$

and (ii) by the balance of momentum along the x-direction:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \nu\frac{\partial^2 u}{\partial y^2} - \frac{1}{\rho}\frac{\mathrm{d}p}{\mathrm{d}x}, \tag{12.2}$$

where ρ is the density, ν the kinematic viscosity and $\frac{\mathrm{d}p}{\mathrm{d}x}$ is the pressure gradient set by the outer flow onto the boundary layer. For water, $\rho = 1000$ kg/m^3 and $\nu = 10^{-6}$ m^2/s. The general steady Navier-Stokes equations would have yielded a non-linear elliptic set of PDEs, which is cumbersome to solve. Thanks to the thin layer approximation, the obtained equations 1 and 2 are parabolic and can be solved step by step in the x-direction starting from an incoming profile at $x = 0$.

12.3.1 No pressure gradient

Many theoretical results are available when the pressure gradient is null or is negligible. Indeed, the velocity profiles correspond to the solution of the Blasius solution, which is obtained by looking for auto-similar solutions of equations 1 and 2. The method is here different: Equations 1 and 2 are solved directly without further modifications.

The computational domain is discretized with $N_y + 1$ points in the y-direction between $y = 0$ (the wall location) and $y = H$ the outer boundary condition where the velocity can be assumed uniform. The corresponding y-grid spacing is denoted by Δy. Adopting a step-by-step approach, $N_x + 1$ iterations are carried out in the x-direction starting from the inlet $x = 0$ up to the last computed profile corresponding to the domain x-extent L. The corresponding x-grid spacing is denoted by Δx.

The discrete velocity fields $u_{i,j}$ and $v_{i,j}$ require "initial" profiles and boundary conditions as for any parabolic PDE. The "initial"profile is the inlet velocity profile and is set as a uniform flow:

$$u(x = 0, y) = u_{0,j} = U_{in}$$

$$v(x = 0) = v_{0,j} = 0.$$

The boundary conditions are

$$u(x, y = 0) = u_{i,0} = 0$$

$$v(x, y = 0) = v_{i,0} = 0$$

$$u(x, y = H) = u_{i,N} = U_{ext}(x),$$

where $U_{ext}(x)$ is the external flow profile which can change due to the pressure gradient. With $\frac{\mathrm{d}p}{\mathrm{d}x} = 0$, $U_{ext}(x) = U_{in}$.

- Consider equation 2 to determine a formula to evaluate $u_{i+1,j}$ from known data at $x = x_i$ (examples: $u_{i,j}$, $u_{i,j+1}$, $u_{i,j-1}$, $v_{i,j}$, $v_{i,j-1}$...) using a Forward Euler integration. Define the dimensionless numbers equivalent to the Courant and Fourier numbers.
- Use then equation 1 to obtain a formula to determine $v_{i+1,j}$ from the computed values $u_{i+1,j}$ and the data at $x = x_i$.
- Implement the derived numerical resolution and plot different computed profiles $u(x, y)$ at different axial positions. Be careful with the stability by controlling Δx and monitoring the effective Courant and Fourier numbers properly.
- The profiles show that the boundary layer thickens. Estimate the boundary layer thickness $\delta_{99}(x)$ defined as the point where $u(x, y = \delta_{99}(x)) = 0.99U_{ext}(x)$. The theoretical behavior $\delta_{99}(x) \approx \sqrt{x}$ is expected away from the leading edge (x=0).

12.3.2 Effects of pressure gradient

The velocity profiles in the boundary layer are greatly impacted by the presence of a pressure gradient in the external flow. The boundary layer can then thicken faster or can get thinner. The flow can even separate from the wall, yielding a large recirculation bubble with negative values of the streamwise velocity u. In this last case, equation 2 breaks down and cannot be solved beyond the separation point.

Accounting for the pressure gradient modifies the outer boundary condition at $y = H$,

$$u_{i,N} = U_{ext}(x),$$

since $U_{ext}(x)$ changes accounting to the Bernoulli theorem:

$$\frac{\mathrm{d}U}{\mathrm{d}x} = -\frac{1}{\rho U(x)}\frac{\mathrm{d}p}{\mathrm{d}x}.$$

- Define a function that returns $\frac{\mathrm{d}p}{\mathrm{d}x}$ which is null in the first half of the computed domain (between $x = 0$ and $x = L/2$) and non-null yet constant in the second half (between $x = L/2$ and $x = L$).
- Add the effect of $\frac{\mathrm{d}p}{\mathrm{d}x}$ and solve the case of a negative pressure gradient. Why is this called a favorable pressure gradient ?
- Solve the case of a positive pressure gradient. Why is this called a unfavorable pressure gradient ?

Chapter 13

Hyperbolic and parabolic equations: numerical errors

The present workshop focuses on the linear advection equation:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0 \tag{13.1}$$

with periodic boundary conditions over the domain $x \in [0, 1]$. The analytical solution of this equation is $u(t, x) = u(t = 0, x - at)$, such that it presents no diffusion (maxima and minima are preserved) nor dispersion (all wave numbers travel at the same speed). It is thus an ideal case for evaluating numerical errors as well as diffusive and dispersive properties of numerical schemes.

13.1 Order of convergence of numerical schemes

In this first problem, the objective is to evaluate the order of accuracy of different schemes by comparing the analytical solution to the numerical one. The initial condition is a Gaussian distribution:

$$u(t = 0, x) = \exp\left(-\frac{(x - 0.5)^2}{0.1^2}\right) \tag{13.2}$$

- Implement the upwind scheme, Leap-Frog, Lax-Wendroff and Lax-Friedrichs schemes.
- Observe the different behaviors of each scheme, and identify dispersive and diffusive effects.
- Identify the impact of the CFL number on the numerical solution.
- evaluate the error of each scheme as a function of the space discretization for different CFL numbers.
- Quantify the order of accuracy of each scheme.

13.2 Analysis of numerical dispersion and diffusion

In this second problem, the goal is to observe the evolution of a pure sinusoidal perturbation:

$$u(t = 0, x) = \sin(\omega x) \tag{13.3}$$

- Implement this new initial condition.

- For a given wave number, evaluate the numerical diffusion of all schemes. You have to find an adequate metric.
- Then, evaluate the numerical dispersion of all schemes. You also have to define an adequate metric.
- evaluate theses errors for several wave numbers and several CFL numbers.

Chapter 14

Implicit methods for parabolic equations

14.1 Implicit method applied to 1D Unsteady diffusion

Let's consider the unsteady heat equation seen in Workshop #05:

$$\frac{\partial \hat{T}}{\partial t} = D \frac{\partial^2 \hat{T}}{\partial x^2},$$

with the corresponding boundary conditions

$$\begin{cases} \hat{T}(x=0,t) & = 0 \\ \hat{T}(x=L,t) & = 0 \end{cases},$$

and initial profile,

$$\hat{T}(x,0) = \sin\left(\frac{\pi x}{L}\right).$$

The length of the domain is $L = 1$ cm and the thermal diffusivity is $D = 10^{-4}$ m^2/s. The corresponding analytical solution of the problem is :

$$\hat{T}(x,t) = \sin\left(\frac{\pi x}{L}\right) e^{-\frac{\pi^2}{L^2} Dt}.$$

An implementation of the Thomas algorithm to solve tridiagonal linear systems is given in the module file TDMA.py on the course website.

- Solve the discrete parabolic PDE using the implicit Crank-Nicolson method and the centered-2$^{\text{nd}}$-order difference formula for space discretization.
- Verify the unconditional stability on the chosen value of the Fourier number.
- Implement also the resolution of the unsteady heat equation with the implicit Backward Euler method.
- Compare the numerical solutions of both methods with the analytical solution on the centerline $\hat{T}(x = L/2, t)$. Outline the effect of numerical accuracy order with different time steps.

14.2 Implicit methods to reach steady state solution

Given the unconditional stability of implicit methods, taking large time steps allows to quickly jump the physical transient (without considering its accurate description) to compute the steady state solution. Doing so, makes the ADI method quite competitive in respect to iterative methods for elliptic PDEs. In fact, the ADI time-stepping can be considered as a preconditioned iterative method.
Let's consider the Poisson equation solved in Workshop #03 solved on $\Omega = [-1, 1]^2$ with Dirichlet boundary conditions on $\partial\Omega$:

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = f(x, y) = 2(x^2 + y^2 - 2) \qquad \text{if } (x, y) \notin \partial\Omega \tag{14.1}$$

$$\phi(x, y) = 0 \qquad \text{if } (x, y) \in \partial\Omega \tag{14.2}$$

This is the steady state solution of the following unsteady equation with the same boundary conditions:

$$\frac{\partial\phi}{\partial\tau} = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} - f(x, y) \tag{14.3}$$

This equation is similar to a 2D unsteady heat equation with unity heat diffusivity and a source term. This parabolic PDE is solved with an implicit method to carry out large time steps and get to the final state as soon as possible.

14.2.1 Alternate Direction Implicit method

The ADI formula by Peaceman and Rachford is an approximate factoring of the 2D Crank-Nicolson method: The full linear system resolution is replaced a series of tridiagonal systems which are very cheap to solve. The ADI method then keeps all the advantages of Crank-Nicolson (2^{nd}-order accurate in time, unconditionally stable) without any drawbacks.

- Implement the ADI method to solve Eq. 14.3.
- Check the unconditional stability of the method by increasing the retained Fourier number.
- Compare the ADI method efficiency to compute the steady state solution to elliptic iterative methods for different system sizes.

14.2.2 2D Crank-Nicolson

Combined with centered-2^{nd}-order spatial discretization, the Crank-Nicolson yields a large linear system seen in class to be solved at each time step.

- Implement the time-stepping Crank-Nicolson method to solve Eq. 14.3 by solving the linear system at each step with an iterative method (Jacob, Gauss-Seidel, SOR or ...).
- Carefully design the stopping criterion of the iterative method applied at each step.
- Check the unconditional stability of the method by increasing the retained Fourier number.
- Compare its wall time clock to the ADI implementation.

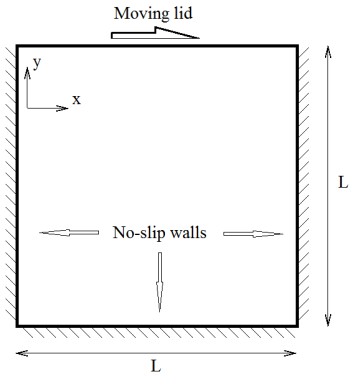
This implementation of Crank-Nicolson is not very efficient compared to ADI method to compute the accurate unsteady behavior of the PDE. Besides, it is definitely useless as an efficient method based on accelerated parabolic PDE to compute the steady state which can be determined by a single application of the iterative method on the steady elliptic PDE.

Chapter 15

Navier-Stokes equations

The lid driven cavity problem

The objective of the present workshop is to solve the incompressible flow inside a square lid-driven cavity of size L. In this problem, the boundary of the domain are walls with zero normal and tangential velocities, except for the upper wall, which has a fixed velocity U. Due to the fluid viscosity, the upper wall will induce the generation of a vortex, with the potential formation of corner recirculation zones, depending on the flow regime.



The flow condition is uniquely determined by the Reynolds number $Re = \frac{UL}{\nu}$. In the following, we will investigate the flow for $Re = 100$.

- Implement the explicit method given in class
  - Forward Euler for time marching
  - Projection method for pressure-velocity coupling
  - Second order centered finite difference for space derivatives.
- Investigate the impact of time and space discretizations on the solution, as well as the impact of the convergence of the Poisson solver.
- Plot the velocity profile at the steady state for the vertical line at $x = 0.5L$ and the horizontal line at $y = 0.5L$.

Appendix A

Eigenvalues of Tridiagonal Matrices

Contents

A.1 Recurrence Relation for the Determinant of a Tridiagonal Matrix

Let T be a $(n-1) \times (n-1)$ tridiagonal matrix:

$$M = \begin{pmatrix} b & c & & 0 \\ a & \ddots & \ddots & \\ & \ddots & \ddots & c \\ 0 & & a & b \end{pmatrix} = B[a, b, c]. \tag{A.1}$$

Let us introduce the notation $D_{(N-1)} = \det(T)$
Developing the determinant yields:

$$D_{(N-1)} = \begin{vmatrix} b & c & & 0 \\ a & \ddots & \ddots & \\ & \ddots & \ddots & c \\ 0 & & a & b \end{vmatrix} = b \underbrace{\begin{vmatrix} b & c & & 0 \\ a & \ddots & \ddots & \\ & \ddots & \ddots & c \\ 0 & & a & b \end{vmatrix}}_{D_{(N-2)}} - c \underbrace{\begin{vmatrix} a & c & 0 & & 0 \\ 0 & b & \ddots & & \\ 0 & a & \ddots & \ddots & \\ & & \ddots & \ddots & c \\ 0 & & & a & b \end{vmatrix}}_{aD_{(N-3)}} \tag{A.2}$$

Eventually, it writes:

$$D_{(N-1)} = bD_{(N-2)} - acD_{(N-3)} \tag{A.3}$$

A.2 Solving the Corresponding Characteristic Equation

Writing the characteristic equation:

$$X^2 - bX + ac = 0 \tag{A.4}$$

One finds:

$$X_1 = \frac{b + \sqrt{b^2 - 4ac}}{2}; X_1 = \frac{b - \sqrt{b^2 - 4ac}}{2}. \tag{A.5}$$

Introducing (r, θ) such that $r = \sqrt{ac}$ and $2r\cos\theta = b$, one can then write

$$\Delta = 4r^2\cos^2\theta - 4r^2 = -4r^2\sin\theta, \tag{A.6}$$

and

$$X_1 = re^{i\theta}; X_2 = re^{-i\theta} \tag{A.7}$$

Hence,

$$D_{(N-1)} = \alpha X_1^{N-1} + \beta X_2^{N-1}, \tag{A.8}$$

with

$$D_{(1)} = |b| = b, \tag{A.9}$$

then $\alpha X_1 \neq \beta X_2$, and:

$$\beta X_2 = 2r\cos\theta. \tag{A.10}$$

$$D_{(2)} = \begin{vmatrix} b & c \\ a & b \end{vmatrix} = b^2 - ac \tag{A.11}$$

$$\alpha X_1^2 + \beta X_2^2 = 4r^2\cos^2\theta - r^2 \tag{A.12}$$

Writing $(X_1 \cdot (A.10)) - (A.12)$, one gets:

$$\beta\left(X_1X_2 - X_2^2\right) = 2r^2\cos\theta e^{+i\theta} - 4r^2\cos^2\theta + r^2 \tag{A.13}$$

$$\beta = \frac{2\cos qe^{i\theta} - 4\cos^2\theta + 1}{1 - e^{-2i\theta}} = \frac{1 - 2\cos^2\theta + i\sin(2\theta)}{1 - e^{2i\theta}} \tag{A.14}$$

$$\beta = \frac{-e^{-2i\theta}}{1 - e^{-2i\theta}} = \frac{1}{1 - e^{2i\theta}} = -\frac{e^{-i\theta}}{2i\sin\theta} \tag{A.15}$$

$$\alpha = \frac{2r\cos\theta - \beta X_2}{X_1} = \frac{2r\cos\theta + \frac{re^{-2i\theta}}{2i\sin\theta}}{re^{i\theta}} = \frac{2i\sin(2\theta) + e^{-2i\theta}}{2ie^{i\theta}\sin\theta} \tag{A.16}$$

$$\alpha = \frac{\cos(2\theta) + i\sin(2\theta)}{2ie^{i\theta}\sin\theta} = \frac{e^{i\theta}}{2i\sin\theta} \tag{A.17}$$

Eventually,

$$D_{(N-1)} = \frac{e^{i\theta}}{2i\sin\theta}r^{N-1}e^{i(N-1)\theta} - \frac{e^{-i\theta}}{2i\sin\theta}r^{N-1}e^{-i(N-1)\theta} \tag{A.18}$$

$$= \frac{r^{N-1}}{2i\sin\theta}\left(e^{iN\theta} - e^{-iN\theta}\right) \tag{A.19}$$

$$D_{(N-1)} = r^{N-1}\frac{\sin(N\theta)}{\sin\theta} \tag{A.20}$$

A.3 Determination of the Eigenvalues of the Tridiagonal Matrix

In order to compute the eigenvalues of $B[a, b, c]$, one has to solve the polynomial equation:

$$\begin{vmatrix} (\lambda - b) & -c & & 0 \\ -a & \ddots & \ddots & \\ & \ddots & \ddots & -c \\ 0 & & -a & (\lambda - b) \end{vmatrix} = 0 \tag{A.21}$$

This corresponds to solving $D_{(N-1)} = 0$ for $B[-a, \lambda - b, c]$:

$$D_{(N-1)} = \frac{r^{N-1}}{\sin\theta} sin(N\theta) = 0. \tag{A.22}$$

for

$$\begin{cases} r = \sqrt{(-a)(-c)} = \sqrt{ac} \\ 2r\cos\theta = \lambda - b \end{cases} \tag{A.23}$$

From

$$\sin(N\theta) = 0 \text{ (for } \theta \neq 0) \tag{A.24}$$

it comes that

$$N\theta = j\pi, \text{ for } j = 1, ..., N-1 \tag{A.25}$$

Then

$$\theta = \frac{j\pi}{N} \tag{A.26}$$

Remark

what would $\theta = 0$ yield ?

Eventually,

$$\boxed{\lambda_j = b + 2\sqrt{ac}\cos\left(\frac{j\pi}{N}\right)} \text{ for } 1 \leq j \leq N-1 \tag{A.27}$$

Bibliography

[1] S. Candel. *Mécanique des fluides, Cours*. Dunod, 2005. second edition.

[2] L. Prandtl. Grenzschicht theorie. In *Verhandlungen des dritten internationalen Mathematiker-Kongresses*, page 484, Heidelberg, 1904. English trans. in Early Developments of Modern Aerodynamics, J. A. K. Ack- royd, B. P. Axcell, A. I. Ruban, eds., Butterworth-Heinemann, Oxford, UK (2001), p. 77.

[3] F. Richecoeur. *Aérodynamique : équations générales, écoulements laminaires et turbulents autour d'un profil, couche limite*. Ellipses, 2013.

[4] H. Schlichting and K. Gersten. *Boundary Layer Theory*. Springer, 2000. 8th revised and enlarged edition.