Please note: this is a reviewed preprint, excerpted from a compiled volume, that may subject to style modifications by the Editors and the Publisher

Preliminary citing:
Storti, M.A. and Ferreri, J.C., "Numerical Methods in Nuclear Thermal-Hydraulics, Chapter 20", in: "Handbook on Thermal-Hydraulics in WCNR", F.S. D'Auria and Y. Hassan, Eds., Elsevier, to be published, 2022

CHAPTER 20

NUMERICAL METHODS IN NUCLEAR THERMAL-HYDRAULICS

Mario Alberto Storti
CIMEC, Predio CONICET "Dr. Alberto Cassano"
Colectora. Ruta Nac. 168 Km 0, Paraje El Pozo, 3000 Santa Fe, Argentina
mario.storti@gmail.com

Juan Carlos Ferreri
Academia Nacional de Ciencias de Buenos Aires, Av. Alvear 1711, 3º, 1014 CABA, Argentina
jcferreri@ciencias.org.ar jcferreri@gmail.com

Abstract

Numerical methods utilized for the solution of governing fluid flow equations are considered, with emphasis on the influence of the adopted schemes in the solution and the computational costs of the approximations. Consequently, the subject as presented is a mix between algorithmic, modeling, and algebraic aspects, keeping the usual separation in the type of partial differential equations. A detailed view of the theory manuals of the major Thermal Hydraulic codes permits the verification of the coherence of this approach. The verification of the validity of the solutions and the automatic sensitivity analysis of codes is also analyzed.

This Nomenclature is aimed at listing the main symbols and abbreviations used. All variables are defined when they first appear in the text.

NOMENCLATURE

ACRONYMS

1D, 2D, 3D	One, two and three dimensional space dimensions
CFD	Computational Fluid Dynamics
MMS	Method of Manufactured Solutions
PDE	Partial differential equations
SYS-TH	System Thermal Hydraulics
TH	Thermal Hydraulics
DA	Discrete Approximation
TE	Truncation Error
IC	Initial Conditions
BC	Boundary Conditions
A	Linear restricted operator
CN	Crank-Nicholson
SETS	Stability Enhancing Two-Step
ADI	Alternating Direction Implicit
SPD	Symmetric Positive Definite (matrix)
CG	Conjugate Gradient

VARIABLES

C	Transport Velocity
C_{OU}	Courant Number
I	Identity Operator
L	Linear Differential operator
N	Non-linear Differential operator
$Nops$	Number of operations
R	Boundary of the integration domain D
u	Unknown theoretical solution
U	Discrete approximation for u
x, t	Space and Time coordinates
P_e	Peclet Number
k	Thermal conductivity
a	Diffusion coefficient
D	Laplacian operator (Equation 20.47)
$\Delta x, \Delta t$	Space and Time step intervals
τ, D	Limit of a regular integration domain in space and time

SUBINDEXES

i	Position of a point in space discretization
h	Discrete version of a differential operator

SUPERINDEXES

n	Position of a point in time discretized domain

Chapter Foreword

This chapter considers a basic description of numerical methods, starting with the discretization of partial differential equations (PDE) relevant to governing equations. Two basic techniques are considered: finite differences and finite volumes. This restricted choice is not arbitrary since they are the most used in Thermal Hydraulics (TH) industrial codes. It is not possible, in the limited space of one chapter, to consider all the variations of numerical methods. Furthermore, it is assumed that the reader has basic skills in the topics, so the focus will be on the implications of the numerical schemes *and* their interaction with the solutions. The scope will be limited to transient approaches in the time-space domain, with the obvious exception of the elliptic equations of the Poisson type. This choice implies leaving apart the phase domain methods, which are typically applied to pure stability analysis or wave propagation analysis. The starting point for the derived approximations will be scalar one-dimensional PDE and different approaches will include explicit and implicit expressions. In the case of multiple space dimensions, implicit methods will imply considering direct matrix algebra and alternating direction implicit methods, especially for parabolic and elliptic cases.

Whenever possible, the discrete procedures derivation will be also based on physical reasoning. Then, the truncation error analysis will be important. This puts into evidence the influence of the Los Alamos school. Despite this, the analysis of truncation error and the formal definition of essential aspects like consistency, stability, and convergence will be discussed. Even when this chapter is not oriented to numerical analysis, some aspects will be considered, following G.I. Marchuk, Marchuk, 1975. Usual terminology, like robustness, accuracy, and some other aspects will be explicitly addressed. At this point, it may be obvious that the numerical regularization via added numerical viscosity will be the chosen path whenever necessary. In this case, the question of the accuracy of the solution concerning mesh size and time step will be addressed and exemplified. Other chapters (e.g., Chapter 5 on Governing Equations, Chapter 11 on the Structure of SYS-TH codes and Chapter 22 dealing with numerical aspects connected with Verification) have considered some of the issues just mentioned in different detail, making it possible to restrict the analysis in this Chapter to the numerics of discretization. A basic feature of the numerical approach is the preservation of the conservation of mass and energy. This property is intrinsic to the governing equations and must be respected. This may be considered obvious, but it was not always so in considering the momentum equation. Because of this reason, this aspect will be considered.

Two aspects related to the discretization of PDE are a) the verification of the correctness of the solution obtained using the numerical representation and, b) the automatic computation of sensitivities to the PDE coefficients or the discretization parameters. In the first case, the Method of Manufactured Solutions (or MMS) has been systematized by P. Roache (see also Chapter 22 of the book), even when more elementary procedures are possible. In the second case, automatic differentiation of codes can be used, employing available software. Both cases will be addressed.

The discretization of the governing equations leads to the problem of solving the associated algebraic problem. This implies considering the coupling of dependent variables and the choice of implicitness in the solution procedure. The usual pressure-velocity solution, the coupling of the temperature or enthalpy field, and the concentration or turbulence modeling in the simple case of single-phase TH, lead to large sets of linear algebraic equations. Coupled Heat Conduction increases the number of equations very much. These systems may be solved by iterative or direct methods. The choice depends on the availability of adequate computing resources. Consequently, the discussion of the aspects related to the use of these methods is in order.

20.1 An Introduction to numerical methods: basic concepts on the discretization of partial differential equations

This section aims at considering several aspects of numerical techniques giving the background to the computation of Thermal-Hydraulics as applied in industrial codes. It will be shown that there is plenty of consolidated literature and concepts on numerical methods in the last 50 years and that this is not a recent activity concerning the TH of Nuclear Safety. Many outstanding people contributed to this subject in the last decades. A brief account of early contributions (and contributors) is presented. In this way, Nuclear Safety benefits from well-developed techniques in other areas of Computational Mechanics. Most publications deal with the needs and challenges posed by physics and with clarifying the degree of detail required by Nuclear Safety.

Discrete numerical techniques applications are not a recent activity concerning the Thermal-Hydraulics of all the flow stages of transients in nuclear installations. In the Authors´ view, it was the work at Los Alamos Scientific Laboratory (LANL), which, now more than fifty years ago in those days of “open” exchange of scientific information, opened the way to “widespread” numerical modeling in Fluid Dynamics. See, e.g., LANSCE, (1981), summarizing work done up to 1981 at LANL. Fresh persons may be tempted to consider that the state of the art fifty years ago was rather crude (instead of this, computers were not powerful enough), however, contributions like the one by Roache, (1972), and (1998), giving an important and comprehensive review and recommendations on numerical techniques and CFD by 1972, may clear this question. A few people that contributed to the development from the numerical analysis side will be mentioned in what follows because the list is too long. The names of R. Courant, K.O. Friedrichs, D. Hilbert, J. von Neumann, P.D. Lax, R.D. Richtmyer, G.I. Marchuk (in Russia), and W.F. Ames must also be mentioned. It must be stressed that the Finite Element Method was starting to be applied to solve the Navier-Stokes equations. The Drift-Flux theory of two-phase flow, as developed by Zuber and Findlay in 1956 and by M. Ishii in his 1975 book (see the general list of references of the book), was the two-phase fluid model that allowed generating results of Nuclear Safety significance through its implementation in the TRAC code and other hydrodynamics codes at LANL.

Before continuing, we introduce hereafter a suitable conceptual excerpt from Scannapieco and Harlow, 1995. "*In as much as we can simulate reality, we can use the computer to make predictions about what will occur in a certain set of circumstances ... . Finite-difference techniques can create an artificial laboratory for examining situations which would be impossible to observe otherwise, but we must always remain critical of our results ... .Finite differencing can be an extremely powerful tool, but only when it is firmly set in a basis of physical meaning. For a finite-difference code to be successful, we must start from the beginning, dealing with simple cases and examining our logic each step of the way*". The importance of this assertion must be emphasized today, given the advancement of CFD applications and the relatively easy access to sophisticated CFD tools. The philosophy of merging physical intuition and numerical properties has been always inspiring for CFD practitioners and code developers. This tradition persists, and an example may be found in Knoll et al., 2005.

It seems that there are at least two decades of delay in the *detailed* computation of fluid flow considering other problems in Applied Mechanics. The reasons for this delay, inter alia, maybe:

a) Scaling laws usually "force" to quasi 1D integral test facilities representations of real-life installations, with pre-established flow patterns.
b) A huge effort was focused on the development of a representative physical data-base (amenable to 1D analysis) and separate effects on the other side (like plume analysis, non-symmetric flow distribution, particular aspects of reactor components behavior, etc.) affordable through detailed computational techniques.
c) Code assessment for safety analysis also imposed and still imposes a huge effort for 1D.
d) Time scales to solve problems realistically (as a compromise between cell Courant number limitation for fast transients using semi-implicit methods and time inaccuracies, like damping, for implicit methods impose a large number of time steps to span long time transients (e.g., in SBLOCA); and
e) The Ill-posedness of the governing equations "precluded" the search for detailed convergence of solutions, leading to coarse grid computations and stabilization of flow solvers solutions by numerical means.

There is another aspect that is somewhat redundant with the previous paragraph. As stated by Ferreri and Ambrosini, 2002, in dealing with natural circulation problems, "*Sometimes, scaling leads to the adoption of the 1-D approximation; this may, in turn, hide important aspects of the system physics. A simple example of this situation consists in keeping the height of the system unchanged to get the same buoyancy. Then, if the system is scaled accordingly to the power/volume ratio, the cross-section area of the volume will be reduced; this leads to a much smaller pipe diameter that makes the 1-D representation reasonable, at the cost of eliminating the possibility of fluid internal recirculation ... . A workaround for this situation is providing paths for recirculation, in the form of additional, interconnected components; however, this solution may impose*

the flow pattern in the system and the balance between these aspects is a challenge to any practitioner in natural circulation modeling".

It is important to start with a general formulation of a discrete approximation of an evolution problem, without making differences for the moment on the type of differential equation, because it gives the basis for nearly all discrete formulations of the Navier-Stokes equations. The approach to be followed is the classic one, as may be found in Mitchell and Griffiths, 1980, and Marchuk, 1975. The theory related to operators (linear ones, to be strict) may be found in classic books. It may be mentioned that fluid flow conservation equations derive from integral conservation principles. It is after analytical manipulation that differential equations are obtained.

Before continuing it is necessary to introduce some illustration of how the discrete solution at a given point (i, $n+1$) in the integration domain (τ, D) may depend on the surrounding points. The goal is, of course, advancing in time to $t+\Delta t$, noting that the problem is time marching. The most elementary discretization of a regular integration domain is shown in Figure 20.1.

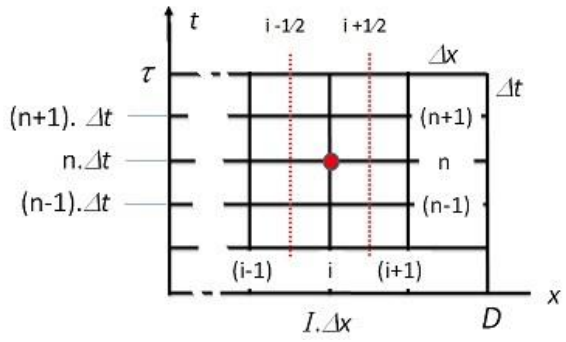

Figure 20.1 – Elementary discretization of an integration domain (τ, D).

Figure 20.2, A-D, shows how the discrete solution at an advanced time point (i, $n+1$) depends on neighboring points for the most common discretizations:

a) explicit, in which the solution depends on values at time level n.Δt;
b) fully implicit in which the solution is advanced simultaneously in all the points at $(n+1).\Delta t$ from n.Δt;
c) the time step centered implicit method in which the solution is advanced simultaneously in all the points at $(n+1).\Delta t$ from $n.\Delta t$ involving all the points at $n.\Delta t$,
d) a 3D time explicit approximation similar to the 1D case.

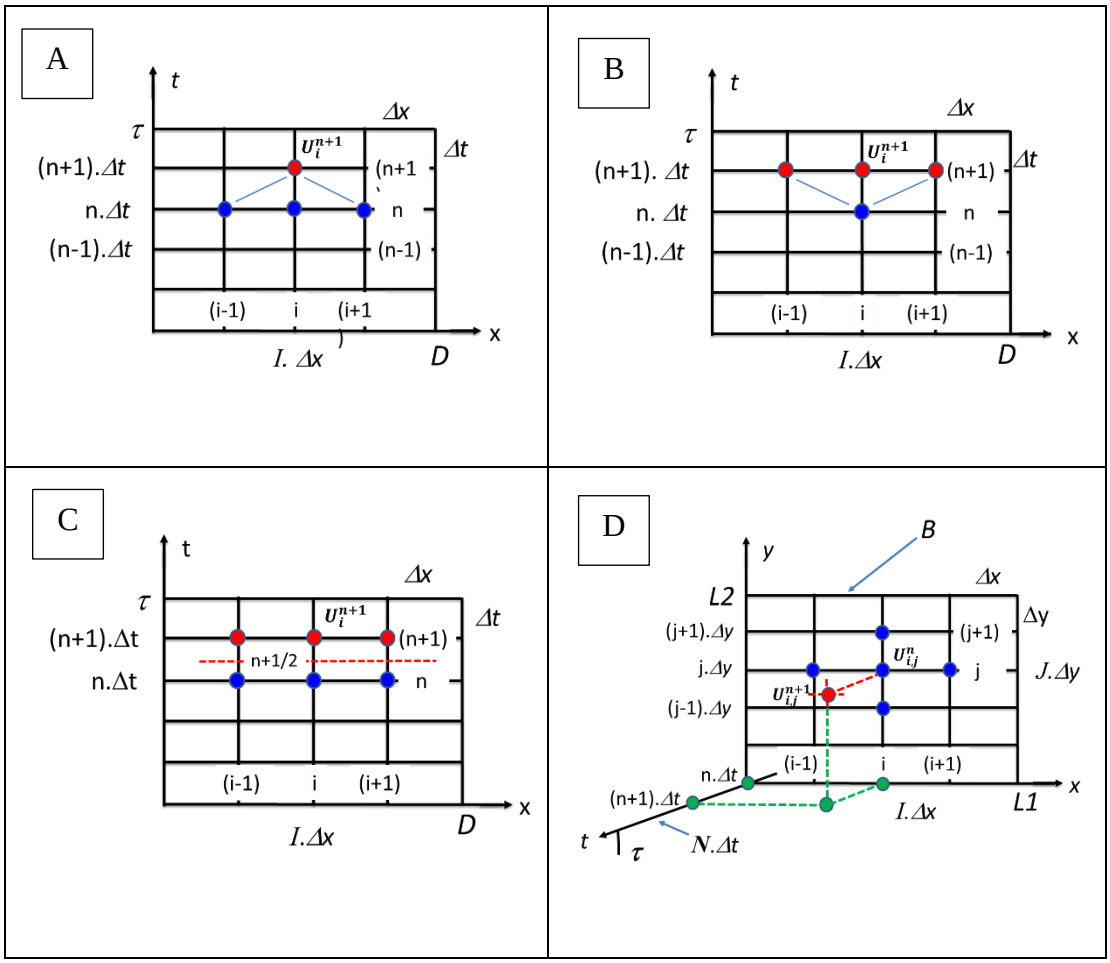

Figure 20.2 – A-D: Discrete domain of dependence of the solution at (*i, n+1*): (A) Explicit, (B) Fully implicit, (C) Mid-time centered - Crank Nicholson, (D) 3D explicit.

The implications of the use of the distinct types of discretization will be evident in the following, associated with the form of the implied operators.

Let us assume that *L* is a linear differential operator relating a dependent variable *u* to the space coordinate *x* and time *t*. The linearity of *L* does not pose an essential limitation, due to the possibility of linearizing the problem. Considering a 1D problem is convenient for simplicity.

Then, the differential problem may be formulated as follows:

$$\frac{\partial u}{\partial t} = L(t, x, \frac{\partial^k}{\partial x^k})u, \quad k = 1, K \tag{20.1}$$

Expanding *u* in terms a Taylor series around *t* in terms of a time increment *Δt*, then:

$$u(t+\Delta t, x) = u(t,x) + \frac{1}{1!}\frac{\partial u}{\partial t}\Delta t + \frac{1}{2!}\frac{\partial^2 u}{\partial t^2} + HOT \qquad (20.2)$$

where *HOT* means terms of higher-order in Δt. This is a two-level time-discrete formulation, implying also that L is independent of t, a reasonable assumption for many practical applications.

Now, equation (20.2) may be written in operator form as:

$$u(t+\Delta t) = exp(\Delta t \frac{\partial}{\partial t})u(t,x) = exp(\Delta t \cdot L)u(t,x) \qquad (20.3)$$

Equation (20.3) opens the way to all the different approximations for two-level time discretization. For example, a suitable operator pre-multiplication of this equation leads to:

$$exp(-\frac{\Delta t}{2} \cdot L)u(t+\Delta t) = exp(\frac{\Delta t}{2} \cdot L)u(t,x) \qquad (20.4)$$

which is a time-centered (Crank-Nicholson) approximation.

The essential difference between the previous two equations consists in the level of algebraic difficulty to obtain the solution for the time-marching at the advanced time level $t + \Delta t$. In the case of equation (20.3), the advanced values may be obtained explicitly from values at time level *t*, while in the case of the Crank-Nicholson approximation, equation (20.4), a simultaneous set of equations (usually linear) must be solved. In the first case, the approximation is explicit, while in the second case the approximation is implicit.

A fully implicit approximation may be obtained again by pre-multiplication of equation (20.3) in the form:

$$exp(-\Delta t \cdot L)u(t+\Delta t) = u(t,x) \qquad (20.5)$$

Another important aspect of the formulation shown as equation (20.3), is that it allows for considering problems in multidimensional space coordinates as a succession of 1D problems. Suppose for the moment that the space operator L may be written as the sum of three operators L_1, L_2, L_3, each one depending on *x*, *y*, *z*. Then, for example, it may be written that:

$$exp(-\Delta t \cdot L_1)\, exp(-\Delta t \cdot L_2)\, exp(-\Delta t \cdot L3)u(t+\Delta t) = u(t,x) \qquad (20.6)$$

In the previous expressions, it has been assumed that operators commute. This equation may be split by suitable definitions, in a sequence of 1D problems as follows:

$$\begin{aligned} exp(-\Delta t \cdot L_1)u^* &= u(t,x) \\ exp(-\Delta t \cdot L_2)u^{**} &= u^* \end{aligned} \qquad (20.7)$$

$$exp(-\Delta t \cdot L_3)u(t+\Delta t) = u^{**}$$

This approach, of fully implicit type in time is known as the Dyakonov splitting technique and this and many other similar have been used since the mid of the 60s. What is important to understand is that intermediate values do not necessarily coincide with the time interpolation of u at intermediate times. This subject will be resumed later when be considering general types of operators.

20.1.1 Formulation of exact, discrete approximations (DA)

Now it becomes necessary to fix some ideas on the approximation of *L*. To conduct this task, the approximations of Mitchell and Griffiths, 1980, to space derivatives are in order.

Considering the expansion:

$$u(x \pm \tfrac{1}{2}\Delta x) = u(x) \pm \tfrac{1}{1!}\tfrac{\partial u}{\partial x}\tfrac{\Delta x}{2} + \tfrac{1}{2!}\tfrac{\partial^2 u}{\partial x^2}\tfrac{\Delta x^2}{4} + \cdots , \qquad (20.8)$$

and the definition of the centered space operator

$$\delta[u(x)] = u(x + \tfrac{1}{2}\Delta x) - u(x - \tfrac{1}{2}\Delta x), \qquad (20.9)$$

it follows that:

$$\frac{\partial}{\partial x} = \frac{2}{\Delta x} \cdot sinh^{-1}(\frac{\delta_x}{2}) = \frac{1}{\Delta x}(\delta_x + HOT) \qquad (20.10)$$

Then, considering that Δt, Δx are constant along the plane (t, x) *and* $t = n \cdot \Delta t$, $x = i \cdot \Delta x$, equation (20.3) may be written in exact difference form as:

$$u_i^{n+1} = exp\left[\Delta t \cdot L\left(i \cdot \Delta x, n \cdot \Delta t, \frac{2}{\Delta x} \cdot sinh^{-1}(\frac{\delta_x}{2}), \left(\frac{2}{\Delta x} \cdot sinh^{-1}(\frac{\delta_x}{2})\right)^2\right), \cdots\right] u_i^n \qquad (20.11)$$

Please note that the power of an operator is understood as its iterative application. In equation (20.1.5), u_i^{n+1} is the exact value of u at $(t + \Delta t, x + \frac{\delta_x}{2})$. All Finite-difference equations come from the truncation of this expression and operator manipulations as considered before.

20.1.2 Truncation of exact Difference Approximations (DA) and the equations really solved. Local truncation error (TE). Consistency of DA.

The DA as presented are not useful for numerical calculations and truncated expressions must be used. In doing this, the discrete values for the dependent variables and the discrete form of the operators must be introduced.

In what follows it is assumed that there exists a smooth, continuous function *v* in the integration domain and a discrete function *U*, which coincides with *v* in the discrete set of points that constitute the "grid" or "mesh". This grid, in the usual case of 1D, unsteady TH problems, is a series of points along the x-axis, which repeat in a set of parallel lines in the time axis. The separation of points does not need to be uniform. The previous assumption allows series expansions of these discrete functions (sometimes, for simplicity, nomenclature will be interchanged). Also, in what follows, the implementation of analogs of initial conditions (IC) and boundary conditions (BC) will not be considered explicitly.

The discrete version of the operator *L* will be denoted as L_h and will be applied to *U*, while *L* will be applied to *u* as before. Boundary conditions will be denoted as *B* and its discrete version as B_h.

The goal of the application of numerical methods is the computation of a set of unknown values for *U*, which is a suitable approximation to those of *u* that is also unknown.

A significant difference between the original differential equation and the difference equation is that equation (20.11) cannot be used in practical terms. It must be truncated. This gives origin to the *truncation error* and to the equation that is really solving the numerical scheme. This concept will be considered later in more detail.

To illustrate this aspect, a simple example should suffice.

The scalar linear advection equation is:

$$\frac{\partial u}{\partial t} + C \cdot \frac{\partial u}{\partial x} = 0 \qquad (20.12)$$

where *C* is the constant transport velocity.

If a simple explicit difference approximation is used for the time derivative and centered approximations are used for the space derivative, the following expression results:

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = -C \cdot \frac{U_{i+1}^n - U_{i-1}^n}{\Delta x} \; ; \qquad C > 0 \qquad (20.13)$$

The definition of L_h is easy. Expanding this expression, by considering the definition of *v* before and replacing it by *u* (this explanation will be obviated from here on) it results:

$$\frac{\partial u}{\partial t} + \frac{\Delta t}{2} \frac{\partial^2 u}{\partial t^2} + C \cdot \frac{\partial u}{\partial x} + \frac{C.\Delta x^2}{6} \frac{\partial^3 u}{\partial x^3} = O(\Delta t^2, \Delta x^4) \qquad (20.14)$$

This equation is the differential equation that is solved using the above-mentioned approximation. The additional terms appearing constitute, in terms of the nomenclature of this chapter, the *local truncation* error of the discrete approximation.

To establish the conditions to assure that the computed values of U are a proper representation of u at the grid points, some formal definitions are now possible and will be introduced as follows.

CONSISTENCY: Let us consider a dependent variable that is sufficiently differentiable in a domain D and its boundary R. If a proper norm of the truncation errors $L - L_h$, and $B - B_h$ tend to zero when the increments of the independent variables tend to zero in some way, then the discrete scheme L_h, B_h is said consistent with the differential operators L, B

It may be observed that equation (20.13) above is consistent with the original equation.

STABILITY: Given a function U defined in all the points of a grid in D and its boundary R, then, if it exists a finite quantity K such that in a proper norm, it is

$$\|U\| \leq K\{\|L_h(U)\| + \|B_h(U)\|\} \qquad (20.15)$$

for all the functions U defined in D+R, then the discrete scheme is said stable. Consistency and Stability may be conditional.

CONVERGENCE: Given a function U defined in all the points P of a grid in D and its boundary R, then, considering linear operators, if the discrete scheme is consistent and stable, the discrete scheme is convergent, i.e.:

$$\|u(\mathrm{P}) - U(\mathrm{P})\| \quad \rightarrow 0 \quad \text{(tends to zero)} \qquad (20.16)$$

This is the equivalence theorem of Peter Lax that allows constructing discrete, convergent solutions of a differential problem. Formal proofs of this theorem may be found in texts dealing with numerical analysis, e.g. Marchuk, (1975).

20.1.3 The introduction of artificial viscosity

In this paragraph, the concept of numerical (or artificial) viscosity will be introduced. The need to stabilize numerical solutions by artificial, numerical means was introduced by Von Neumann and Richtmyer, (1950). Many papers in the 60s refer to this pioneering contribution as its sole reference, thus stressing the originality and the importance of the above paper. In dealing with the problem of shock waves, said Authors introduced a non-physical, numerically dissipative term proportional to the gradients of the velocity, using it at the zones of large gradients and setting it to zero at “normal” zones.

The linear, 1D scalar advection-diffusion equation is paradigmatic in showing the effects of proper control of the truncation error of a discrete approximation. In coping with

truncation error, it must be remembered again that it is a necessary consequence of getting useful working approximations to conservation equations. Truncation errors cannot be avoided in real life. One of the more influential papers in the clarification of the effects of truncation error was due to Hirt, Hirt, (1968). He introduced the concept that the equation solved by the numerical scheme is the one obtained by adding the terms arising from the truncation error up to the second order. The lowest order terms of Taylor´s expansion must be the original differential equations. The remaining ones are truncation errors. Numerical consistency forces these added terms to tend to zero when the discretization intervals tend to zero in some way. Hirt showed how the added terms of even order in Δx could lead to unphysical behavior, by assimilating these terms to the so-called "artificial viscosity" or "numerical viscosity" because they were proportional to some power of space and time intervals. It is now interesting to consider explicitly the example given in Hirt, (1968).

The linear advection-diffusion equation is:

$$\frac{\partial u}{\partial t} + C \cdot \frac{\partial u}{\partial x} = \alpha \frac{\partial^2 u}{\partial x^2} \qquad (20.17)$$

If centered approximations are used for the space derivatives, then:

$$\frac{\partial u}{\partial t} + \frac{\Delta t}{2} \frac{\partial^2 u}{\partial t^2} + C \cdot \frac{\partial u}{\partial x} = \alpha \frac{\partial^2 u}{\partial x^2} + O(\Delta t^2, \Delta x^2) \qquad (20.18)$$

This is a hyperbolic equation, instead of the original parabolic one. The analysis of its domain of dependence and the need to keep a positive diffusion coefficient leads to the conditions needed to keep the solution stable, namely:

$$\alpha - C^2 \cdot \frac{\Delta t}{2} > 0 \qquad \frac{2\alpha}{\Delta t} \leq \left(\frac{\Delta x}{\Delta t}\right)^2 \qquad (20.19)$$

The most interesting result for this equation comes from considering α=0 and using a non-centered approximation for the space derivative. In this case:

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = -C \cdot \frac{U_i^n - U_{i-1}^n}{\Delta x} ; \qquad C > 0 \qquad (20.20)$$

Following the same procedure as before, it may be found that:

$$\frac{\partial u}{\partial t} + C \cdot \frac{\partial u}{\partial x} = \frac{C \cdot \Delta x}{2} \cdot \left(1 - \frac{C \Delta t}{\Delta x}\right) \cdot \frac{\partial^2 u}{\partial x^2} + O(\Delta t^2, \Delta x^2) \qquad (20.21)$$

where:

$$C_{OU} = \frac{C \cdot \Delta t}{\Delta x} \qquad (20.22)$$

is the cell Courant number. As may be seen in the previous equation, the diffusion coefficient comes directly from the discretization parameters and the advection velocity. Keeping this coefficient positive implies keeping the cell Courant number less or equal to unity. C_{OU} >1 implies that the fluid transverses more than a space interval (a computing cell) in one time step interval. This is not allowed on physical grounds. Violating this condition leads to growing oscillations (physically sound growing oscillations!). Hirt, 1968) analyzed the more important case of non-linear problems in the same way.

The explicit time, centered space approximation given by equation (20.20) constitutes the "Forward time, upwind space" approximation to the scalar wave equation or, in short, the FTUS method. This is the simplest way to construct solutions in CFD. It is the usual approximation used in the codes to stabilize calculations or to regularize ill-posed models like in industrial SYS-TH codes as will be discussed later, through the introduction of controlled numerical diffusion. Many other, quite sophisticated methods have been developed with this philosophy. The interested reader may consider the books by Laney, (1998) and the two-volume edition by Fletcher, (1991). Adequate discretization permits keeping low truncation error and allowing the computation of unstable flows.

The adequate treatment of truncation error has been shown to allow the computation of solutions. It will be shown in what follows that this is a useful method to get linearized forms of the non-linear governing equations, with controlled truncation error. Important examples of this assertion may be found in Marchuk, (1975). In a practical case of direct application to Nuclear Safety, the SETS (for Stability-Enhanced, Two-Step) method, Fletcher, (1991), was developed to eliminate the C_{OU} limit restriction.

In what follows this will be exemplified, simply by showing how to use this concept to linearize Burger´s type operators. Let:

$$\frac{\partial u}{\partial t} + L_1(u) + N(u) = 0 \qquad (20.23)$$

be the equation under analysis, where L_1 is a linear operator and N is a nonlinear operator.

Let us assume that N is restricted to the form:

$$N(u) = L_2(u) \cdot A(u) \qquad (20.24)$$

where $L_2(u)$ is linear in u and A is such that the algebraic problem resulting from the discrete approximation of the previous equations is also linear.

Burger's equation is a useful example. In this case:

$$A \equiv u, \quad L_1 = -\varepsilon \cdot \frac{\partial^2}{\partial x^2}, \quad L_2 = \frac{\partial}{\partial x} \qquad (20.25)$$

A Crank-Nicholson approximation may be written as:

$$\left[I + \frac{\Delta t}{2} L_1 + \frac{\Delta t}{2} A^* \cdot L_2\right] u^{n+1} = \left[I - \frac{\Delta t}{2} L_1 - \frac{\Delta t}{2} A^* \cdot L_2\right] u^n \qquad (20.26)$$

where I is the identity operator and A^* is an operator that is independent of time and a suitable approximation to A to be defined in what follows.

Expanding this expression in a Taylor series around n, we get:

$$\begin{gathered} u + \Delta t \frac{\partial u}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 u}{\partial t^2} + \\ + \frac{\Delta t}{2} L_1 + \frac{\Delta t}{2} A^* \cdot L_2 + \Delta t \cdot \frac{\Delta t}{2} \cdot \left[\frac{\partial L_1}{\partial t} + A^* \cdot \frac{\partial L_2}{\partial t}\right] + \\ -u + \frac{\Delta t}{2} L_1 + \frac{\Delta t}{2} A^* \cdot L_2 = O(\Delta t^3) \end{gathered} \qquad (20.27)$$

To obtain an estimation for A^*we now ask: under which conditions is this equation an "exact", i.e., an $O(\Delta t^2, \Delta x^2)$ approximation to the solution of the original equation? The answer comes from subtracting the previous equation from the original one:

$$\frac{\partial u}{\partial t} + \frac{\Delta t}{2} \frac{\partial^2 u}{\partial t^2} + L_1 + A^* \cdot L_2 + \frac{\Delta t}{2} \cdot \left[\frac{\partial L_1}{\partial t} + A^* \cdot \frac{\partial L_2}{\partial t}\right] = O(\Delta t^2) \qquad (20.28)$$

Derivation of the original differential equation and replacement leads to:

$$A^* \cdot L_2\left(u^{n+1/2}\right) = A\left(u^{n+1/2}\right) \cdot L_2\left(u^{n+1/2}\right) \qquad (20.29)$$

Then,

$$A^* \equiv A\left(u^{n+1/2}\right) \qquad (20.30)$$

which is the $O(2,2)$ expression searched. The numerical approximation to the intermediate time step may be obtained in different linear ways. Because of the CN formulation, terms involving added diffusion terms do not arise. If A^* is evaluated as shown, then, the technique coincides with a predictor-corrector scheme based on the evaluation of "non-linear" coefficients evaluated at $t=n\Delta t+\Delta t/2$. This result is well known. As may be seen from the above derivation, truncation error analysis was used again in a convenient way.

20.1.4 Phase error in the solution of DA

It is important to point out that there is more to consider than a diffusive truncation error…

In the previous paragraph, the order of some discrete approximations has been discussed. There is another aspect that deserves attention when the coupling and type of dependent variables perturbations govern the dynamics of the solution. Depending on the type of

perturbation, the wave velocity of a given numerical scheme must be considered. The discrete wave velocity is the velocity that a wave packet moves with, compared to the fluid velocity. It is well known that the degree of delay of density (or temperature) perturbations concerning fluid velocity is key to the appearance of instabilities. One of the most important references in this subject is the review by Trefethen, (1982), who showed how this property of a numerical scheme affects the transport velocity of a signal as a function of wavenumber and the discrete (finite-differences) approximation. Figures 20.3 and 20.4 consider the linear advection of a scalar with two different waveforms, namely a wave packet and a smooth Gaussian. In both cases, the transport velocity is C = 1, the space interval is Δx = 1/160, the Courant number is C_{OU} = 0.4 and the total time of integration is t = 2. The numerical scheme is LEAPFROG, defined by the simplest numerical scheme of order $O(2,2)$, i.e.

$$\frac{U_i^{n+1}-U_i^{n-1}}{2\Delta t} = -C\frac{U_{i+1}^n-U_{i-1}^n}{2\Delta x} \qquad (20.31)$$

The analysis for this behavior is simple and comes from considering that numerical schemes transport wave packets composed of waves of different frequencies and wavelengths with different velocities. The speed of propagation of the waves is dependent on κ., where κ is the wavenumber. Define ω as the angular wave frequency. If λ is the wavelength, the resolution is defined as $m=\lambda/\Delta x$. Additionally, by definition: $\theta = 2\pi / m$.

Defining the group velocity as:

$$C_G = \frac{d\omega}{d\kappa} \qquad (20.32)$$

and replacing each term of the terms in the difference equation by $exp(\omega t - \kappa x)$, e.g.:

$$U_{i-1}^{n+1} = e^{j\,[\kappa\cdot(x-\Delta x)\,-\,\omega\cdot(t+\Delta t)]} \qquad (20.35)$$

then:

$$\frac{C_G}{C} = 1 - \frac{1-C_o^2}{2}\cdot\theta^2 \qquad (20.36)$$

Using this equation and the above-mentioned discretization, the waves should be transported from x=0.5 to x=2.5. As may be seen in Figure 20.3, this is fairly confirmed for a smooth Gaussian wave perturbation. When the same signal is modulated by a sine wave having wave number 125.7 and a wave resolution of 8 points per wavelength, the group velocity is nearly 0.74. This results in the wave packet transport from x = 0.5 to x = 2, Figure 20.4. This property may be also verified in multidimensional systems.

The above means that there is more than simply considering diffusive truncation error in assessing the quality of a solution. A simple conclusion is that the transport of the wave packet is correct, but its velocity (the group velocity) lags the phase velocity (that is also different from the fluid velocity) by 25%. It may be concluded that, depending on the type of perturbations, there is a possibility of getting unexpected results about the computation of the onset of instabilities of flows.

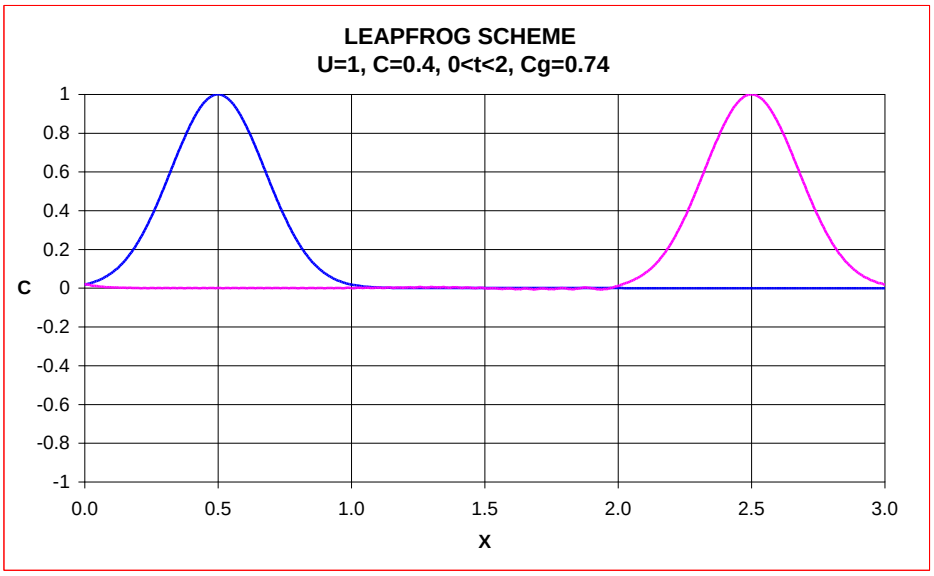

Figure 20.3 – The transport of a smooth Gaussian wave under LEAPFROG, Trefethen, (1982).

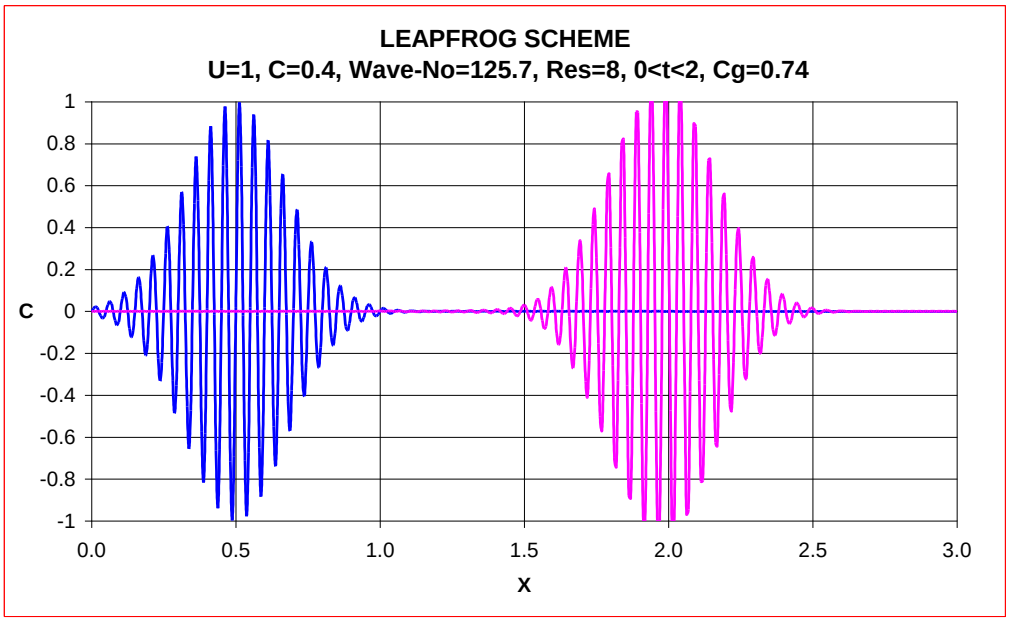

Figure 20.4 –The transport of a wave packet under LEAPFROG, Trefethen, (1982).

On the other hand, the second case imposes further considerations when choosing a norm to measure the error of a numerical solution.

20.1.5 The meaning and control of numerical, non-physical solution oscillations

In many cases of practical significance, the computed solution is stable but shows oscillations. Not necessarily, this is due to some intrinsic (physical) properties of the solution but, apart from being unexpected, these oscillations are spurious. The 1-D, linear, steady-state, advection-diffusion equation solution is another paradigmatic example of this. This equation may be written as:

$$P_e \cdot \frac{du}{dx} - \frac{d^2u}{dx^2} = 0$$

$$u(0) = 0 \qquad u(1) = 1 \tag{20.37}$$

where P_e is the Peclet number, defined as $P_e = C.L/\alpha$, where C is the advection velocity, L is the length of the 1D domain, and α is the mass diffusion coefficient. This equation is the so-called "tough problem". Figure 20.5 shows the solution using centered differences for different Peclet numbers. Solid points show the exact solution. Hollow points show the exact, numerical, stable solution with non-growing oscillations. The oscillations (wiggles) are due to the non-proper boundary layer resolution at high Peclet numbers. Asymptotic analysis, Ferreri, 1985, or, more simply, the introduction of upwinding cuts the problem.

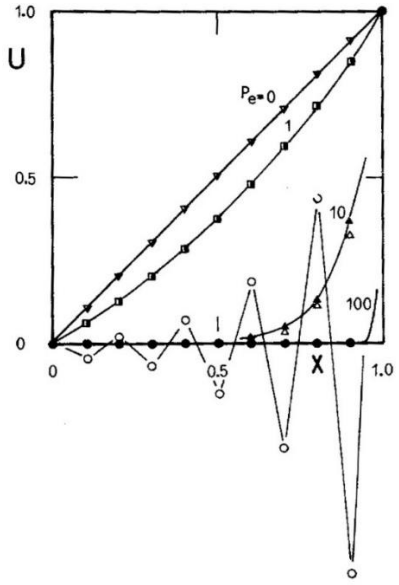

Figure 20.5 – Spurious oscillations in the solution of the 1D, linear, steady-state, advection-diffusion equation and their elimination by asymptotic analysis, Ferreri, (1985).

The existence of spurious oscillations in the solution of conservation equations may be the consequence of the non-appropriate resolution of the boundary layer behavior of the solution, like in this case. Then, suppressing the oscillations may be non-conservative or, equivalently, allow computing a solution not showing all the important aspects of the physics. Work done in the 80s served to clarify these aspects. P. Gresho at the LLNL, Gresho and Lee, (1981), contributed significantly to this subject. However, an earlier discussion on the effects of discretization may be also found in Hirt, (1968).

To conclude this paragraph, it may be stated that non-growing oscillations in a computed solution may be an indication of non-appropriate discretization. Forcing their elimination may be a cause of losing information on the physics and at the same time, of making possible the calculation.

20.2 The solution of Parabolic PDE

Considering time-dependent, diffusive PDE leads to the field of parabolic equations. In earlier paragraphs, we have discussed them (i.e., eq. 20.3). Depending on the numerical scheme and the linearity of the equations, the discrete expressions may lead to more simple or more complicated numerical algorithms to obtain the solutions.

This applies to single or two-phase TH. In the second case, the simultaneous conservation PDEs impose considering various levels of implicitness. In his series of lectures on simulation, one of the pioneers, John Mahaffy at PSU, Mahaffy, 1982, established the coupling of different conservation equations as the key to different degrees of implicitness for practical applications. The *fully explicit method* consists of solving the DAs for density and velocity calculating the advanced time values in terms of old values only. This leads to simple numerical algorithms at the cost of imposing severe time step limitations due to the Courant limit described before. The *semi-implicit method* consists of advancing the density calculation in explicit form and the velocity by implicit methods. This is a more complicated method, but it was used in most SYST-TH codes for safety evaluations. The obvious generalization is considering a *fully implicit method* that leads to a coupled set of non-linear equations that are usually free of stability problems. It is important to keep the simplicity of the numerics and, at the same time, have fewer stability restrictions in the time step. This aspect was considered in the *nearly implicit* DA. The introduction of partial steps in the computation of advanced time variables was the solution proposed in SYS-TH codes. John Mahaffy introduced the *stability-enhancing two-step method* as the SETS Method, Mahaffy, (1982), which consists in the recalculation of the time-advanced variables adding a stabilizing step to the nearly implicit approximations. This procedure applies to all the variables (mass, energy, scalars, and momentum), resembling a fully implicit approximation with a reduced computational effort.

Before continuing, it is useful to introduce a here deeper discussion of the equation (20.20). The up-wind derivative for advection terms was presented in its simplest form and it is a particular form of the so-called donor-cell approximation of advection terms in conservation equation.

Figure 20.6 shows a control volume for the computation of the advective term in 1D flow.

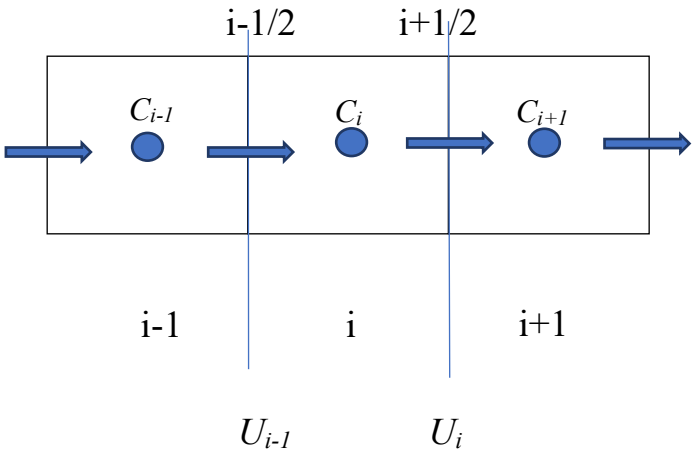

Figure 20.6 – Control volumes for the computation of advective terms in 1D flow.

The cross-section of the 1D problem is considered uniform. In the case of scalars, like density, concentration, and energy/temperature, they are centered at cell i. For convenience, velocities are defined at cell boundaries. The control volume for velocities is displaced $\Delta x/2$.

In a mixed classic style of the FORTRAN codes of the 70's, this automatic way of introducing artificial, numerical viscosity, now for the momentum equation is written as:

FUX = [(U(I)+U(l+1)) * (U(I)+U(I+1)) + α*ABS(U(I)+U(I+1)) * (U(l) -U(I+1))

- (U(I-1)+U(I)) * (U(I-1)+U(I)) - α*ABS(U(I-1) + U(I)) * (U(I-1)-U(I))] / (4*Δx)

(20.38)

where FUX represents $\frac{\partial U^2}{\partial x}$ in the momentum equation. α represents a coefficient that allows for introducing a variable degree of upwind differencing. α=0 implies a centered approximation, while α=1 imposes full up-winding or donor cell approximation. It may be verified that this approximation holds for positive or negative values of the velocity at the control volumes faces. When needed, simple averages of these velocities may be used. The result of imposing α=1 is an added artificial viscosity ABS(U).$\Delta t/\Delta x$. Similar expressions hold for the other terms.

The *fractional step method* to be introduced in what follows is a formal way to consider the last DA solutions, linearizing non-linear sets of equations.

20.2.1 The approximation of the solution of time-dependent problems, step by step splitting

It is useful to reconsider equation (20.3) that is written here in a simpler form.

$$u(t+\Delta t) = exp(\Delta t \cdot L)u(t,x) \tag{20.39}$$

where now it is assumed that L does not depend on time and is a linear operator. A way to deal with particular forms of non-linearities was introduced before. In terms of the analysis, Marchuk, (1975), this was a *predictor-corrector method*. In the case of considering a non-homogeneous equation, the procedure is similar. The *splitting up method* will be considered, as delineated in Marchuk, (1975), to guide the analysis.

Assuming that L may be expressed as the sum of several simple, linear, and time-independent operators as

$$L = \sum_{s=1}^{m} L_s \tag{20.40}$$

The following discrete procedure is introduced, consistent with the definition used in section 20.1:

$$\frac{U_i^{n+\frac{1}{m}} - U_i^n}{\Delta t} = L_1^n\left(U_i^{n+\frac{1}{m}}\right) ;$$

$$\frac{U_i^{n+1} - U_i^{n+(m-1)/m}}{\Delta t} = L_m^n(U_i^{n+1}) \; . \tag{20.41}$$

Considering m = 3 leads to the Dyakonov splitting for tridimensional problems that is a first-order time approximation. If equation (20.39) were non-homogeneous, the previous scheme only changes to consider an independent term in the last equation evaluated at time t. The stability of this scheme is unconditional, and the proof may be found in Mitchell and Griffiths, (1980). In this book, generalizations to consider quasi-linear problems in which L depends on time and the dependent variable may also be found. The basic concept is considering as many intermediate steps as necessary to evaluate the non-linearities to advance from time $t = n \cdot \Delta t$ to $t = (n+1) \cdot \Delta t$. In section 20.1, the simplest case of a quasi-linear equation linearization was introduced.

The algorithmic details and the computational cost will be considered in section 20.3.5.7.

To fix ideas, it is useful to recall the original formulation of the SETS Method, Mahaffy, (1982). This method has been used to implement the space derivatives in one of the major TH codes, the TRACE code.

20.2.2 Explicit and implicit approximations in one and multiple space dimensions, Alternating Direction Implicit (ADI) methods

In multiple space dimensions, equation (20.41) may remain the same formally but the computational effort is hidden in the multiple steps. To fix ideas let us consider two space dimensions and a simple space operator like a Laplacian. Similarly to equation (20.41), it is possible to consider an explicit method, allowing to advance in the time applying space boundary conditions, according to the procedure suggested in Figure 20.2 (D), applying boundary conditions along *B* before advancing from time $t=n\cdot\Delta t$ to $t=(n+1)\cdot\Delta t$. The stability of such a procedure imposes limitations in the selection of time steps, Mitchell and Griffiths, (1980); Marchuk, (1975). To avoid these limitations, one common procedure consists in integrating along time using implicit methods. However, the cost of solving the simultaneous systems of equations may be very high because of the large matrices arriving from the multidimensional discretization. To avoid this problem, operator splitting, like, equations (20.41) is in order. The Dyakonov splitting considered before is a possible scheme. However, alternative splitting procedures become popular in the fifties-sixties. They were introduced by Peaceman and Rachford, (1955), Douglas and Rachford, (1956), and some other authors. The original applications dealt with the solution of elliptic equations coming from fluid flows in porous media by pseudo transient techniques. To illustrate these techniques, similarly to equation (20.39), it is

$$u(t+\Delta t) = exp(\Delta t \cdot L)u(t,x) = exp(\Delta t \cdot (L_x + L_y))u(t,x) \qquad (20.42)$$

that may be expanded as

$$u(t+\Delta t) = exp(\Delta t \cdot L_x).exp(\Delta t \cdot L_y)u(t,x) \qquad (20.43)$$

Considering discrete values, it is

$$-\frac{\Delta t}{2\Delta x^2}.\delta_x^2)(1-\frac{\Delta t}{2\Delta y^2}.\delta_y^2)U_i^{n+1} = (1+\frac{\Delta t}{2\Delta x^2}.\delta_x^2)(1+\frac{\Delta t}{2\Delta y^2}.\delta_y^2)U_i^n \qquad (20.44)$$

Introducing an intermediate value U_i^{n+1*}

$$(1-\frac{\Delta t}{2\Delta x^2}.\delta_x^2)U_i^{n+1*} = (1+\frac{\Delta t}{2\Delta y^2}.\delta_y^2)U_i^n \qquad (20.45)$$

and

$$(1-\frac{\Delta t}{2\Delta y^2}.\delta_y^2)U_i^{n+1} = (1+\frac{\Delta t}{2\Delta x^2}.\delta_x^2))U_i^{n+1*} \qquad (20.46)$$

This is the Peaceman-Rachford ADI Method. The application of boundary conditions has special aspects in irregular boundaries and a throughout discussion may be found in Mitchell and Griffiths, (1980). Similarly, other expressions are available. In the Authors´ experience, the use of the Dyakonov formulation is simpler.

20.3 The solution of Elliptic PDE

All the discretization methods for PDE lead to a linear system of equations that must be solved. If the problem is nonlinear then, the solution can be solved iteratively, solving a series of linear system of equations. So, at the very basis of all computer codes for the solution of PDE, there must be a computer routine to solve linear systems. Moreover, many times this is the step that represents the largest consumption of computing time

We discuss below the solution of the linear systems arising from the numerical discretization of elliptic PDE. There are already very good commercial and open-source libraries for this task. However, it is very important for a user of these libraries to understand the different algorithms involved, to choose the most appropriate for its needs. Many TH computational codes allow for the choice of the solver of system equations and some of the associated parameters (tolerances, maximum number of iterations, type of solver).

20.3.1 Characteristics of the linear system

For simplicity, the solution of the Poisson equation

$$k\Delta\phi = -Q \tag{20.47}$$

on a rectangular domain $\Omega = [0, L_x] \times [0, L_y]$, will be studied. Dirichlet boundary conditions

$$\phi = \overline{\phi} \tag{20.48}$$

at the boundary of Ω are assumed, k is the thermal conductivity (units $[W/m.K]$), and Q is a heat source term (units $[W/m^3]$). The problem is discretized with L steps in the x direction and M steps in the y direction, as shown in Figure 20.7.

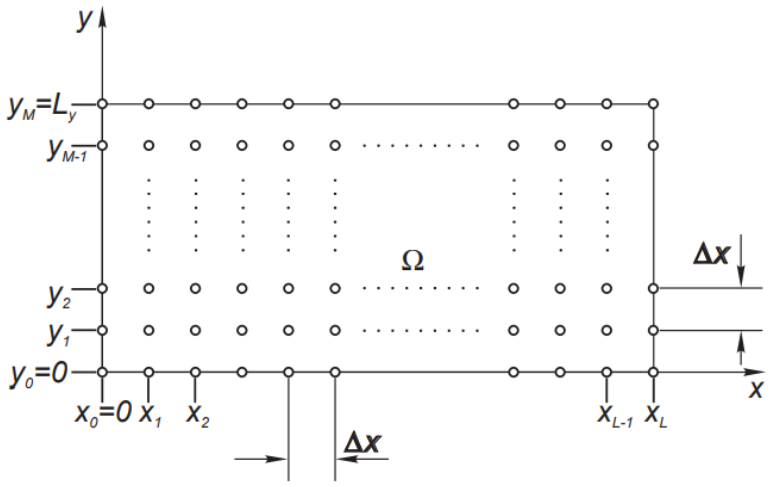

Figure 20.7 – Rectangular domain for the Poisson problem.

Let $\phi_{l,m}$ be the approximate value for ϕ at the node (x_l, y_m), then by finite difference approximation of equation (1), the following equation is obtained

$$k\left(\frac{\phi_{l+1,m}-2\phi_{l,m}+\phi_{l-1,m}}{\Delta x^2}+\frac{\phi_{l,m+1}-2\phi_{l,m}+\phi_{l,m-1}}{\Delta y^2}\right)=-Q_{l,m} \tag{20.49}$$

This represents one linear equation for each interior node ($1 \le l \le L$, $1 \le m \le M$), so that there are $N = (L-1)\times(M-1)$ equations. On the other hand, there is one unknown for each interior node since the boundary nodes have an imposed value, due to the Dirichlet boundary condition. The system of equations can be written as

$$\mathbf{K}\boldsymbol{\phi} = \mathbf{f} \tag{20.50}$$

where $\mathbf{K}$ is the square matrix of the system, of size $N \times N$, and $\boldsymbol{\phi}$ and $\mathbf{f}$ the vector of unknowns of length and right-hand side respectively, both of length N. For simplicity, it will be assumed that $L_1 = L_2$, and $L = M$, so that $\Delta x = \Delta y$, and the matrix system can be expressed in block matrix form as

$$\mathbf{K} = \frac{k}{\Delta x^2}\begin{bmatrix} \overline{\mathbf{K}} & -\mathbf{I} & 0 & 0 & \cdots & \cdots & \cdots & \cdots \\ -\mathbf{I} & \overline{\mathbf{K}} & -\mathbf{I} & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & -\mathbf{I} & \overline{\mathbf{K}} & -\mathbf{I} & 0 & \cdots & \cdots & \cdots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & 0 & 0 & -\mathbf{I} & \overline{\mathbf{K}} \end{bmatrix} \tag{20.51}$$

where $\mathbf{0}, \mathbf{I}$, and $\overline{\mathbf{K}}$ are a square matrix of size $M-1$. $\mathbf{0}$ and $\mathbf{I}$ are the null and identity matrices, respectively, and $\overline{\mathbf{K}}$ is the following matrix,

$$\overline{\mathbf{K}} = \begin{bmatrix} 4 & -1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots \\ -1 & 4 & -1 & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & -1 & 4 & -1 & 0 & \cdots & \cdots & \cdots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 \end{bmatrix} \tag{20.52}$$

The vector of unknowns $\boldsymbol{\phi}$ is the collection of unknowns traversing the internal nodes of the grid by columns,

$$\boldsymbol{\phi} = \begin{bmatrix} \phi_{11} \\ \phi_{12} \\ \phi_{13} \\ \vdots \\ \phi_{21} \\ \phi_{22} \\ \phi_{23} \\ \vdots \\ \phi_{L-1,1} \\ \phi_{L-1,2} \\ \phi_{L-1,3} \\ \vdots \\ \phi_{L-1,L-1} \end{bmatrix} \tag{20.53}$$

as shown in Figure 20.8. The unknown for node (l,m) is located at position $(l-1)(L-1)+m$ in the $\boldsymbol{\phi}$ vector.

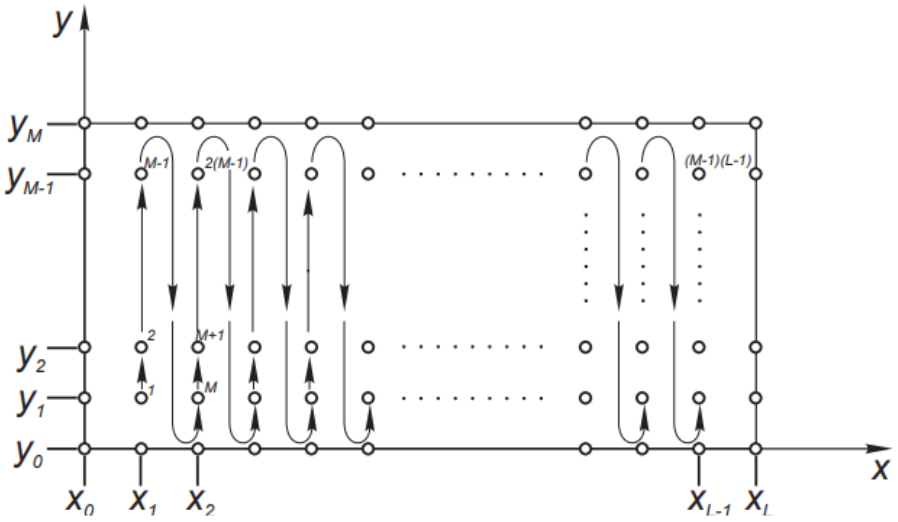

Figure 20.8 – Numbering of unknowns for the internal nodes

Consider an internal node at position (l, m). The equation for this node (see equation 0) involves the unknown at the node (l, m) itself, and the four neighbors $(l \pm 1, m \pm 1)$ (see Figure 20.9). The pattern of nodes whose unknowns are linked with a specific node is called the *stencil* of the operator. In this case, the stencil involves five nodes, and it is called the 5-node stencil operator for the Poisson equation. Noting that the unknown for node (l, m) is located at position $j = (l-1)(L-1) + m$, the five nonzero coefficients in that row are located at columns $j - L - 1, j - 1, j, j + 1, j + L - 1$

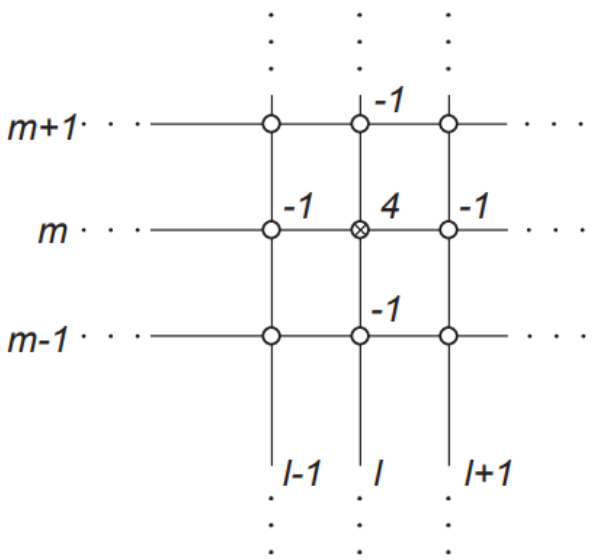

Figure 20.9 – Stencil of the operator.

This means that **K** is a *band matrix* i.e., a matrix such that the null coefficients are concentrated in a band near the diagonal, specifically

$$K_{ij} = 0, \qquad \text{for } |i - j| > a \tag{20.54}$$

where a is the bandwidth of the matrix. In this case, the bandwidth is $a = M$, (remember that $L = M$ is assumed) so that the ratio of the bandwidth to the size of the matrix is very low, $a/N = 1/M$, Fig. 20.10. It will be shown that this band characteristic of the matrix has a substantial impact on the computational effort needed to solve the linear system of equations.

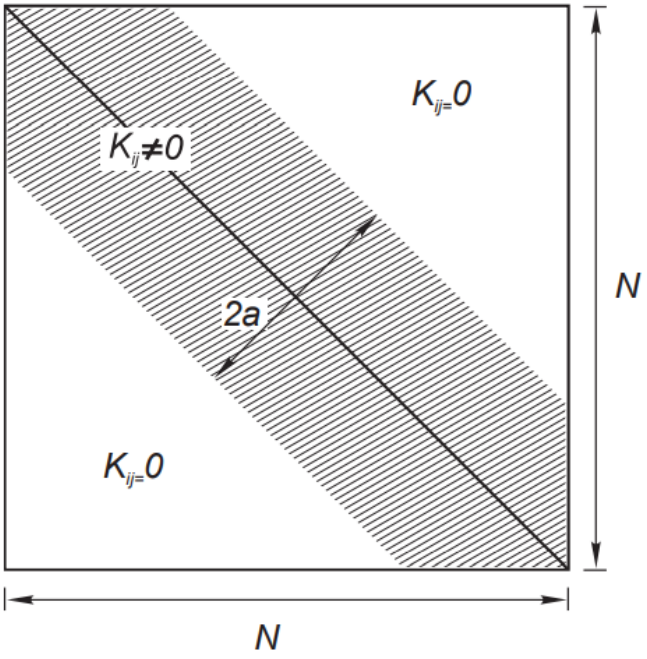

Figure 20.10 – Band matrix and definition of bandwidth.

20.3.2 Memory and computational time requirements for the solution of the linear system

Consider the solution of the linear system of equations (20.50), considering that the matrix has size $N \times N$, and bandwidth a. It is easy to see that during LU or Cholesky factorization, the coefficients outside of the band do not fill, i.e., the coefficients of the factored matrices **L**, **U** are also null for $|i - j| > a$. That means that the computer memory needed to store the factored matrices is $\sim 2Na$. On the other hand, if the matrix were stored as dense (i.e., if all coefficients were stored), then the required computer memory would be N^2. This means that there is a ratio,

$$\frac{memory(band)}{memory(dense)} = 2Na/N^2 = 2a/N \sim 2M/M^2 = 2/M \ll 1, \qquad (20.55)$$

between the memory required for storage as a band versus dense. This ratio becomes very low as the number of steps per side M increases.

Now consider the computing time, i.e., the number of floating-point operations needed to factor the matrix. For simplicity, we will assume Gauss elimination. Consider firstly factorization as a dense matrix. The algorithm proceeds first eliminating the a_{21}

coefficient by doing a row operation between the second and the first row. The number of operations needed to do this row operation is cN where c is a constant, denoting the number of operations needed to eliminate a single entry in the row. This operation must be done for all rows $j = 2, \cdots, N$, i.e., for $N-1$ rows, and the number of operations is $(N-1)N$. Then, the coefficients in column 2, rows 3 to N must be eliminated. That amounts to $N-2$ row operations of length $N-1$, and so on. The total number of operations is then,

$$N_{ops} = c\sum_{j=1}^{N-1}(N-1)N \tag{20.56}$$

where c is the number of operations needed to eliminate one coefficient. The summation can be computed in closed form and it results in

$$N_{ops} = c\frac{N^3}{3} + O(N^2) \tag{20.57}$$

Now, consider a banded matrix with bandwidth a. The number of operations to be done for the elimination of the coefficients is proportional to the number of non-null coefficients in the row. There are just $a+1$ non-null coefficients in the first row, $a+2$ in the second row, and so on until row $a+1$, where there are $2a+1$ non-null coefficients. In fact, due to the banded character of K all the rows have at most $2a+1$ non-null coefficients. For the first column, the coefficients must be eliminated from row 2 up to row $a+1$, i.e., a total of a rows and each one has at most $2a+1$ non-null coefficients, i.e., $ca(a+1)$ operations. For the second column, there are $a+2$ rows with at most $2a+1$ non-null coefficients. In general, it can be shown that there are for each column at most $2a$ rows with $2a+1$ non-null coefficients each, so the total number of operations is at most $N2a(2a+1)$

$$N_{ops}(\text{banded}) = 4ca^2N \sim O(N^2) \sim O(M^4) \tag{20.58}$$

The ratio between banded and dense is, then

$$\frac{N_{ops}(\text{banded})}{N_{ops}(\text{dense})} = \frac{4ca^2N}{cN^3/3} = 12(a/N)^2 \tag{20.59}$$

So, for instance, for a mesh of $L = M = 200$ steps in each direction, the total number of unknowns is $N = (L-1)(M-1) \sim LM = 40.000$, the bandwidth is $a = M$ and the ratio of operation count between banded and dense is,

$$\frac{N_{ops}(\text{banded})}{N_{ops}(\text{dense})} \sim \frac{2}{M^2} = 3e-4. \tag{20.60}$$

The huge difference between the computing times makes the use of banded matrices (or similar) almost mandatory.

Regarding the memory storage for the full matrix, the number of coefficients to be stored are

$$N_{mem}(\text{dense}) = N^2 \tag{20.61}$$

whereas for the banded matrix the storage is

$$N_{mem}(\text{banded}) = 2aN \sim O(N^{1.5}) \tag{20.62}$$

so, the ratio is

$$\frac{N_{mem}(\text{banded})}{N_{mem}(\text{dense})} \sim \frac{2aN}{N^2} = 2\frac{a}{N} = \frac{2}{M} \tag{20.63}$$

As a conclusion, it is evident that there is a large gain in using banded versus dense matrices, being the ratio $O(M^{-1})$ for the storage and $O(M^{-2})$ for the computing time.

For general non-structured grids, the concept of a banded matrix is generalized as a sparse matrix, but most of the concepts still apply, i.e., sparse matrix methods are much more efficient in terms of storage and computing time, than methods that do not consider that pattern, i.e., methods for dense matrices.

20.3.3 Basic concepts on iterative methods

Even if exploiting the banded pattern of the matrix greatly reduces the computational requirements, both in memory storage and computing time, they still grow at a very high rate, $O(N^{1.5})$ and $O(N^2)$ respectively, where N is the total number of unknowns. This means that if the user wants to refine the mesh by a factor of 2x in each dimension, then the number of unknowns grows by a factor of 4x, the required memory by a factor of 8x, and the computing time by 16x. Moreover, all the factorization methods are very difficult to implement in parallel, i.e., they cannot use efficiently the computational power of modern multicore processors, nor of HPC (for High Performance Computing) equipment.

A distinctive feature of the factorization methods is that they are “direct”, i.e., they solve the linear system to machine precision in a finite number of steps. Iterative methods, instead, produce a sequence of unknown vectors that converge in some norm to the solution.

20.3.4 Stationary iterative methods

Let

$$Ax = b, \tag{20.64}$$

be the system to be solved, with $A \in \mathbb{R}^{N\times N}$, the matrix of coefficients and $x, b \in \mathbb{R}^N$ the solution and right-hand side, respectively. One of the simplest iterative methods is the Richardson method,

$$x_{k+1} = x_k + \omega r_k, \tag{20.65}$$

where

$$r_k = b - Ax_k = -A(x_k - x) = -A\,\Delta x_k, \tag{20.66}$$

is the residual for the iteration vector x_k, and ω is a relaxation parameter, to be determined, and $\Delta x_k = x_k - x$ is the error vector at iteration k. Subtracting x from both sides of equation (20.65), a recursive equation for the error vector can be obtained,

$$\Delta x_{k+1} = M \Delta x_k \tag{20.67}$$

where

$$M = I - \omega A \tag{20.68}$$

is the *amplification matrix*, i.e., the error vector is amplified by this matrix at each iteration. Of course, the idea is not to amplify the error, but rather to dampen it. Methods that can be represented as in equation (20.67) with a given constant matrix M are called *stationary*. This means that the next iteration vector x_{k+1} depends only on the previous vector x_k, not on the previous history $x_{k-1}, x_{k-2}, \ldots$

Note that if the dimension N of the problem were 1, then M would be a scalar, and the requirement for convergence would be simply

$$|M| = |1 - \omega A| < 1. \tag{20.69}$$

As $|M|$ gets lower, the error at each iteration is reduced more, and the convergence of the algorithm is faster. In this case, the value of $\omega = 1/A$, can be chosen and then the rate of convergence would be ideal, i.e., the error is reduced to zero in just one iteration.

For $N > 1$ the requirement for convergence is much the same but applied to the eigenvalues of A; i.e., the algorithm converges if all the amplification factors m_j are smaller than one in magnitude

$$|m_j| = |1 - \omega \lambda_j| < 1, \tag{20.70}$$

for all the eigenvalues λ_j of A. For the Poisson equation considered, the matrix A is symmetric and positive definite, and so there are N positive eigenvalues $\{\lambda_j\}_{j=1}^{N}$, which will be assumed ordered, i.e.,

$$\lambda_{max} = \lambda_1 \geq \lambda_2 \cdots \geq \lambda_N = \lambda_{\min} > 0 \tag{20.71}$$

To converge, all the amplification factors must satisfy eq. (20.67). For each eigenvalue, the amplification factor is a linear function starting with $m_j = 1$ at $\omega = 0$, see Figure 20.11 and decreasing its algebraic value with increasing ω. So that for ω small all the amplification factors are $|m_j| < 1$ and the iteration converges. For $\omega = 1/\lambda_{max}$ the amplification factor m_1 gets null. For larger values $m_1$becomes negative and starts increasing its value, eventually becoming $m_1 = -1$ at

$$\omega_{crit} = 2/\lambda_{max} \tag{20.72}$$

For values of $\omega > \omega_{crit}$ the amplification factor is $m_1 < -1$, i.e., $|m_1| > 1$, and the iteration diverges. So, ω_{crit} is the critical value beyond which iteration diverges.

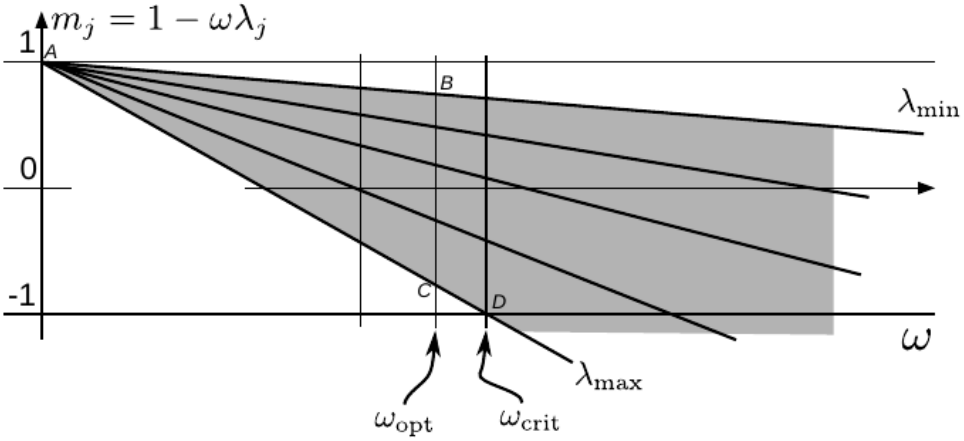

Figure 20.11 – Amplification factors for all eigenvalues as a function of ω.

For small ω the convergence is given by the largest amplification factor m_N corresponding to λ_{min} (segment A-B in Figure 20.11) and decreases from A to B. For larger ω the convergence is given by the largest amplification factor $m_1$corresponding to λ_{max} (segment C-D in Figure 20.11), increases from C to D. So, the minimum amplification is obtained at the relaxation factor ω_{opt}, where the convergence rate switches from B to C, i.e.

$$1 - \omega_{opt}\lambda_{max} = -\left(1 - \omega_{opt}\lambda_{min}\right) \tag{20.73}$$

$$\omega_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}} \tag{20.74}$$

and the optimal convergence rate is

$$\left|m_{\text{opt}}\right| = \left|1 - \omega_{opt}\lambda_{max}\right| = \left|1 - \frac{2\lambda_{max}}{\lambda_{min} + \lambda_{max}}\right| \tag{20.75}$$

$$\left|m_{\text{opt}}\right| = \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} = \frac{1 - \kappa^{-1}}{1 + \kappa^{-1}} \tag{20.76}$$

where

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1 \tag{20.77}$$

is the condition number of the matrix A.

In many cases, the condition number is very high, $\kappa \gg 1$, and then the amplification factor is very close to one. For $\kappa \gg 1$ the amplification factor can be approximated as

$$\left|m_{\text{opt}}\right| = 1 - 2\kappa^{-1} \tag{20.78}$$

The convergence of the iteration is controlled by $|m_{\text{opt}}|$, i.e., the error is multiplied at each iteration by a factor $|m_{\text{opt}}|$, so that after n iterations the residual is reduced by a factor

$$\|\Delta x_n\| = |m_{opt}|^n \|\Delta x_0\| \tag{20.79}$$

That means that the number of iterations needed to reduce the error by a factor of ε is

$$n_{iter} = \frac{\log(\varepsilon^{-1})}{\log(|m_{opt}|)} \cong \frac{\kappa\, log(\varepsilon^{-1})}{2} \tag{20.80}$$

The number of operations for the solution with the iterative method is the cost of the iteration times the number of iterations needed. In this case, the cost of the iteration is, see equations (20.65) and (20.66), a matrix-vector product operation Ax and a few vector (sum and scale) operations. All of them are $O(N)$, so the total number of operations for the solution of the linear system is

$$N_{ops}(\text{Richardson}) \sim N\, \kappa\, log(\varepsilon^{-1}) \tag{20.81}$$

Firstly, the computing time is proportional to $log(\varepsilon^{-1})$, i.e., to the number of orders of magnitude that the user wants to reduce the error. Many bounds on the convergence of iterative algorithms behave like that. Also, the computing time is proportional to the condition number of the matrix κ. Many other methods converge at a rate that depends on κ, for instance, it will be shown that the rate of convergence of the Conjugate Gradient Method is proportional to $\sqrt{\kappa}$.

The condition number grows with the refinement, and it depends on the operator, the discretization scheme, and the type of grid. For the Poisson equation described above, the condition number can be estimated as

$$\kappa = O(M^2) \tag{20.82}$$

i.e., quadratic on the number of steps in each spatial direction. Replacing in eq. (20.74), the following estimate is obtained

$$N_{ops}(\text{Richardson}) \sim N^2\, log(\varepsilon^{-1}) \tag{20.83}$$

Then, it is evident that Richardson has a computational time that is like the banded version of the direct method. Of course, there is a crucial difference between them. The direct method uses several operations which grow $O(N^2)$ to solve the equations in a finite number of steps. On the other hand, Richardson requires $O(N^2)$ to reduce the error by a given factor. This is reflected in the $\log(\varepsilon^{-1})$ factor. This can be seen as a disadvantage because the system of equations is not solved "exactly", but also it may be considered an advantage in certain situations because a reasonably good approximation can be obtained at a low cost. For instance, for a nonlinear problem, the linear solution is needed for each iteration of the outer loop. For the initial iterations, it is not needed a very low error; so,

the tolerance ε may be chosen high (for instance 10^{-3}) for the first iterations and then lowered to 10^{-6} for the final iterations.

Regarding memory storage, Richardson only needs to store the non-null coefficients, there is no "fill-in" as in the direct methods that require factorization. The number of non-null coefficients is a fixed number (5 for the example presented) per node, so that

$$N_{mem}(\text{Richardson}) = O(N) \tag{20.84}$$

Then, Richardson iteration, as compared with the direct-banded method, does not have a major difference in computing time but it represents a huge difference in memory storage. Consider for instance a mesh of $M = 1000$ steps per side and $N = 10^6$ unknowns, then, the memory requirements for dense matrix would be N_{mem} (dense) $= 10^{12}$ coefficients (8 TB in double precision), $N_{mem}(\text{banded}) = 2e9$ coefficients (16 GB), N_{mem} (Richardson) $= 5e6$ coefficients (40 MB).

Improved iterative methods that have significantly lower iteration count are presented in the following sections.

20.3.5 Krylov space-based iterative methods

Krylov space-based iterative methods exhibit a convergence far superior to the stationary methods. The formal definition is as follows. Given an initial vector x_0, the iteration vector x_k is defined as the vector that minimizes the norm of the error in the affine space $x_0 + \mathcal{K}_k$, where $\mathcal{K}_k$ is the Krylov subspace defined as

$$\mathcal{K}_k = \text{span}\{r_0, Ar_0, A^2r_0, \dots, A^{k-1}r_0\},\ k \geq 1 \tag{20.85}$$

This is known as the *minimization property* of the Krylov methods.

20.3.5.1 The Conjugate Gradient method

The norm to be minimized is the so-called *energy norm*, i.e.

$$\|x_k - x\|_A^2 = (x_k - x)^T \cdot A \cdot (x_k - x) \tag{20.86}$$

It can be shown that this is actually a norm because A is a symmetric positive definite matrix (SPD). For this kind of matrices, there is a very efficient algorithm to compute the sequence x_k, namely the Conjugate Gradient (CG) algorithm. CG does not need to store the whole basis of the Krylov space, but rather operates on a fixed set of 4 iteration vectors. So, the memory storage needed amounts to 4 vectors (of size N) and, of course, the matrix A in compact sparse form (5 non-null coefficients per row). Regarding computing time, the computational cost to compute a new vector in the sequence is one matrix-vector product and a fixed number (5 in total) of vector operations. Here, vector

operation stands for scalar products, sums of vectors, or scalar multiple of vectors, i.e. many operations that require $O(N)$ floating-point operations.

Regarding the convergence rate of CG, there is a bound given by

$$\|x_k - x\|_A \leq 2\, {m_{\mathrm{CG}}}^k \|x_0 - x\|_A \tag{20.87}$$

where the amplification factor m is given by

$$m_{\mathrm{CG}} = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \tag{20.88}$$

as the condition number is normally large ($\kappa \gg 1$) this can be simplified to

$$m_{\mathrm{CG}} \cong 1 - \frac{2}{\sqrt{\kappa}} \tag{20.89}$$

So, the number of iterations is now

$$n_{iter}(\mathrm{CG}) = \frac{\log(\varepsilon^{-1})}{\log(|m_{\mathrm{CG}}|)} \cong \frac{\sqrt{\kappa}\, log(\varepsilon^{-1})}{2} \tag{20.90}$$

which is much lower than Richardson's method.

In addition, the rate of convergence given by equations (20.87) and (20.88) is a lower bound; the actual rate of convergence is usually better. Firstly, consider the Krylov space by equation (20.85). As the iterations proceed, the number of vectors in the generating sequence $\{r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0\}$ grows. It can be shown that, for SPD matrices, all of them are linearly independent; then at iteration $k = N$ the Krylov space fills the whole $\mathbb{R}^N$ space, and then the iteration vector (by the minimization property) reaches the solution $x_k = x$. That means that CG is not an iterative method, but rather a direct method, i.e., it reaches the solution in a finite number of iterations. In practice, this condition is rarely reached for large systems (let's say $N > 10^5$) since iteration is usually stopped much earlier. However, it is an indication that the convergence rate for CG accelerates as the number of iterations proceeds.

Another difference between Krylov and stationary methods is that even if an estimation of the convergence rate based on the condition number κ can be found, the convergence of Krylov methods is much more complex than that. Its convergence depends not only on the condition number (i.e., on the maximum and minimum eigenvalues) but rather on the actual distribution of the eigenvalues on the real axis.

For instance, consider the two distributions of eigenvalues, as shown in Figure 20.12. Both have the same condition number $\kappa = \lambda_{\max}/\lambda_{\min}$ but, in the lower one, they are

clustered in two subintervals $[\lambda_{\min}, \lambda_{q}]$ and $[\lambda_{r}, \lambda_{\max}]$. Consequently, the convergence rate for CG is faster for the system corresponding to the clustered distribution.

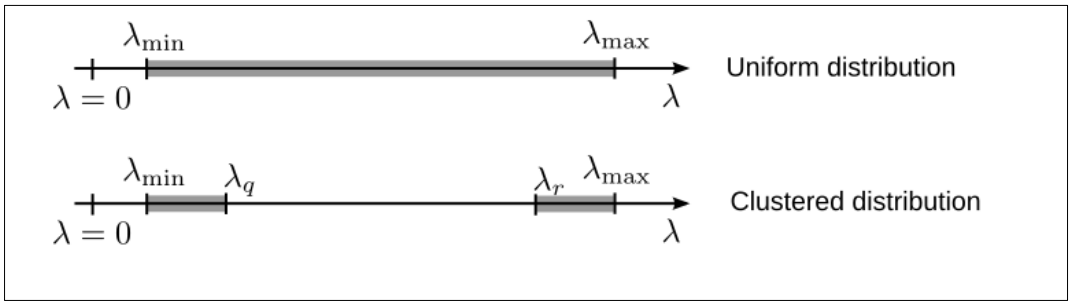

Figure 20.12 – Distribution of eigenvalues. In the upper distribution, the eigenvalues are uniformly distributed in $[\lambda_{\min}, \lambda_{\max}]$. In the lower distribution, they are concentrated in two clusters, $[\lambda_{\min}, \lambda_{q}]$ and $[\lambda_{r}, \lambda_{\max}]$.

Another advantage of CG versus the stationary methods is that CG is parameter-free, whereas, for instance, the convergence rate for Richardson depends on the choice of the relaxation parameter ω, whose optimal value depends on the eigenvalues of the operator. Of course, the eigenvalues of the operator are not known, so the relaxation parameter is chosen from estimates of them, and then, again, the convergence rate is lower than the optimal one.

20.3.5.2 Preconditioning

Suppose that a non-singular linear operator P is available, such that P is close to A^{-1}, then the following preconditioned system of equations can be obtained by left multiplying both sides of the equation by P

$$(PA)x = (Pb) \tag{20.91}$$

$$\tilde{A}x = \tilde{b} \tag{20.92}$$

which is equivalent to the original one, i.e., the solution of both systems x is the same. Since P is close to A^{-1}, it is expected that $\tilde{A} = PA$ is closer to the identity matrix, and then the condition number of $\tilde{A}$ is closer to unity, i.e., lower than the condition number of A. Since the condition number is lower, it is expected that the convergence rate (both for stationary and for Krylov methods) is better. Regarding the extra cost, note that the right-hand side $\tilde{b}$ can be computed only once, at the start of the iteration, so that the increment per iteration is a matrix-vector product by P, i.e., computing $y = Pz$, given the vector z. So, there is a trade-off between the reduction in the iteration count and the extra cost of applying the preconditioning. A given preconditioning may be very good at reducing the number of iterations, but the cost of applying the preconditioned counterbalances this reduction.

The simplest (and many times quite useful) preconditioning is Jacobi's

$$P_J = \text{diag}(A)^{-1} \tag{20.93}$$

i.e., to divide element-by-element by the diagonal part of A, specifically, the result of applying $y = P_J z$ is

$$y_j = \frac{z_j}{A_{jj}}, \quad \text{for } 1 \leq j \leq N \tag{20.94}$$

The cost of such operation is very low, just a vector element-wise division. Jacobi's preconditioning fixes many sources of bad conditioning, for instance, the diagonal terms, see equation (20.51), are proportional to $k/\Delta x^2$, which means that they have large variations over the domain if there is some type of mesh refinement or the thermal conductivity of the medium is not constant. Jacobi's preconditioning can fix these problems.

There are other pre-conditioning methods available; some of them may be problem-specific and others purely algebraic. One such preconditioning is the so-called *multi-grid* preconditioning. The multi-grid method is an iterative method based on the iteration on several grids of varying refinements, transferring the residuals and the corrections between the different meshes of different levels of refinement. Multi-grid is an iterative method by itself, that can be considered as an alternative to CG and others, but also can be combined with CG as a preconditioning.

The preconditioned system, eq. (20.92), deserves a comment. Note that even if P and A are SPD, the product of two SPD matrices is not necessarily SPD, so the preconditioning cannot be applied as simply as in (20.92). The correct way to apply the preconditioning is to solve

$$\tilde{A}\tilde{x} = \tilde{b} \tag{20.95}$$

With $\tilde{A} = P^{1/2} A P^{1/2}$, $\tilde{x} = P^{-1/2} x$, $\tilde{b} = P^{1/2} b$, where $P^{1/2}$ is the square root (in the matrix sense) of A. With this definition now $\tilde{A}$ is indeed SPD so that CG converges. But, of course, $P^{1/2}$ is very hard and expensive to compute. However, there is a modification of CG, known as *Preconditioned Conjugate Gradient (PCG),* that produces the iteration over eq. (20.95) without having to compute $P^{1/2}$.

20.3.5.3 Matrix free implementation

Note that in contraposition to direct methods, to solve the linear system with iterative methods like Richardson or CG the matrix does not need to be assembled physically in memory, nor even as a sparse matrix. They just need a routine to implement the matrix-vector product. This is called a *matrix-free* implementation.

Many pre-conditioning ways are possible in the matrix-free form. Most iterative methods can be implemented in matrix-free form, and libraries that implement those iterative methods implement them, usually, in *functional form*, i.e., the user has to implement a function that, given a vector x, produces the matrix-vector product $y = Ax$ (or either $y = Px$ for the preconditioning).

20.3.5.4 Non-SPD matrices. CG over normal equations

If the matrix system A is not SPD then CG cannot be directly applied. One possible strategy is to transform the system into an SPD one. A simple possibility is to multiply both sides of the system (20.64) by the transpose of A,

$$\tilde{A}x = \tilde{b} \tag{20.96}$$

with

$$\tilde{A} = A^T A, \quad \tilde{b} = A^T b \tag{20.97}$$

It is easy to show that $\tilde{A}$ is SPD. The problem with this strategy is that the condition number of $\tilde{A}$ is probably close to the square of the condition number of A, $\kappa(\tilde{A}) \sim (\kappa(A))^2$. As $\kappa(A)$ is usually high, $\kappa(\tilde{A})$ is much higher and then it would be impractical to solve.

There is a variation of this strategy

$$\tilde{A}\tilde{x} = b \tag{20.98}$$

with this transformation,

$$\tilde{A} = AA^T, \quad x = A^T\tilde{x} \tag{20.99}$$

As before, it can be easily shown that AA^T is SPD. Both strategies are very similar, and the main disadvantage is also the high condition number of the modified system. The first strategy (equation 20.97) is called *Conjugate Gradient on the Normal equations to minimize the Residual (CGNR)*, and eq. (20.99) is called *Conjugate Gradient on the Normal equations to minimize the Error (CGNE)*. The term *normal equations* refer to the fact that the equations obtained are the same as the so-called normal equations that are obtained when solving the system in the sense of Least Squares.

Another disadvantage of CGNE and CGNR is that they need not only the matrix-vector product $y = Ax$, but also the product with the transpose, i.e., $y = A^T x$. If the matrix is built and stored (for instance in some kind of sparse form), then both operations can be easily implemented. But this may be difficult to implement in a pure matrix-free form.

Both CGNE and CGNR can be combined with suitable pre-conditioning ways.

20.3.5.5 GMRES

Another strategy to cope with non-SPD matrices is to build the Krylov space, as defined by eq. (20.85) and proceed with the minimization property to obtain the next iteration. The disadvantage of this is the cost of storing the Krylov space (which is of size kN, i.e., k vectors of size N, at iteration k) and also at each iteration the minimization property needs an operation to orthogonalize the new residual $A^k r_0$ with all the vectors in the space, which amounts to computational cost $O(kN)$. So, both storage and operation count grow linearly with the iterations, in contraposition with Richardson and CG and its variants, where the storage cost and the computational cost per iteration are both $O(N)$. For instance, for an SPD matrix, the iterations generated by GMRES and CG are the same but, of course, CG has a much lower computational cost and much lower storage needs.

As the cost per iteration grows with iterations, it becomes impractical to iterate GMRES right to convergence. Not only do the iterations become increasingly slower, but also the stored Krylov space grows in storage, and eventually it simply does not fit in computer memory. So, the iteration is *restarted*, i.e., at a certain iteration n GMRES is started again, using x_n, as the new starting point. This process is repeated until convergence, i.e., each

n iterations GMRES is restarted. The restart count n is a parameter to be determined by the user, perhaps by trial and error. An upper bound to n is certainly given by the memory available.

20.3.5.6 Other methods for non-SPD matrices

For mildly non-SPD matrices, i.e., matrices that are non-SPD but not too far, some methods are somewhat intermediate between CG and GMRES, i.e., they have a computational cost for iteration $O(N)$ but are still applicable even if the matrix is not SPD. One of the most popular is Bi-CG; it is based on keeping the orthogonality of the vector iterations with respect to the Krylov space, but not necessarily imposing the minimization property. In that way, the Bi-CG algorithm needs only to store four vectors and two matrix-vector products. For SPD matrices, Bi-CG is equivalent to CG, at a cost of roughly double in memory storage and operation count.

Bi-CG may suffer from *breakdown*, i.e., during iteration at a certain point the iteration cannot proceed. This is more likely to occur as the matrix is less SPD, i.e., the difference with respect to an SPD matrix is higher. When this happens, the user can restart Bi-CG from the last iteration or just switch to GMRES, which is more robust.

There are improvements and variations on Bi-CG, namely *Bi-CGSTAB* (for *Bi-CG Stabilized*), and others (*Conjugate Gradient Squared CGS*, and *Transpose Free Quasi-Minimal Residual TFQMR*). All of them have $O(N)$ extra storage and $O(N)$ operation count per iteration. Bi-CGSTAB and TFQMR do not need matrix-vector operations involving the transpose matrix.

20.3.5.7 Pure three-diagonal systems

The concepts developed in section 20.2.1 for the discretization of 1D evolution problems can be extended for other dimensionalities. If d is the spatial dimension ($d = 2$ for 2D and so on) then, we have the total number of equations $N = M^d$, and the bandwidth is $a = M^{d-1}$ and the equation (20.58) transforms into

$$N_{ops}(\text{banded}) = O(a^2 N) \sim O(N^{1+2(d-1)/d}) \tag{20.100}$$

i.e., $N_{ops} = O(N)$ in 1D, $O(N^2)$ in 2D, and $O(N^{2.33})$ in 3D.

The 1D case is especially interesting because many TH codes are based on the 1D discretization of each pipe in a pipe network. In the 1D case, the bandwidth of the matrix is $a = 1$, and the cost of both solution and memory storage is $O(N)$, which is the lowest possible cost (up to a multiplicative constant) for any solution algorithm (direct or iterative), since, at least, any such algorithm would need to perform at least one arithmetic operation on each of the coefficients and unknowns. There are specialized variants of the Gauss elimination that may be more efficient than the standard version (by a multiplicative constant), for instance, the Thomas algorithm.

20.3.5.8 Network three-diagonal systems

Consider now a closed loop, i.e., a pipe where the outlet of the pipe is connected to its inlet. In such a case the matrix would look like this

$$\mathbf{K} = \begin{bmatrix} k_{11} & k_{12} & 0 & 0 & \cdots & \cdots & \cdots & k_{1N} \\ k_{21} & k_{22} & k_{23} & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & k_{32} & k_{33} & k_{34} & 0 & \cdots & \cdots & \cdots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ k_{N1} & 0 & 0 & 0 & 0 & 0 & k_{N,N-1} & k_{NN} \end{bmatrix} \tag{20.101}$$

The matrix is three-diagonal, except for the coefficients k_{1N} and k_{N1}. So, the bandwidth is $a = N$ and the cost would be very high ($O(N^3)$ with a dense direct method). However, a simple algebraic trick can still recover the $O(N)$ cost.

Note that the system can be recast in the following form, by separating explicitly the last row and column,

$$\mathbf{A}\tilde{\boldsymbol{\phi}} + \phi_N \mathbf{w} = \tilde{\mathbf{f}} \tag{20.102}$$

$$k_{N,1}\phi_1 + k_{N,N-1}k_{N,N-1}\phi_{N-1} + k_{N,N}\phi_N = f_N \tag{20.103}$$

where $\mathbf{A}$ is the upper-left block of size $(N-1) \times (N-1)$, i.e.

$$\mathbf{A} = \begin{bmatrix} k_{11} & k_{12} & 0 & 0 & \cdots & \cdots & \cdots \\ k_{21} & k_{22} & k_{23} & 0 & \cdots & \cdots & \cdots \\ 0 & k_{32} & k_{33} & k_{34} & 0 & \cdots & \cdots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & 0 & k_{N-1,N-2} & k_{N-1,N-1} \end{bmatrix} \tag{1.104}$$

and $\mathbf{w}$ is a vector composed of the first $N-1$ coefficients of the last column

$$\mathbf{w} = \begin{bmatrix} k_{1N} \\ k_{2N} \\ 0 \\ \ddots \\ k_{N-1,N} \end{bmatrix} \tag{20.105}$$

and

$$\tilde{\mathbf{f}} = \begin{bmatrix} f_1 \\ f_2 \\ 0 \\ \ddots \\ f_{N-1} \end{bmatrix} \tag{20.106}$$

Note that, from equation (20.102), the solution can be put as

$$\tilde{\boldsymbol{\phi}} = (\mathbf{A}^{-1}\tilde{\mathbf{f}}) - \phi_N (\mathbf{A}^{-1}\mathbf{w}) = \mathbf{x} - \phi_N \mathbf{y} \tag{20.107}$$

Both auxiliary vectors **x** and **y** can be computed, and then, $\tilde{\boldsymbol{\phi}}$ is a linear combination in terms of ϕ_N

$$\phi_1 = x_1 - \phi_N\, y_1, \qquad \phi_{N-1} = x_{N-1} - \phi_N\, y_{N-1} \tag{20.108}$$

Now, these expressions can be inserted in equation (20.103) and a linear scalar equation on ϕ_N is obtained. This equation is solved and then replaced in equation (20.102) and $\tilde{\boldsymbol{\phi}}$ (and then $\boldsymbol{\phi}$) is obtained.

The full algorithm is as follows,

1. Compute auxiliary vectors **x** and **y** from $\mathbf{A}\boldsymbol{x} = \tilde{\mathbf{f}}, \quad \mathbf{A}\mathbf{y} = \mathbf{w}.$
2. Obtain ϕ_N by replacing eq. (20.108) in eq. (20.103) and solving the resulting linear scalar equation for ϕ_N.
3. Finally, obtain $\tilde{\boldsymbol{\phi}}$ replacing ϕ_N in eq. (20.102).

Note that the cost of the algorithm is mainly the solution of two linear three-diagonal systems so that it is $O(N)$. In addition, the coefficient matrix is the same in both cases, so it must be factored in only once.

This procedure can be extended to networks of pipes. The variables that have coefficients out of the three-diagonal part (like ϕ_N in this example) are called *network variables*, and their equations are called *network equations*. Leaving free the network variables, a three-diagonal system is obtained. The system must be solved for each of the m network variables, and then replacing the resulting expression in the network equations, a linear system of size $m \times m$ is obtained. The total cost of the algorithm is $O(Nm + m^3)$. This is very convenient, provided that m is low.

20.3.5.9 Solution of elliptic equations using ADI methods

In the previous sections, the solution of elliptic problems was presented in terms of iterative methods. In this way, it is possible to solve a relatively large number of simultaneous equations with minimum use of computer memory arising from a detailed discretization of the space domain. Considering equation (20.44) and imposing boundary conditions, it is possible to obtain the solution for steady-state using the transient method as a pseudo iterative one. When the steady-state is reached, the solution corresponds to the elliptic problem. In this case, the intermediate computed solutions are not relevant. When $\Delta x = \Delta y$, the ratio $\Delta t/\Delta x^2 = \Delta t/y^2$ may be considered an iteration parameter. The convergence to steady-state may be optimized by using repeated sequences of values. The details of the determination of these values may be obtained considering Peaceman and Rachford, (1955).

20.3.6 Parallel implementation of direct and iterative methods

Nowadays computers usually have many cores, even personal computers, and notebooks. Computing servers have dozens of cores, and HPC equipment may allow the user to run on several nodes with dozens of cores, i.e., hundreds of cores or more. Most of the thermo-hydraulics codes can exploit this computing power. There are two almost exclusive computing technologies to use parallelism, namely Message Passing Interface (MPI) and

OpenMP. OpenMP can only be used with multi-core processors, i.e., so-called *shared memory* architectures, while MPI can be used in both shared memory and distributed *memory* (i.e., computing nodes in an HPC cluster). Most commercial and open-source TH codes support either one or both technologies

Parallelism can be used both in building the linear system and in the solution of the resulting system for each time step. Direct methods cannot efficiently be parallelized, so iterative methods are almost normally used in parallel architectures. It is conceptually easy to parallelize iterative methods; vectors are distributed among the processors, and each processor computes the contribution of some part of the matrix-vector products. Of course, for a given problem with a given computational mesh, if the number of processors is increased, at a certain point there is more communication between the processors than computations to be performed so that the parallelization does not *scale* beyond that number of processors. The user should determine this limit by trial and error, and as a rule, it can be expressed as the number of unknowns per core. Typically, the scaling limit is within $O(10^4 - 10^5)$ unknowns per core.

20.4 The solution of Hyperbolic PDEs

20.4.1 First order equations, scalar transport

Consider the momentum equation for the one-dimensional form of the Navier-Stokes equations, in its simplest form,

$$\rho\left(\frac{\partial u}{\partial t} + u\,\frac{\partial u}{\partial x}\right) + \frac{\partial p}{\partial x} = \mu\frac{\partial^2 u}{\partial x^2} \tag{20.109}$$

where ρ is density, u velocity, p pressure, and μ viscosity. We can easily identify the temporal, advective, and pressure term on the left-hand side, and the viscous term on the right-hand side.

To have a single equation with a single variable, we drop the pressure term and consider constant properties ρ and μ,

$$\frac{\partial u}{\partial t} + u\,\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{20.110}$$

This is the advection-diffusion equation. $\nu = \mu/\rho$ has units of m^2/s and is called the kinematic viscosity of the fluid.

Here, the equation has been derived for the velocity u but is valid as the general transport equation for scalar variables, like temperature, species concentration, and many others. In those cases, we will denote u the scalar field being transported and a the transport velocity, i.e.

$$\frac{\partial u}{\partial t} + a\,\frac{\partial u}{\partial x} = \alpha\frac{\partial^2 u}{\partial x^2} \tag{20.110}$$

α is the diffusivity of the scalar field in the fluid. In the transport of scalar fields, other than velocity, the process is called "advection", instead of "convection". That is, the term convection is then reserved for the case when the transported quantity is the same as the velocity.

When the diffusive term is null (i.e. $\alpha = 0$) we talk of "pure advection" and the equation is hyperbolic. When the diffusive term is not null the equation is parabolic.

In what follows, we will focus on the convective term $a\ \partial u/\partial x$ and its discretization. This term poses several problems. On one hand, if the first derivative is discretized with centered three-point schemes, then non-physical oscillations arise in the solution when the convective term dominates. On the other hand, in the case of convection, as posed in eq. (20.110), the term is non-linear, while the other terms are linear. This also poses numerical problems and instabilities.

20.4.2 The method of characteristics

Consider now the case of pure advection with constant velocity a, i.e

$$\frac{\partial u}{\partial t} + a\ \frac{\partial u}{\partial x} = 0 \tag{20.111}$$

With initial condition

$$u(x, 0) = \bar{u}(x) \tag{20.112}$$

Where $\bar{u}(x)$ is the initial value for the scalar u. We can easily see that the solution is simply a shift of the initial condition by at at time t, i.e.

$$u(x, t) = \bar{u}(x - at) \tag{20.113}$$

Because

$$\frac{\partial u}{\partial t} + a\ \frac{\partial u}{\partial x} = -a\bar{u}' + a\bar{u}' = 0 \tag{20.114}$$

This means that the value of u is preserved along the lines $x = at$. These lines are called "characteristics" and can be visualized as the trajectory of particles that move with the velocity of the fluid. The solution tells that the value of the scalar is preserved along these trajectories. In the case of advection-diffusion with a finite (but small) diffusivity, we expect that the value will not be exactly preserved along the characteristics, but rather there will be some smearing due to the diffusion.

If the velocity field is not constant, i.e., $a = a(x, t)$, then the characteristic starting at x_p at $t = 0$ is defined as an ordinary differential equation, see Figure 20.13,

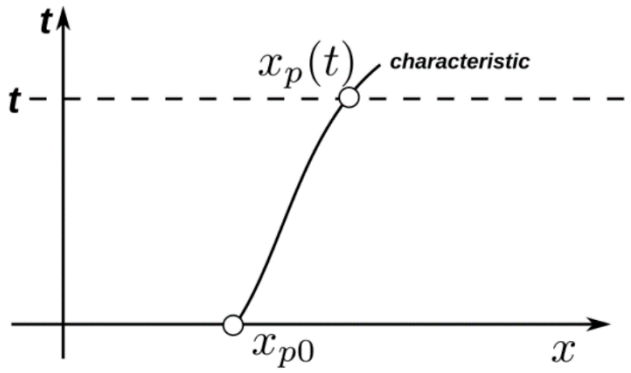

Figure 20.13 – Characteristic trajectory for scalar advection with variable velocity.

$$\frac{dx_p}{dt} = a\big(x_p(t), t\big), \qquad x_p(t=0) = x_{p0} \tag{20.115}$$

This property of the hyperbolic equations induces a numerical method, which is then called the *"Method of Characteristics"* (MOC). Given the nodal values u_j^n of u at nodes $x_j = j\Delta x$ at time $t^n = n\Delta t$ (point *A*, see Figure 20.14), then we can obtain the values at t^{n+1} by tracing back the characteristic that passes by the target point $x = x_j, t = t^{n+1}$ and find its position at $t = t^n$, i.e. solving for the characteristics equation (20.115) backward in time from $t = t^{n+1}$ to $t = t^n$ (point A' in Figure 20.14).

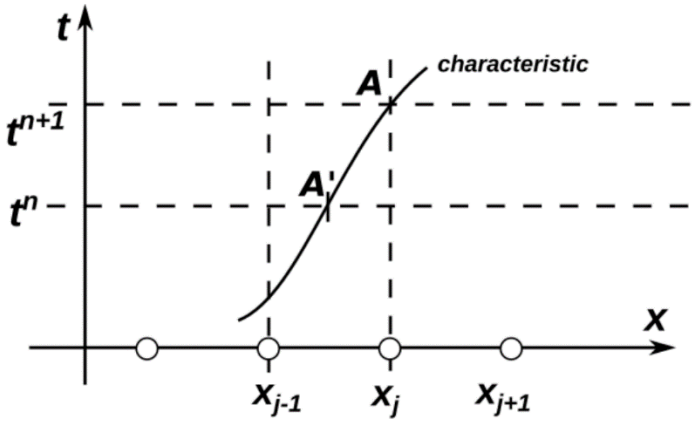

Figure 20.14 – Method of characteristics.

Once the point A' is determined the new nodal values are determined as

$$u_j^{n+1} = u^n(A') \tag{20.116}$$

If the velocity field is $a > 0$ then the source point A' falls to the left of x_j. If the velocity were negative, it will fall to the right. For small time steps, the source point will fall in

one of the cells adjacent to x_j, for larger time steps the source point may fall farther, two or more cells from the target point. The number of cells traversed by the particle (i.e., the characteristic) in the time step Δt is simply the CFL number.

It remains to explain how to solve numerically the ODEs for the characteristics, and then how to interpolate the discrete values of the field at t^n to the source point A'. There is a variety of possibilities to do this, giving rise to methods of different precision. Note that if the velocity field is constant and the time step is adjusted so that $\Delta x = a\Delta t$ (i.e., CFL=1) then the point A' is coincident with one grid point at the time t^n, and so the scheme amounts simply to a shift of one cell on the values

$$u_j^{n+1} = u_{j-1}^n \tag{20.117}$$

Then there is no loss of information, and the solution is exact. The same happens in general if $a\Delta t = n\Delta x$, with n integer (i.e., CFL an integer number). On the other hand, if the CFL number is $C = 0.5$, then A' will fall half in between $x_j$and x_{j-1}. If linear interpolation is used, we will have

$$u_j^{n+1} = \frac{u_{j-1}^n + u_j^n}{2} \tag{20.118}$$

Then, if we have a very sharp step-like initial field

$$u_j^0 = [0,0,0\ldots,0,1,1,1\ldots], \quad \text{at } t = 0, \tag{20.119}$$

then we will have

$$u_j^1 = [0,0,0\ldots,0.5,0.5,1,1\ldots], \quad \text{at } t = \Delta t, \tag{20.120}$$

$$u_j^2 = [0,0,0\ldots,0.25,0.5,0.75,1,1,1\ldots], \quad \text{at } t = 2\Delta t, \tag{20.121}$$

and so on, with consequent strong smearing of the step. However, the BFECC method, Dupont and Liu, (2003), for Back-and-Forth Error Compensation and Correction can fix this problem in a very simple and efficient way.

Note that, unlike most other numerical methods, MOC doesn't need the solution of a system of equations. It is based mostly on the tracking of particles. Many methods related to or derived from MOC are known also as Lagrangian (or, better, semi-Lagrangian) because they are based on the Lagrangian point of view of mechanics (i.e., following material points) in contrast to the Eulerian point of view.

20.4.3 Numerical approximations to the solution of hyperbolic PDE.

The extension of the pure advection equation (20.111) to many coupled fields is

$$\frac{\partial \boldsymbol{U}}{\partial t} + \boldsymbol{A}\,\frac{\partial \boldsymbol{U}}{\partial x} = 0 \tag{20.122}$$

where now $\boldsymbol{U}$ is an array of m fields and A is a matrix of size $m \times m$, that may be, in general, a function of position x and time t. In the nonlinear case, it may be also a function of the flow field $\boldsymbol{U}$. Many important physical problems can be put in this form, for instance, gas dynamics, shallow water, Maxwell equations for electromagnetics, or the wave equation. The system is *hyperbolic* if $\boldsymbol{A}$ is diagonalizable with real eigenvalues, i.e., there exists a set of m linearly independent eigenvectors $\boldsymbol{s}_j$and real eigenvalues λ_j such that,

$$\boldsymbol{A}\boldsymbol{s}_{\boldsymbol{j}} = \lambda_j \boldsymbol{s}_{\boldsymbol{j}}, \qquad j = 1, \cdots, m \tag{20.123}$$

or, in matrix form

$$\boldsymbol{A}\boldsymbol{S} = \boldsymbol{S}\boldsymbol{\Lambda} \tag{20.124}$$

with $\boldsymbol{\Lambda}$ a diagonal matrix with real diagonal entries λ_j, and $\boldsymbol{S}$ real and non-singular. Note that this is the case if $\boldsymbol{A}$ is real and symmetric, but this is not required. As the $\boldsymbol{s}_j$ are linearly independent, we can decompose any given state $\boldsymbol{U}$ as a linear combination of them, i.e.,

$$\boldsymbol{U} = \sum_{\boldsymbol{k}} v_k \boldsymbol{s}_k \tag{20.125}$$

Replacing in the evolution equation (20.122) we get,

$$\frac{\partial}{\partial t}\left(\sum_{\boldsymbol{k}} v_k \boldsymbol{s}_k\right) + \boldsymbol{A}\,\frac{\partial}{\partial x}\left(\sum_{\boldsymbol{k}} v_k \boldsymbol{s}_k\right) = 0 \tag{20.126}$$

$$\left(\sum_{\boldsymbol{k}} \boldsymbol{s}_k \frac{\partial v_k}{\partial t}\right) + \left(\sum_{\boldsymbol{k}} \frac{\partial v_k}{\partial x} \boldsymbol{A}\boldsymbol{s}_k \quad\right) = 0 \tag{20.127}$$

$$\sum_{\boldsymbol{k}} \boldsymbol{s}_k \left(\frac{\partial v_k}{\partial t} + \lambda_k \frac{\partial v_k}{\partial x}\right) = 0 \tag{20.128}$$

But, as the $\boldsymbol{s}_{\boldsymbol{k}}$ are linearly independent,

$$\frac{\partial v_k}{\partial t} + \lambda_k \frac{\partial v_k}{\partial x} = 0, \quad k = 1, \cdots, m \tag{20.129}$$

This means that the original system of equations is decoupled in m independent scalar equations, with $\lambda_k$being the velocity for the k-th characteristic component of the field v_k. The sign of the eigenvalues λ_k represents the sense of propagation for that field. Fields with $\lambda_k > 0$ propagate from left to right and vice versa. The method of characteristics as was described could be used for each of the characteristic fields v_k. If the λ_k have a different sign (some positive and some negative) then the updated value of the vector field u_j^{n+1} will depend on the value of cells at both sides of the position x_j.

Many numerical methods that are developed for the scalar case ($m = 1$) can be extended easily to the hyperbolic system case ($m > 1$) by simply replacing the velocity a by the advective Jacobian $\boldsymbol{A}$. The same applies to stability criterion, precision, and other criteria. As an example, we will focus on the well-known Lax-Wendroff scheme.

The Lax-Wendroff scheme can be developed as a direct application of the operator expansion as explained in section 20.2.1. Formally we can solve equation (20.111) in operator form as

$$u(t + \Delta t) = \exp\left(-\Delta t\, a\, \frac{\partial}{\partial x}\right)\ u(t) \tag{20.130}$$

Performing a Taylor expansion to second-order we get,

$$u(t + \Delta t) = \left(1 - \Delta t\, a\, \frac{\partial}{\partial x} + \frac{\Delta t^2\, a^2}{2}\, \frac{\partial^2}{\partial x^2} + O(\Delta t^3)\right)\ u(t) \tag{20.131}$$

Now, using second-order centered finite differences for the spatial derivatives the Lax-Wendroff numerical scheme is obtained

$$u_j^{n+1} = u_j^n - \Delta t\, a\, \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} + \frac{\Delta t^2\, a^2}{2}\ \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} \tag{20.132}$$

which can be written as

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a\, \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} = \frac{\Delta t\, a^2}{2}\ \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} \tag{20.133}$$

This scheme is second-order in space and time.

Note that the right-hand side term has the aspect of a diffusion term and comes consistently from the temporal expansion. It has not to be confused with an artificial diffusion term. Note that if this term were not present, clearly the scheme would be first-

order since it is based on a forward first-order approximation of the temporal term. The term on the right-hand side not only does add stability to the numerical scheme but also is fundamental for second-order precision.

The extension for hyperbolic systems ($m > 1$) is trivial,

$$\frac{\boldsymbol{U}_j^{n+1} - \boldsymbol{U}_j^n}{\Delta t} + \boldsymbol{A}\,\frac{\boldsymbol{U}_{j+1}^n - \boldsymbol{U}_{j-1}^n}{2\Delta x} = \frac{\Delta t\,\boldsymbol{A}^2}{2}\;\frac{\boldsymbol{U}_{j+1}^n - 2\boldsymbol{U}_j^n + \boldsymbol{U}_{j-1}^n}{\Delta x^2} \tag{20.134}$$

Consider now the stability of the scheme in the scalar case. Following a standard von Neumann analysis, the nodal values are replaced by a plane wave

$$u_j^n = \mu^n\, e^{ikx_j} \tag{20.135}$$

with

$$|k| < \frac{\pi}{\Delta x} \tag{20.136}$$

the *wavenumber* and μ (in general a complex number) the *amplification factor*. In all these expressions, the imaginary part is neglected, i.e., we assume that just the real part of the expressions is relevant. Replacing the plane wave in the evolution equation (20.133) we obtain the amplification factor in terms of the wavenumber,

$$\frac{\mu - 1}{\Delta t} + a\,\frac{e^{ikh} - e^{-ikh}}{2\Delta x} = \frac{\Delta t\,a^2}{2}\;\frac{e^{ikh} - 2 + e^{-ikh}}{\Delta x^2} \tag{20.137}$$

and, after some algebra

$$\mu = 1 - iC\,\sin(\mathrm{kh}) - 2C^2\sin^2\left(\frac{\mathrm{kh}}{2}\right) \tag{20.138}$$

where $C = a\Delta t/\Delta x$ is the CFL number. In Figure 20.15, μ is a plot in the complex plane for $-\pi \leq k\Delta x \leq$and several C from 0.1 to 1.1. It is clear from Figure 20.15 (and can be shown analytically) that the scheme is stable i.e., $|\mu| \leq 1$, for $0 < C \leq 1$. There exists also an implicit variant of the Lax-Wendroff scheme, which is unconditionally stable, i.e., it is stable for all $C > 0$.

The von Neumann analysis for the hyperbolic system case ($m > 1$) starts proposing a plane-wave solution of the form

$$\boldsymbol{U}_j^n = \bar{\boldsymbol{U}}\, e^{ikx_j} \tag{20.139}$$

Replacing this expression in the evolution equation for hyperbolic systems, eq. (20.134), we obtain that the solution at the next step $n + 1$ is also a plane wave of the form

$$\boldsymbol{U}_j^{n+1} = \boldsymbol{G}\bar{\boldsymbol{U}}\, e^{ikx_j} \tag{20.140}$$

With

$$\boldsymbol{G} = \boldsymbol{I} - i\,\frac{\Delta t}{\Delta x}\boldsymbol{A}\,\sin(\mathrm{kh}) - 2\,\frac{\Delta t^2}{\Delta x^2}\boldsymbol{A^2}\,\sin^2\left(\frac{\mathrm{kh}}{2}\right) \tag{20.141}$$

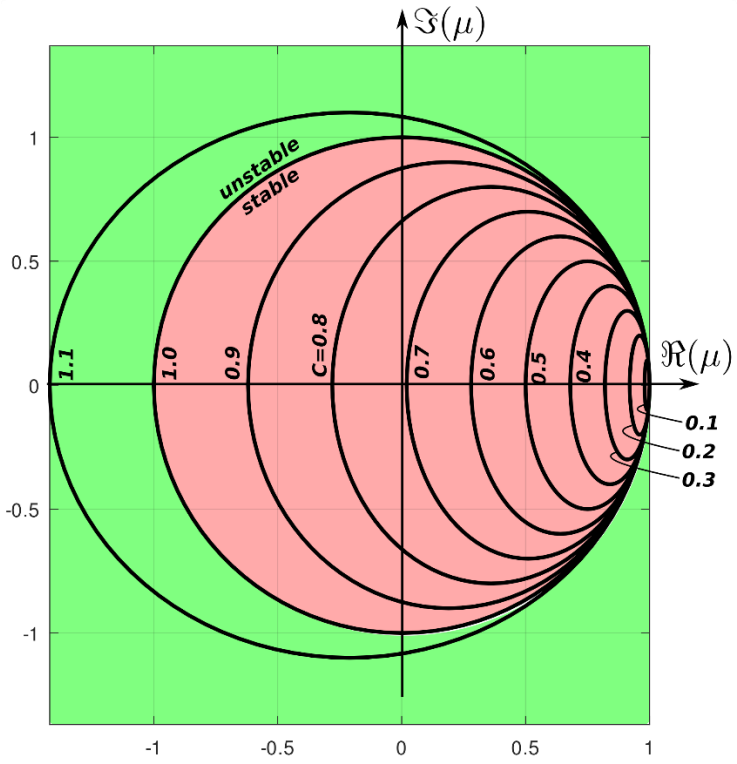

Figure 20.15 – Stability of the Lax-Wendroff scheme.

Note that, for hyperbolic systems, the matrix $\boldsymbol{G}$ plays the role of the amplification factor, and the CFL number C is replaced by the matrix $(\boldsymbol{A}\,\Delta t/\Delta x)$. The scheme is stable provided that the norm (in fact the spectral radius) of the matrix is less than one, that means

$$|\mu_k| \leq 1 \tag{20.142}$$

for all eigenvalues of $\boldsymbol{G}$. As $\boldsymbol{G}$ is a polynomial (of second order) on $\boldsymbol{A}$ ($\boldsymbol{G} = p(\boldsymbol{A})$), it results that the eigenvalues of $\boldsymbol{G}$ are the polynomial evaluated on the eigenvalues of $\boldsymbol{A}$, i.e.

$$\mu_{\boldsymbol{k}} = 1 - i\,\frac{\Delta t\lambda_k}{\Delta x}\,\sin(\mathrm{kh}) - 2\,\frac{\Delta t^2\lambda_k^2}{\Delta x^2}\,\sin^2\left(\frac{\mathrm{kh}}{2}\right) \tag{20.143}$$

It results then that the stability condition is that the CFL number for each of the characteristics must be less than one, i.e.

$$\frac{\Delta t|\lambda_k|}{\Delta x} \leq 1, \qquad \text{for all } k, \tag{20.144}$$

i.e.,

$$\Delta t \leq \frac{\Delta x}{\max(\lambda_k)} \tag{20.145}$$

In other words, this means that the maximum characteristic velocity of the system dictates the stability.

20.5 The validity of computer codes solutions

The determination of the validity of a computer code solution is one of the key issues concerning the application of TH codes. This results from the combination of many factors, which received a lot of attention in the last three decades when CFD techniques became widely available. The goal was to determine the uncertainty associated with the numerical predictions. In so doing, physical model qualification, computer code verification, validation, uncertainty determination of code predictions, may be considered together in the subject of code results qualification and the whole activity conforms to what is known as code Verification and Validation (V&V), Chapters 13 and 22 of the book. There is another aspect that must be considered: user qualification, adding a computational non-specific limitation. The combination of all these aspects justifies the complexity mentioned.

It is not possible in the limited context of a chapter paragraph even to consider a detailed list of references on the subject (see Chapters 13 and 22). It would suffice to mention that the application of TH codes must be done in the domain of the intended applications of a code, considering the limitations detailed in the code assessment manuals.

To guide the reader on the subject, it may be mentioned that several professional societies contributed standards or recommendations related to the use of computational codes that may be applied vis-à-vis to TH and CFD codes. Among the hundreds of authors, the names of W.L. Oberkampf, T.G. Trucano, P.H. Roache, and F.S. D'Auria are outstanding in the field of V&V. In what follows, it is assumed that the reader is aware of the different professional associations and their role.

Peters et al., (2011), reviewed several (different) methods for codes V&V and compared their approaches. The procedures were those available from ASME, (2008); AIAA, (1998); ASTM, (2005); Boyack et al., (1990a), and OECD/NEA/CSNI, (2007b).

The compared approaches go well beyond code verification that may be considered the first step in the V&V activity leading to determining the validity of codes solutions. From all this work it becomes evident that the validity of a computer code solution implies also considering the proper determination of the physical model representing the physical reality. The AIAA Guide, AIAA, 1998, is perhaps the clearer and the more exhaustive, giving definitions and necessary information to get an overview of the subject. The reader is referred to this reference without a lack of recommendation of the remaining ones.

The Method of Manufactured Solutions (or MMS) has been systematized by Roache, (2009), even when more elementary procedures are possible. Patrick Roache also stated that the verification step is more a question of software and mathematics rather than

physical. The idea behind the MMS is to introduce a known arbitrary function in the model equation, establishing and setting the values on the boundaries as boundary conditions and computing the solution. This procedure allows, for a given discretization, computing solution the error in the solution at the points in the grid.

The complexity of the validity of codes solutions also implies the analysis of the validity of the model governing TH equations, their regularization, and aspects related to the convergence of the computed solution to the (unknown) theoretical one. These questions deserved detailed consideration from the FONESYS network, e.g. Ahn et al., (2017).

To illustrate further the aspects that may be considered, the following list adapted and appended from OECD/NEA/CSNI, (2007b), and Ferreri and Ambrosini, (1998), is included in the following. Please note that the checklist initially applies to CFD but almost all the items have a correlate in the application of TH codes. The appended information on user effects is directly related to TH codes as applied to NPP Safety Analysis. The reader must consider that no checklist may be considered definitive and keeps evolving.

Initial Preparation

- *Produce a clearly written problem description, specifying the system and scenario requiring analysis, and clearly listing study objectives*
- *Assemble a panel of experts and go through the Process Identification and Ranking Table (PIRT) process based upon the problem description*
- *Do special phenomena such as containment wall condensation require the addition of models to a standard CFD code or the use of a special purpose CFD package?*
- *With knowledge of the problem and physical processes select an appropriate CFD code and if necessary, develop enhancements*
- *Does the problem require full CFD or are classic TH codes adequate?*
- *Is coupling required between a CFD and a TH code to supply boundary conditions to the CFD?*

Geometry Generation

- *Is the coordinate system correct?*
- *Are the units correct?*
- *Have any substantial modifications been made to the geometry?*
- *Is the geometry complete?*
- *Are there oversimplifications due to symmetry assumptions, etc.?*
- *Are inlet, outlet, symmetry, and cyclic boundary condition regions located correctly?*
- *Selection of Physical Models*
- *Develop a basic understanding of the prevalent physical phenomena and flow fields (part of the PIRT process)*
- *Select the appropriate level of turbulence representation (RANS, T-RANS, LES, hybrid approach)*
- *For RANS or T-RANS select an appropriate statistical model for turbulence*
- *Either resolve the wall boundary layer or choose a wall function model*
- *Establish boundary conditions consistent with your choice of turbulence model*

Grid Generation

- *Are the grid angles larger than 20° and less than 160°?*
- *Are the ratios of adjacent volumes less than 2?*
- *Are the aspect ratios below the values given in the solver manual (typically 10-50)?*
- *Is the grid scalable?*

- *Are grid nodes concentrated in areas of foreseeable physical significance?*
- *Does the grid contain non-matching grid interfaces in critical regions?*
- *Is the grid compatible with the physical models (turbulence model, wall treatments, etc.)?*

Numerical Methods

- *Generally, avoid the use of first-order upwind spatial discretization and first-order implicit time integration schemes.*
- *If first-order methods are used, compare the numerical diffusion coefficient to an estimate of the turbulent diffusion coefficient at several locations in your mesh*
- *When using LES, select a higher-order central difference method, preferably 4th order*

Verification

- *Check for round-off errors*
- *Check for errors associated with the selection of iteration convergence criteria*
- *Check for errors associated with the discretization of space and time*
- *Follow procedures to limit and locate user errors including:*
  - *i. selection of a high-quality user interface to the CFD or TH code; and,*
  - *ii. use of quality assurance practices.*

Validation

- *Follow a tiered approach comparing first to separate effects experiments (unit problems) and working up through complete system experiments*
- *Where possible use repeat experiments to help quantify the experimental error*
- *Using guidance from the PIRT process, select target variables and metrics for agreement between calculation and experiment*
- *Characterize experimental uncertainty for all target variables, distinguishing between random and systematic (bias) contributions to the uncertainty*
- *If sufficient computer resources are available, perform uncertainty analysis on the simulation, to place bounds on results, and to cross-check the initial PIRT assumptions about the relative importance of physical phenomena*

Related to User Effects

Basic user skills on

- *Fluid Dynamics (1-p / 2-p)*
- *Numerical methods: finite volumes and finite differences*
- *Computational modeling uncertainty evaluation. This includes V&V*
- *Data structure QA*
- *Plant layout and related systems*
- *Full awareness of systems code models and capabilities*

Basic user attitudes

- *Continuous disposition to check his/her ability as a modeler against benchmark analytical and experimental scenarios*
- *Being non-confident of standard rules as applying to every situation*
- *Keeping confident on the necessary steps to perform a qualified analysis, disregarding non-affordable deadlines*
- *Avoiding non-essential analysis (i.e., knowing when to stop searching)*

Basic user needs

- *Being defined as a developer and qualifier of nodalizations or/and the person in charge of the analysis of scenarios*

- *Respect for his/her patient, time-consuming work for data preparation and qualification (a typical, simple NPP nodalization implies nearly 3000 lines of 12 items each)*
- *Time to seek for convergence of results in each scenario*
- *Enough time and collaboration for unforeseen aspects of his/her results*
- *Collaboration for nodalization QA and exploitation regarding plant scenarios*
- *Respect for his/her interest to check standard modeling criteria for new scenarios leading the code to its boundaries of validity (these will be of obvious help to understand "strange" results)*
- *Interacting with colleagues in appropriate forums (users clubs, code developer meetings, and so on)*
- *Reporting to skilled officers.*

To summarize this section, it may be stated that relevant bibliography has been considered to illustrate a subject (V&V) that is *per se* an independent activity related to the validity of computed solutions. The user of TH codes must be aware of these efforts and fulfill the conditions mentioned as basic skills. The user must consider that the more advanced and detailed the code is, the need for basic skills including numerical techniques and Fluid Dynamics increases.

20.6 Automatic computation of sensitivities to parameters in TH codes

The sensitivity of the results of a computational model in TH to system parameters may be key to the evaluation of their general validity because permits the ranking of importance that must be attributed to closing correlations and coefficients. There are, at least, two ways to proceed: a) brute force, i.e., introducing suitable small variations of the parameters and calculating the derivative of some figure of merit to the parameter by discrete calculation, and b) generating an alternative version of the code that calculates the solution *and* the derivatives simultaneously. The first option is simple and does not deserve a further explanation, but the computational cost may be prohibitive. In what follows the second option will be explored in a limited context in TH, namely the boundaries of neutral flow stability in one-dimensional natural circulation flows. This choice is not arbitrary but is based on the simple cases that may be considered.

A methodology of analysis based on numerical discretization of partial differential equations governing fluid-dynamic problems is also useful to get information on the capabilities of numerical methods in accurately predicting stability. A key point of this methodology is the evaluation of the derivatives of the Jacobian matrices of the algebraic relationships that define the numerical method and the related boundary conditions. It is in this perspective that the use of automatic code differentiation tools plays an interesting role.

The pioneering work on Automatic Differentiation by Christian Bischof and collaborators at the Argonne National Laboratory in the nineties, Bischof et al., (1996), and ANL, (1999), permitted starting the automatic computation of sensitivities to parameters in conservation equations, among many other applications. The procedure to be considered in what follows corresponds to discretize first and differentiate later. Therefore, the sensitivity to physical and discretization parameters is analyzed using a tool for the

automatic differentiation of FORTRAN codes. This is so because FORTRAN is the programming language usually used for programming TH system codes. At least this is true for legacy codes. ADIFOR (meaning Automatic DIfferentiation of FORtran) is the adopted tool that allows for evaluating the derivatives of model variables with respect to model parameters. Bischof et al., (1996), and ANL, (1999), also mention different implementations for other programming languages.

More recently, Bischof et al., (2007), showed the application of ADIFOR to a large-scale industrial CFD code like FLUENT, showing that this is not just an academic tool but one amenable to industrial applications. To get an idea it is enough to mention that the code generated is nearly 550,000 source lines long. The definitive version of the code was about 700,000 FORTRAN lines. Bischof et al., (2007), give a brief account of the significance of code differentiation and establish the difference of ADIFOR with other symbolic differentiation tools. Results of the final code as applied to the Navier Stokes equations are shown for a backward step flow and a rotating flow including closure correlations for turbulence modeling.

To illustrate the application of ADIFOR to fluid flow, an early application is discussed hereafter, Ferreri and Ambrosini, (2001). A necessary recapitulation consists in mentioning here that ADIFOR is a pre-processor code that, given a FORTRAN 77 code that computes a function, automatically generates another, augmented, FORTRAN 77 code. It must be considered that any code may be put in the form of a function, simply by introducing a call to a main routine after setting parameter values. The latter computes the function and the derivatives with respect to a list of variables. The user must specify the list of dependent and independent variables. After generating the augmented code that calculates the specified derivatives via ADIFOR, the user must provide a new driving FORTRAN 77 code that considers the new set of variables. Many references document the accuracy of the derivatives calculated in this way. Bischof et al., (2007), point out, this is not the only way to apply ADIFOR, nor ADIFOR is the only tool. However, in the experience of one author of the present chapter (JCF), small problems may be tackled efficiently.

The analysis will be applied in the limited context of the determination of the neutral stability boundary in natural circulation in a simple system, amenable to analytical steady-state treatment. Results have been obtained using implicit coupling of the momentum and energy equations and the forward time, upwind-space finite-difference method (FTUS) for the energy equation. The consequences of using this approximation on the results are well known, i.e., an $O(1,1)$ solution in the space and time increments. However, the combined effects of these errors and those of the variables coupling and their quantification are not simple, at least in the field of the stability limits of natural circulation systems.

To fix ideas, the following approximation, Ambrosini and Ferreri, (1998), is adopted for the energy equation in the loop:

$T_i^{n+1} = (1 - C_{OU})\ T_i^n + C_{OU}\ T_{i-1}^n \qquad (i=2, \ldots, N-1)$ (20.146)

$$T_N^{n+1} = \left(1 - C_{OU} - \frac{\Delta t}{\Delta s} F_{NOD}(q^n)\right) T_N^n + C_{OU} T_{N-1}^n - \frac{\Delta t}{\Delta s} F_{NOD}(q^n)\ T_1^{n+1} \quad (20.147)$$

$$T_N^{n+1} = -T_1^{n+1}$$

where T is the temperature, q is the volumetric flow rate, n is the time step number such that $t^{n+1} = t^n + \Delta t$, C_{OU} is the Courant number $C_{OU} = \frac{q \cdot \Delta t}{\Delta s}$ and $\Delta s = \frac{1}{N-1}$ is the space increment, with N being the number of nodes. The function $F_{NOD}(q)$ is a source-sink heat transfer multiplier, introduced to provide steady-state convergence of eq. (20.147) to the exact steady-state solution. It is a non-linear function of the variables.

The momentum equation is integrated along the loop and is discretized in time as follows:

$$q^{n+1} = q^n + \left(\alpha \cdot \Delta s \sum_{i=1}^{N-1} \frac{T_i^{n+1} + T_{i+1}^{n+1}}{2} - \varepsilon\ |q^n|^{\xi-1} \cdot q^{n+1}\right)\ \Delta t \quad (20.148)$$

In equation (20.148), α and ε measure, respectively, the driving buoyancy and the friction in the loop.

The finite-difference procedure defined by equations (20.147) and (20.148) may be written as

$$\boldsymbol{F}(\boldsymbol{y}^{n+1}, \boldsymbol{y}^n, \boldsymbol{p}) = 0 \quad (20.149)$$

where $\boldsymbol{y}$ is the vector of nodal unknowns, p is the vector of system parameters (C_{OU} and ξ in this case). Steady-state conditions for a given set of parameters satisfy the relationship:

$$\boldsymbol{F}(\boldsymbol{y}^s, \boldsymbol{y}^s, \boldsymbol{p}) = 0 \quad (20.150)$$

By assuming small perturbations around the steady-state:

$$\boldsymbol{y}^n = \boldsymbol{y}^s + \delta \boldsymbol{y}^n \qquad \boldsymbol{y}^{n+1} = \boldsymbol{y}^s + \delta \boldsymbol{y}^{n+1} \quad (20.151)$$

Introducing this in eq. (20.149) and expanding up to the 1st order:

$$\boldsymbol{F}(\boldsymbol{y}^{n+1},\boldsymbol{y}^{n},\boldsymbol{p}) = \boldsymbol{F}(\boldsymbol{y}^{s},\boldsymbol{y}^{s},\boldsymbol{p}) + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}^{n}}\Big|^{s} \delta\boldsymbol{y}^{n} + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}^{n+1}}\Big|^{s} \delta\boldsymbol{y}^{n+1} + \ldots = 0 \tag{20.152}$$

Using equations (20.150) and (20.152) leads to

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}^{n}}\Big|^{s} \delta\boldsymbol{y}^{n} + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}^{n+1}}\Big|^{s} \delta\boldsymbol{y}^{n+1} = 0 \tag{20.153}$$

and finally:

$$\delta\boldsymbol{y}^{n+1} = -\left[\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}^{n+1}}\Big|^{s}\right]^{-1} \cdot \left[\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}^{n}}\Big|^{s}\right] \cdot \delta\boldsymbol{y}^{n} \tag{20.154}$$

Then, in the adopted notation:

$$\boldsymbol{A} = -\left[\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}^{n+1}}\Big|^{s}\right]^{-1} \cdot \left[\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}^{n}}\Big|^{s}\right] \tag{20.155}$$

The value of $\Delta\rho$ given by:

$$\Delta\rho = \rho(\boldsymbol{A}) - 1 \tag{20.156}$$

where $\rho(\boldsymbol{A})$ is the spectral radius of the matrix $\boldsymbol{A}$. As told before, $\Delta\rho$ is used as a measure of the margin in excess to neutral stability and to quantify the degree of damping or amplification. It takes negative values for stable conditions and positive ones for unstable conditions. Ferreri and Ambrosini, (2001), obtained the sensitivity of $\Delta\rho$ by calculating its derivative concerning $\mathbf{p}$ using the original code that allowed the calculation of $\Delta\rho$ itself. This implied obtaining the derivatives of the classical EISPACK path for the eigenvalues and eigenvectors of a general matrix. It, in turn, imposed a non-trivial computational effort when *N* was several hundred.

Here, the calculation of the sensitivity of $\Delta\rho$ to parameters is based on the following algorithm:

Define A *and* A^T*, respectively as the matrix relating perturbations at time steps n and n+1 and its transpose. This matrix may also be specified using* ***ADIFOR****. However, second-order derivation to get derivatives of* ***A*** *with respect to* p_i *is beyond the objective of this paper. Consequently,* ***A*** *will be specified analytically.*

Use of ***ADIFOR*** *to automatically calculate the derivative of* ***A*** *with respect to system parameters*

Calculation of the eigenvalues of ***A*** *and its eigenvectors,* $\boldsymbol{\lambda_i}$***, x.***

Calculation of the eigenvectors of $\boldsymbol{A^T}$***, v.***

Determination of i*, the index of the eigenvalue that corresponds to the spectral radius of* ***A*** *and the corresponding eigenvectors of* ***A*** *and* $\boldsymbol{A^T}$*.*

Calculation of the sensitivity of λ_i *with respect to system parameters.*

Calculation of the spectral radius sensitivity with respect to system parameters.

The parameter to be considered is ξ, which defines the dependence of the friction term on the flow type.

The results are as follows. Figure 20.16 shows the stability map in the plane of α and ε, the system physical parameters. They measure, as told before, respectively the buoyancy and friction resistance in the loop. This map has been obtained using 31 nodes (dimensionless space increment $\Delta s = 1/30$), Courant number $C_{OU} = 0.8$, and the exponent of flow rate in the friction term of momentum equation given by $\xi = 1.75$.

The interest in the analysis to follow is showing how the solution depends on ξ (≡ 2-b), where b is the exponent in the friction law). This has been quantified by calculating the derivative $\partial \Delta\rho / \partial \xi$ and is shown in Figure 20.17. Further considerations and previous analyses may be found in Ferreri and Ambrosini, (2001).

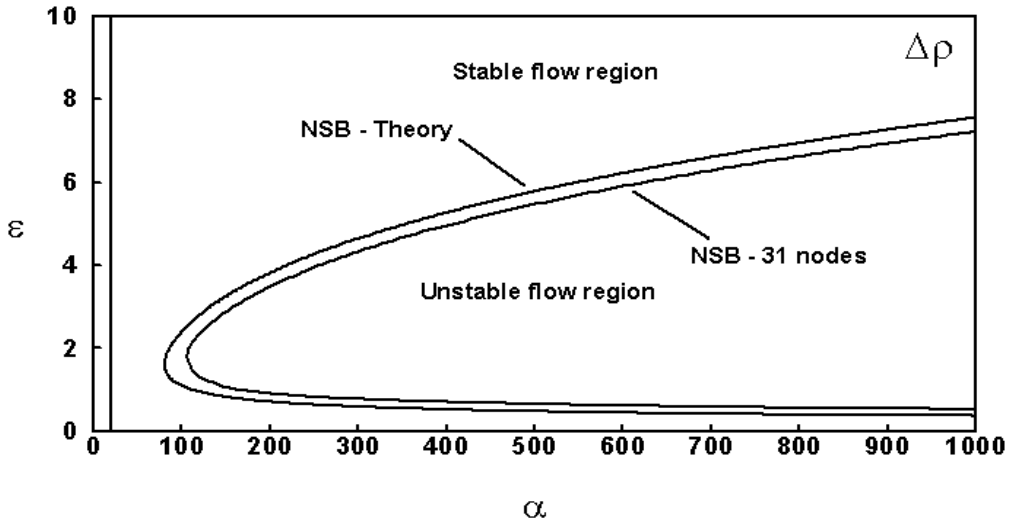

Figure 20.16 – Map of Δρ for the first-order, explicit momentum, implicit temperature coupling, C_{OU}=0.8 and ξ=1.75, Δs = 1/30.

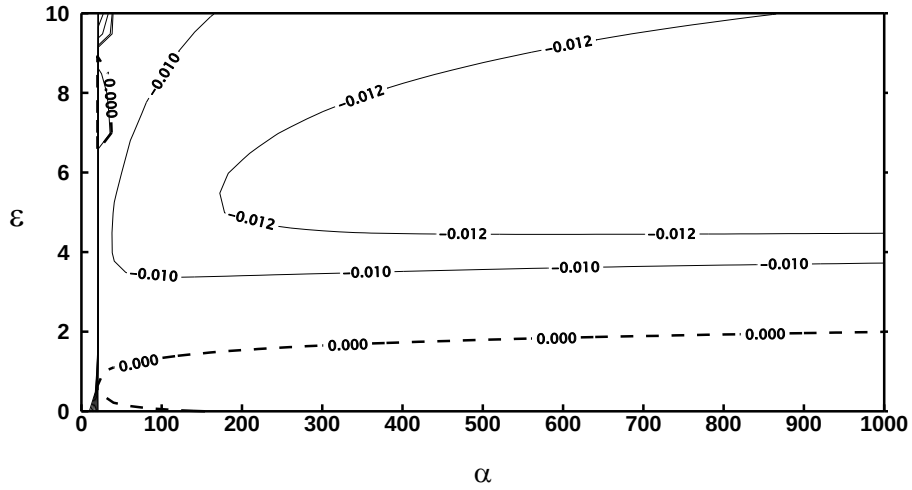

Figure 20.17 – Sensitivity map of $\Delta\rho$ to the friction law exponent $(\partial\Delta\rho/\partial\xi)_0$ for the first-order, explicit momentum, and implicit temperature coupling. [COU = 0.8, ξ = 1.75 and Δs = 1/30].

To summarize, we discussed and exemplified a procedure for computing automatically sensitivities of code results to system parameters. The limited context gives an idea of the importance of the technique and is affordable at the expense of some effort of development.

Acknowledgment

The overview of methods for the numerical solution of the governing equations (or the content of the chapter) benefits from earlier work by JCF in collaboration with Prof. Walter Ambrosini, University of Pisa, who is gratefully acknowledged. Work started in the early 90's because of the generous hosting of Prof. Francesco Saverio D'Auria in his group. The friendly atmosphere of many other colleagues was also encouraging.

REFERENCES

Ahn, S.H., Aksan, N., Austregesilo, H., Bestion, D., Chung, B.D., Coscarelli, E., D'Auria, F., Emonot, P., Gandrille, J-L., Sauvage, J-Y., Hanninen, M., Horvatovic, I., Kim, K. D., Kovtonyuk, A., Lutsanych, S., Petruzzi, A., 2017. Hyperbolicity and numerics in SYS-TH codes: The FONESYS point of view, J. Nuclear Engineering and Design, Vol 322, pp 227-239

ASME, 2008. Standard for Verification and Validation in Computational Fluid Dynamics and Heat Transfer, ASME V&V 20, American Society of Mechanical Engineers, New York, NY, USA

AIAA, 1998. Guide for the Verification and Validation of Computational Fluid Dynamics Simulations AIAA Standards. American Institute of Aeronautics and Astronautics, AIAA G-077, Reston, VA, USA. ISBN 978-1-56347-354-8.

Ambrosini, W., Ferreri, J.C., 1998. The effect of Truncation Error on Numerical Predictions of Stability in a Natural Circulation Single-Phase Loop, Nuclear Eng. and Design, 183, pp 53-76.

ANL, 1999. Computational Differentiation Technical Reports, Argonne National Laboratory, Chicago. Ill, USA. http://www.mcs.anl.gov/autodiff/tech_reports.html

ASME, 2008. Standard for Verification and Validation in Computational Fluid Dynamics and Heat Transfer, ASME V&V 20, American Society of Mechanical Engineers, New York, NY, USA.

ASTM, 2005. Standard Guide for Evaluating the Predictive Capability of Deterministic Fire Models. American Society for Testing and Materials, ASTM std. E1355-05a, see also American National Standards Institute (ANSI), Washington DC, USA.

Bischof, C.H., Carle, A., Khademi, P., Mauer A., 1996. ADIFOR 2.0 Automatic differentiation of Fortran 77 programs, IEEE Computational Science and Eng., 3(3), pp 18-32.

Bischof, C.H., Bücker, H.M., Rasch, A., Slusanschi, E., Lang, B., 2007. Automatic Differentiation of the General-Purpose Computational Fluid Dynamics Package FLUENT. ASME J. of Fluids Engineering, 129, 5, pp 1-17.

Boyack, B.E., Catton, I., Duffey R.B., Griffith, P., Katsma, K.R., Lellouche, G.S., Levy, S., Rohatgi, U.S., Wilson, G.E., Wulff, W., Zuber, N., 1990a. Quantifying Reactor Safety Margins. Part I: An Overview of the Code Scaling, Applicability and Uncertainty Evaluation Methodology, J. Nuclear Engineering and Design, Vol 119, 1, pp 1-15.

Douglas, J. Rachford, H.H., Jr, 1956. On the numerical solution of heat conduction problems in two and three space variables. Trans. Am. Math. Soc., 82(2), p 421.

Dupont, T. F., Liu, Y., 2003. Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. J. of Computational Physics, 190(1), pp 311-324.

Ferreri, J.C., 1985. A note on the Advection-Diffusion Equation, Int. J. Num. Methods, Fluids, 5, pp 593-596.

Ferreri, J.C., Ambrosini, W., 1998, User qualification and NPP safety analysis: a case dependent issue or a discipline of learning? IAEA Spec. Meet. on User Qualification for and User Effect on Accident Analysis for Nuclear Power Plants, 31 Aug. - 2 Sept., Vienna, Austria.

Ferreri, J.C., Ambrosini, W., 2001. Calculation of Sensitivity to Parameters in Single-Phase Natural Circulation Using ADIFOR, Int. J. Computational Fluid Dynamics, 16 (4), pp 277-281.

Ferreri, J.C., Ambrosini, W., 2002. On the analysis of thermal-fluid-dynamic instabilities via numerical discretization of conservation equations. Nuclear Eng. Des., 215, pp 153–170.

Fletcher, C.A.J., 1991. Computational Techniques for Fluid Dynamics, Vol 1, Fundamental and General Techniques and, Vol 2, Specific techniques for Different Flow Categories, 2nd Ed, Springer Verlag, Berlin Germany.

Hirt, C.W., 1968. Heuristic Stability Theory for Finite-Difference Equations, J. Comp. Physics, 2, pp 339-355.

Gresho P., Lee R., 1981. Don´t suppress the wiggles, they are telling you something, Computers and Fluids, 12, pp 223-231.

Knoll, D., Morel, J., Margolin, L., Shashkov, M., 2005. Physically Motivated Discretization Methods A Strategy for Increased Predictiveness, Los Alamos Science, 29, pp 188-212. Los Alamos, NM, USA.

Laney, C.B., 1998. Computational Gas dynamics, Cambridge University Press, Cambridge, UK.
Marchuk, G.I., 1975. Methods of Numerical Mathematics, Springer Verlag, Berlin, Germany.

LANSCE, 1981. Los Alamos Science, 2, 2, summer/fall, Los Alamos Nat. Lab., NM, USA.

Mahaffy, J.H., 1982. A stability enhancing two-step method for fluid flow calculations. J. Comput. Phys., 46, pp 329–341.

Marchuk, G.I., 1975. Methods of Numerical Mathematics, Springer Verlag, Berlin, Germany.

Mitchell, A., Griffiths, D., 1980. The Finite-Difference Method in Partial Differential Equation, John Wiley & Sons, New York, NY, USA.

OECD/NEA/CSNI*, 2007b. Best Practice Guidelines for the use of CFD in Nuclear Reactor Safety Applications, NEA/CSNI/R(2007)5, Paris, France.

Peaceman, D.W., Rachford, H.H., Jr, 1955. The numerical solution of parabolic and elliptic differential equations. J. Soc. Ind. Appl. Math., 3, pp 28-41.

Peters, S., Tschaepe, L., Zhang, B., Ruggles, A., 2011. V&V Methodology Comparisons: AIAA G-77, (1998), ASME V&V 20, (2009), ASTM E1355-05a, (2005), NEA/CSNI/R(2007), and NRC CSAU, (1988). In Proc. 14th Int. Top. Meet. on Nuclear Reactor Thermal hydraulics, NURETH-14, Toronto, Ontario, Canada, Sept. 25-30.

Roache, P.J., 1972. Computational Fluid Dynamics, Hermosa Publishers, Albuquerque-Socorro, NM, USA, pp 1-434.

Roache, P.J., 2009. Fundamentals of Verification and Validation, Hermosa Publishers, Albuquerque-Socorro, NM, USA.

Scannapieco, E., Harlow, F.H., 1995. Introduction to Finite-Difference Methods for Numerical Fluid Dynamics, LA 12984, Los Alamos, NM, USA.

Trefethen, L.N., 1982, Group Velocity in Finite Difference Schemes, Society for Industrial and Applied Mathematics (SIAM) Review, 24, pp 113-135.

Von Neumann, J., Richtmyer, R.D., 1950. A Method for the Numerical Calculation of Hydrodynamic Shocks, Journal of Physics, 21, pp 232-237.

About the AUTHORS of Chapter 20

Juan Carlos Ferreri was born on February 11, 1944. He graduated as Aeronautical Engineer at La Plata University in 1967 and has dedicated his professional career to the field of computational fluid mechanics and heat and mass transfer. For thirty years, he has devoted his work to numerical modeling in Nuclear Safety and Engineering at the Nuclear Regulatory Authority of Argentina (ARN). He is currently: a) Fellow Member at the National Academy of Sciences of Buenos Aires –ANCBA (since 2009) and Past President, from 2017 to 2021; b) Retired Researcher at the National Research Council (CONICET). He has received the Senior 2004 Award for Research, Professional and Teaching Achievements in Argentina from the Argentinean Association for Computational Mechanics (AMCA). He has been a Member and President of the Argentine Committee for Heat and Mass Transfer (CONICET). He lectured on numerical methods as applied to natural circulation in nuclear installations and heat transfer effects in soils for archaeometry purposes in Argentina and abroad (Italy, USA, China, France, and Perú). In the period 1995-2010, he developed research activities in collaboration with researchers at the University of Pisa. He has published more than one hundred-ten papers in his fields of expertise and has delivered tens of seminars and invited conferences in Argentina and abroad.

Mario A. Storti was born on August 10, 1960. He is Senior Researcher of CONICET at CIMEC (Center for Computational Methods Research, Santa Fe, Argentina). He graduated in Physics from the Balseiro Institute, National University of Cuyo, Bariloche in 1983 and in 1990 he obtained his Ph.D. at UN Litoral (Santa Fe, Arg), where he is currently a professor in undergraduate and graduate programs. He received the 2012 Senior Scientist Award from the Argentine Association of Computational Mechanics (AMCA). He has published more than 80 articles in indexed journals, has been an advisor to seven doctoral theses, organized international conferences and edited seven volumes of the Computational Mechanics (AMCA) series. He has done numerous transfer activities in the field of Computational Fluid Dynamics for industrial processes. Prof. Storti currently participates in the BIOTRAFO project of the European Union.