Chapter 2

Numerical solution of fluid flow equations

2.1 Overview

In section 1.2, a continuous model for fluid-flow problems was presented. In this section, different techniques for its numerical integration are outlined. The model problem is a set of strongly coupled, non-linear partial derivative equations. The following operations are to be done to achieve a numerical solution (not necessarily in this order):

- From Partial Derivative Equations (PDE) to discrete algebraic equations: spatial and temporal discretization (sections 2.3 and 2.4).
- From a non-linear problem to a sequence of linear problems: linearization (section 2.5).
- From coupled unknowns (u_1, u_2, u_3, p, T) to uncoupled unknowns: uncoupling (section 2.6).
- From (huge) linear equation systems to their solution: linear solvers (section 2.7).

This list is a bit artificial: the total separation of certain algorithms in different stages is not always possible. As an example, consider geometric non-linear multigrid (section 3.1.3), that groups linearization, spatial discretization and solution of the linear algebraic equations. But the simplification is useful to clarify the goals of the different stages. For instance, a better linearization process will reduce the number of linear problems to be solved at each time step but not necessarily the time to solve each of them. A more efficient procedure to solve the segregated linear equations will be of little help if the bottleneck for a particular problem is the coupling between the unknowns.

The aim of the following sections is to present the computational implications, for sequential and parallel computers, of the different operations. An exhaustive discussion of many methods for incompressible flows can be found in [49].

2.2 Convection-Diffusion equation

Instead of considering the spatial and temporal discretization of each of the NS equations (1.1-1.5) separately, it is useful to express each of them as a particular case of generic *convection-diffusion* equation. In three-dimensional Cartesian co-ordinates the convection-diffusion equation has the following expression:

$$\overbrace{\underbrace{S}_{source\ term}}^{generation\ of\ \phi} = \overbrace{\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{transient\ term}}^{accumulation\ of\ \phi} \overbrace{+\underbrace{\sum_{i=1}^{3}\left[\frac{\partial}{\partial x_i}(\rho u_i\phi)\right]}_{convection\ term} + \underbrace{\sum_{i=1}^{3}\left[\frac{\partial}{\partial x_i}\left(-\Gamma\frac{\partial\phi}{\partial x_i}\right)\right]}_{diffusion\ term}}^{transport\ of\ \phi} \tag{2.1}$$

The use of a generic convection-diffusion equation is not only useful to simplify the discretizaton, but also provides information about the physical meaning of the terms in the model. It is used to model the transport of a generic physic magnitude (momentum, energy or mass depending on the equation) in a continuous fluid medium with a velocity field $\mathbf{u} = (u_1, u_2, u_3)$, that in this point is assumed to be known. The changes in this generic magnitude are described in terms of ϕ, the unknown of the equation. It can stand for a variety of different quantities such as mass fraction or a velocity component. As an example, temperature (or enthalpy) are the variables used to describe the energy in each point of the fluid.

None of the terms of the equation has a meaning in absence of the others. However, if they could be isolated, their role in the transport and generation of ϕ in a infinitesimally small control volume would be (still using the energy equation as an example):

- The **source term** describes the generation of energy in the control volume.
- The **transient term** describes the energy accumulated.
- The **convection term** describes the flux of energy leaving the control volume due to the velocity $\mathbf{u}$ of the fluid medium.
- The **diffusion term** describes the energy flux leaving the control volume due to molecular diffusion. This process transports energy from points of higher energy to points of lower energy concentration, independently of the velocity field $\mathbf{u}$.

Thus, equation (2.1) is simply a balance between the generation, accumulation and transport of a generic variable ϕ. The transport can be due to convection (associated to a macroscopic movement of the medium) or to diffusion (due to molecular diffusion).

The terms ϕ, Γ and S (generic unknown, generalized diffusion coefficient and source term) are to be changed depending on the equation, according to the following table:

Equation	ϕ	Γ	S
Mass conservation	1	0	0
x_1 **momentum conservation**	u_1	μ	$-\frac{\partial p}{\partial x_1}$
x_2 **momentum conservation**	u_2	μ	$-\frac{\partial p}{\partial x_2}$
x_3 **momentum conservation**	u_3	μ	$-\frac{\partial p}{\partial x_3} + \rho g \beta (T - T_0)$
Energy conservation	T	$\frac{k}{c_p}$	0

Table 2.1: Terms of the generic convection-diffusion equation.

Remarks:

- The same pattern can be used for other transport equations. As an example, consider concentration transport equation. For this case, $\phi = C$, $\Gamma = \rho D$ and $S = 0$ if the flow is inert (non-reactive), where C is the mass fraction or concentration and D is the diffusion coefficient.
- The source term is also typically used to model other processes, apart from generation or destruction. For instance, in porous media flow modelization [67], it is used to express the non-linear dispersion transport due to the microscopic interaction between the flow and the solid obstacles.

- If $\mathbf{u}$, Γ and S are independent of the transported variable ϕ, equation (2.1) is a linear PDE. This is not the case of any of the equations (1.2-1.5) :
  - Momentum equations are clearly non-linear due to convective terms that for the case of momentum equations ($\phi = u_j$) become $(\rho u_j \phi) = \frac{\partial}{\partial x_i}(\rho u_i u_j)$. We may think of momentum equations as convection-diffusion equations in which the components of the velocity themselves are being transported. S is also non-linear, as pressure gradient terms depend on $\mathbf{u}$ and buoyancy terms on T (coupled with $\mathbf{u}$).
  - Energy equation is also non linear because the velocity field depends on T.

The use of a generic expression for all the scalar equations, is a first step towards the numerical solution of the set. However, due to a number of reasons, such as the use of staggered grids, in many CFD codes the discretizaton of each of the equations is implemented in a separate function. Additionally, the mass conservation equation, (a first order and non-transient equation) is quite artificially expressed in terms of the general convection diffusion equation (second order and transient). The use of the convection-diffusion pattern, although useful, does not solve the computational difficulties associated with it (section 1.2).

2.3 Spatial discretization. Finite control volume method.

The discretization technique approximates the continuous solution of the PDE, member of a space of infinite dimension, with a function characterized by a finite (discrete) number of parameters that describe it. The parameters are usually, but not necessarily, the values of the unknown function at different spatial points. Only the finite control volume method will be considered in this work. Other methods used in CFD are: finite differences, finite elements, spectral/pseudo-spectral methods and spectral elements.

They can be classified into two main categories [69]: *local* and *global*, depending on whether the parameters governing the discrete function describe local or global properties. Finite differences, finite control volumes and finite elements are local discretization methods while the spectral method is a global method: the unknown functions are expressed as a combination of known functions taking non-vanishing values over all the computational domain. Thus, a change in one of the parameters changes the approximation in all the domain.

The finite control volume method (FCV) will be used to describe the discretization process. In spite of the advances achieved by other discretization methods, FCV is still the standard approach, used by most commercial CFD codes but also for certain high performance applications such as DNS [70].

Many descriptions of the FCV method applied to CFD are available. For instance, [57] gives a clear description of the discretization technique using first order schemes, while [71, 72] provide a self-contained discussion to the extension to second order schemes using the deferred correction procedure and the normalized variable and space formulation (NVSF). Here it will only be concisely described to present its computational implications. The method, based on a conservation of fluxes of the different primitive variables, has a direct physical interpretation [57].

An *implicit* variant of the method will be used for the description. As discussed in section 2.2, Navier-Stokes equations can be interpreted as a set of convection-diffusion equations. In order to focus the attention in the spatial discretization, a steady state equation has been considered.

Before beginning the discretization, the generic convection-diffusion equation will be expressed in terms of a flux vector $\mathbf{J}$. Equation (2.1) can be expressed using vectorial operators as:

$$\nabla \cdot (\rho \mathbf{u} \phi) - \nabla \cdot (\Gamma \nabla \phi) = S \tag{2.2}$$

The total flux vector $\mathbf{J}$ is defined as

$$\mathbf{J} = \rho \mathbf{u} \phi - \Gamma \nabla \phi \tag{2.3}$$

It is decomposed into a convective $\mathbf{J}^C$ plus a diffusive $\mathbf{J}^D$ fluxes:

$$\mathbf{J}^C = \rho \mathbf{u} \phi \tag{2.4}$$

$$\mathbf{J}^D = -\Gamma \nabla \phi \tag{2.5}$$

Flux vector allows us to express equation (2.2) as

$$\nabla \cdot \mathbf{J} = S \tag{2.6}$$

To obtain a discrete approximation to the solution of equation (2.6), the first step is to divide the calculation domain is into a number of non-overlapping *control volumes* (CV). This partition of the domain is known as *mesh.* A *discretization node* (or, simply a *node*), where the approximated value of ϕ is evaluated, is located inside each CV.

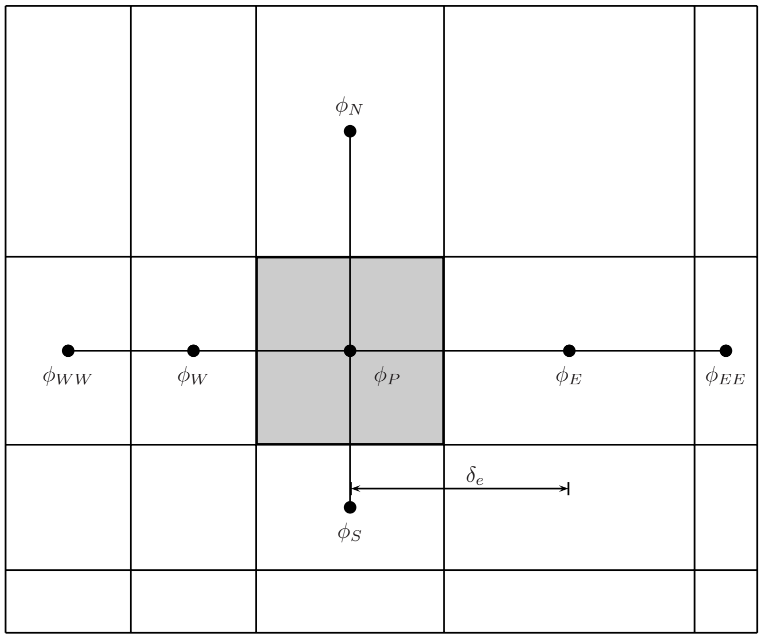

Figure 2.1: Fragment of a Cartesian structured two-dimensional mesh.

An algebraic, linear *discretization equation* will be obtained for each of the control volumes. The global set of linear equations will allow us to solve for each of the nodes. To do so, equation (2.6), is integrated in each CV. For instance, consider a generic control volume filled in the Cartesian mesh shown Fig. 2.1.

$$\int_{V_p} \nabla \cdot \mathbf{J} \, dv = \int_{V_p} S dv \tag{2.7}$$

At this point, V_p denotes the spatial region covered by the control volume. Using divergence theorem on the left hand side and expressing the right hand side in terms of the average of the source term, we obtain:

$$\int_{\partial V_p} \mathbf{J} \cdot \mathbf{n} \, ds = \overline{S} V_P \tag{2.8}$$

where V is the integrated volume, ∂V_p the outer surface of the control volume and $\mathbf{n}$ a unit vector normal to it. The left hand-side integral can be expressed as a sum of the contributions on each face. Assuming for simplicity a two-dimensional domain,

$$J_e - J_w + J_n - J_s = \overline{S} V_P \tag{2.9}$$

where J_f is the integral of the convective-diffusive flux of ϕ across cell face f (f = e,w,n or s). For instance,

$$J_e = \int_e \mathbf{J} \cdot \mathbf{n} \, ds = J_e^C + J_e^D \tag{2.10}$$

and $\overline{S}$ the averaged source term,

$$\overline{S} = \frac{1}{V_p} \int_{V_p} S dv \tag{2.11}$$

No approximations have been made so far. Equation (2.8) is still exactly equivalent to the set of integral equations over all the control volumes. The integral expressions (2.9) are *conservative*: they express the conservation principle for the unknown variable ϕ in the control volume considered, as the differential equation expresses it for an infinitesimal control volume. For any group of control volumes, including the whole computation domain, an integral conservation of quantities such as mass, momentum and energy is satisfied even for coarse meshes [57]. Thus, FVC method is said to be conservative.

Next step is to express integral equation (2.9) in terms of the unknown nodal values ϕ_E, ϕ_W, ϕ_N, ϕ_S, ϕ_P. To do so, two approximations are made:

1. Consider flux vectors **J** as constant along each face and equal to the value at the central point of the face.
2. Use a function profile to approximate the values of the flux vectors **J** at the central points of the interfaces as a function of the nodal values.

In order to preserve the conservative property of the method, the discretization has to be *consistent*, i.e, exactly the same expression has to be used to evaluate the convective and diffusive fluxes at the control volumes sharing an interface. Otherwise, a flux balance will be satisfied only in each control volume but neither in groups of control volumes nor in all the domain. It is convenient to treat convective and diffusive terms of **J** separately. Using as an example the e face we proceed as follows:

Diffusive terms are approximated using a second-order central difference scheme:

$$\begin{aligned} J_e^D = \int_e \mathbf{J}^D \cdot \mathbf{n} ds = \int_e -\Gamma \nabla\phi \cdot \mathbf{n} ds = \int_e -\Gamma \frac{\partial \phi}{\partial x} ds \approx \\ -\Gamma_e \left(\frac{\partial \phi}{\partial x} \right)_e S_e \approx -\Gamma_e S_e \frac{\phi_E - \phi_P}{\delta x_e} = D_e (\phi_P - \phi_E) \end{aligned} \tag{2.12}$$

Here the subindex e denotes the central point of the east interface and $D_e = \frac{\Gamma_e S_e}{\delta x_e}$. This central-difference approximation, involving the central node ϕ_P plus four neighbours is second order accurate[1]. If the generalized diffusion coefficient Γ depends on the position, the value Γ_e has to be interpolated from the neighbouring values Γ_E and Γ_P [57]. This can either be due to non-homogeneous domains ($\Gamma(\mathbf{x})$) or to non-constant physical properties ($\Gamma(T)$).

Convective terms are approximated as:

$$J_e^C = \int_e \mathbf{J}^C \cdot \mathbf{n} \, ds = \int_e \rho \mathbf{u} \phi \cdot \mathbf{n} \, ds = \rho \int_e u_e \phi \, ds \approx \rho u_e \phi_e S_e = F_e \phi_e \tag{2.13}$$

where u_e is the component of **u** normal to the e face, evaluated in its central point, S_e is the surface of the e face and $F_e = \rho u_e S_e$ is the mass flow rate at the e face. In general, an interpolation has to

[1]If the interface is located midway of the discretization nodes

be done to evaluate F_e, but this is not a critical point. The main problem is the evaluation of ϕ_e. It is considered to be a function of ϕ_P and its neighbouring nodes. For orthogonal meshes the values at the faces are usually considered to be a function of the neighbours at the same axis, i.e.

$$\phi_e = f\left(\phi_{WW}, \phi_W, \phi_P, \phi_E, \phi_{EE}\right) \tag{2.14}$$

To avoid physically unrealistic flows, the function f has to be bounded by the node values used in its interpolation. Different *numerical schemes* (functions suitable for this interpolation) have been proposed such as QUICK [73, 74] or SMART [75]. The latter scheme is used by default in DPC.

The **source term** $\overline{S}$ can be approximated in different ways. The most simple is to assume that the nodal value prevails over all the control volume,

$$\overline{S} = S_P \tag{2.15}$$

Combining the previous expressions, the discretized flux conservation equation can be written as:

$$\left(J_e^D + F_e\phi_e\right) - \left(J_w^D + F_w\phi_w\right) + \left(J_n^D + F_n\phi_n\right) - \left(J_s^D + F_s\phi_s\right) = S_P V \tag{2.16}$$

that can be expressed as a linear relation between ϕ_P and its neighbours:

$$a_P\phi_P = \sum_{nb} a_{nb}\phi_{nb} + b \tag{2.17}$$

However, direct expansion of equation (2.16) does not necessarily lead to acceptable coefficients for equation (2.17). Two conditions should be satisfied:

1. To avoid physically unrealistic results, the coefficients should be positive.
2. If an iterative solver is used to solve equation (2.17), it has to be diagonal dominant,

$$a_P \geq \sum_{nb} a_{nb} \tag{2.18}$$

Additionally, it is considered important that both convective and diffusive fluxes should be approximated using second order accurate schemes. A procedure to guarantee that these conditions are satisfied is the *deferred correction procedure* (DC) [71, 72]. It is an iterative algorithm. To use it, we should have a distribution of values of ϕ available. It can be the initial guess or the result of a previous iteration. The convective flux at the control volume face is replaced by

$$F_e\phi_e = F_e\phi_e^U - F_e\left(\phi_e^U - \phi_e\right) \tag{2.19}$$

where the value ϕ_e^U is a first order approximation of ϕ_e. Usually (but not necessarily) upwind scheme is used. Accordingly, the ϕ value is equal to the value at the grid point on the upwind side of the face e:

$$\phi_e^U = \begin{cases} \phi_P & \text{if } u_e > 0 \\ \phi_E & \text{if } u_e < 0 \end{cases} \tag{2.20}$$

The upwind convective flux is then expressed as:

$$F_e\phi_e^U = \phi_P \max\left(F_e, 0\right) - \phi_E \max\left(-F_e, 0\right) \tag{2.21}$$

This formulation was originally proposed to be used without the deferred correction terms. It is more stable than a central difference approximation, but too inacurate for current standards. It arose from its physical interpretation: the convection transport carries the information downstream but not upstream. As a base formulation it is usually preferred over other alternatives such as Power Law scheme [57] because it allows a clear separation between convective and diffusive terms. This separation allows the fluxes to be evaluated according to different spatial and temporal schemes. This is convenient for many reasons. For instance, in flux-splitting techniques [60], diffusive terms are treated implicitly and convective terms explicitly.

Equation (2.16) can then be expressed as:

$$\left(J_e^D + F_e\phi_e^U\right) - \left(J_w^D + F_w\phi_w^U\right) + \left(J_n^D + F_n\phi_n^U\right) - \left(J_s^D + F_s\phi_s^U\right) = \overline{S}V + b_{dc} \tag{2.22}$$

where

$$b_{dc} = F_e\left(\phi_e^U - \phi_e\right) - F_w\left(\phi_w^U - \phi_w\right) + F_n\left(\phi_n^U - \phi_n\right) - F_s\left(\phi_s^U - \phi_s\right) \tag{2.23}$$

The left hand-side of the equation is an implicit first order approximation, involving only five unknowns. The term b_{dc} of the right hand-side is a *deferred correction* to achieve the desired second order precision. This correction is done using ϕ values of the control volume faces explicitly evaluated according to the currently available nodal values.

Introduction of equation (2.21) into equation (2.23) gives the following expression for the discretization coefficients:

$$\begin{aligned} a_E &= D_e + \max(-F_e, 0) \\ a_W &= D_w + \max(F_w, 0) \\ a_N &= D_n + \max(-F_n, 0) \\ a_S &= D_s + \max(F_s, 0) \\ a_P &= \sum_{nb} a_{nb} + F_e - F_n + F_w - F_s \\ b &= S_P V_P + b_{dc} \end{aligned} \tag{2.24}$$

Coefficients a_{nb} are positive independently of the sign of the mass fluxes. Similar expressions would be obtained using other low order schemes instead of upwind. The other condition, diagonal dominance ($a_P > \sum_{nb} a_{nb}$) is equivalent to the condition:

$$F_e - F_n + F_w - F_s \geq 0 \tag{2.25}$$

This sum of the mass flow rates should be identically equal to zero for divergence free velocity fields, i.e. for steady-state compressible flows and always for incompressible flows. However, diagonal dominance must be ensured even for non-divergence free velocity fields, that even for the case of incompressible flows may arise during the convergence process depending on the solution algorithm. To get rid of the term $F_e - F_n + F_w - F_s$, discrete mass conservation equation multiplied by ϕ_p, $\phi_P\left(F_e - F_n + F_w - F_s\right)$ is subtracted to the previous discretization equation. An expression equivalent to equation (2.24) is obtained, but using:

$$a_P = \sum_{nb} a_{nb} \tag{2.26}$$

The notation used here for discrete equations (as in equation 2.17) emphasizes the relation between one node and its neighbours, so it is useful to describe the discretization process. If a Gauss-Seidel algorithm is used to solve the equation system, it is possible to evaluate only the coefficients for a single node and then solve it. This is mandatory if RAM memory is scarce, and maybe this practice is in the origin of the notation. Currently, the number of nodes is usually limited by the CPU time and not by the RAM, unless special techniques such as Dual Shur complement are used. If the coefficients for all the problem are evaluated before calling the solution procedure, better algorithms can be used. It is of course possible to use a matrix notation, i.e. $A\phi = b$, where A is a penta-diagonal matrix, to be assembled from the discretization equations. For certain solution algorithms, like Krylov [76] or Dual Schur complement (section 6), the latter notation is clearly better.

In our overview we have outlined the control volume discretization process, introducing different rules to be obeyed by the resulting algebraic equation: consistency at the control volume faces, positive coefficients, diagonal dominance. An additional rule discussed in [57], negative-slope linearization of the source term, has not been considered.

The general transient convection-diffusion equation, here not considered to stress the separation of the spatial and temporal discretization steps, is slightly more complex than equation (2.24) but their coeffients are equivalent with respect to the three rules introduced: the transient terms are only involved in the expressions for a_P and b and they are positive, contributing to increasing the diagonal-dominance of the system.

However, due to the non-linear nature of the equations, to the couplings between the variables and to the deferred correction method used, temptative $\mathbf{u}$ and ϕ fields should be used to evaluate the coefficients of the linear equations. As certain variables are only defined for positive values, care has been taken to guarantee the positiveness of the right hand side of the linear equation (2.17). If this is done, positive values of ϕ will always be obtained. However, the deferred correction term (equation 2.23) can be negative and spurious negative values of ϕ can in some cases be obtained during the iterative process. This may cause (usually causes) divergence.

Remarks about the FCV method:

- **Application to other models**. One of the main advantages of the finite control volume method is that it can easily deal with other convection-diffusion equations, including additional non-linear terms. This is useful, as it allows a code to solve a variety of transport phenomena, perhaps not with optimal efficiency but with only "minor changes".

- **Application to other meshes**. This formulation can be extended to more general Cartesian meshes [3, 77]. Non-orthogonal Cartesian meshes present additional difficulties [78]. Usually, FCV method is associated with structured meshes, but there are many examples of non-structured applications of the FCV method for CFD, such as [79, 80, 81, 82].

- **Accuracy.** The use a conservative method does not imply anything about the accuracy of the solutions obtained with a particular grid size, but it is certainly helpful while debbuging computer codes and to control the iterative numerical solution process.

  As in other methods, the accuracy of the solution achieved with a given mesh depends on the order of the profile assumed for the unknown function and on the mesh disposition. For a given number of nodes, the optimal mesh disposition depends on the problem being solved. In spite of the iterative nature of the DC, it is faster to use a second order accurate scheme and DC rather than a first order scheme and a finer mesh. A detailed example can be found in [6]. The DC approach has been extended to higher accuracies [83].

- **Staggered meshes**. The source term of the momentum equations contains pressure gradient terms such as $-\frac{\partial p}{\partial x_1}$. If the nodes of pressure and velocity components are located in the central node of each control volume, each velocity component is related to two alternate (and not adjacent) pressure nodes. This causes spurious checkerboard pressure fields to arise during the iterative process. There are methods to solve this inconvenience without changing the position of the discretization nodes [78]. However, the classic solution is the *staggered grid*, where the scalar variables (T, p, etc) are located at the central nodes as shown in Fig. 2.1, but the control volumes for the momentum equations are staggered half a control volume in the respective direction (i.e. x direction for x momentum equation). This mesh disposition complicates the discretization procedure. However, it is clearer from a mathematical and physical points of view.

- **Sparsity patterns of discretization equations**. Discrete equations obtained with finite differences, control volumes or elements retain the locality underlying in the continuous PDE:

each unknown is related *explicitly* only to the neighbouring unknowns. The discrete linear equations to be solved are huge, but *sparse* (a significant fraction of the elements A_{ij} in matrix A is 0). If the mesh is structured (or unstructured but with the unknowns carefully ordered), A is *banded*:

$$A_{ij} \neq 0 \, if \;\; |i-j| < B_W \tag{2.27}$$

If structured meshes are used, the dependence relations between a node and its neighbours is constant and matrices have a *fixed band size*. If the dependence of each unknown is restricted to its immediate neighbours, penta-diagonal matrices and hepta-diagonal matrices are obtained for two-dimensional and three-dimensional problems respectively. The direct use of higher order discretization schemes, without the deferred correction approach, would increase the number of nodes involved in the discrete equation obtained for each finite control volume. More information can be found in [84]. However, this approach is not often used in practice.

In spite of the sparse pattern of A, its inverse A^{-1} is *not* sparse. This means that the value of each point in the solution $\phi = A^{-1}b$ has a wider area of influence. This is considered in relation to temporal integration in next section.

Unlikely finite differences, finite control volumes or finite elements, *boundary element method* [85] approximates the unknown continuous function only in its boundaries. Thus, the area to be discretized has one dimension less, the number of unknowns is only $O\left(L^2\right)$, $O\left(L^1\right)$ for three and bi-dimensional problems respectively, where L is a representative measure of the size of the domain. However, the matrices obtained are *dense*: each of the boundary values is explicitly related to all the others.

2.4 Temporal discretization: time marching

2.4.1 Difference between spatial and temporal coordinates

When presenting the model problem in section 1.2, both position $\mathbf{x}$ and time t were classified as independent variables. There is, however, an important difference between them. Independent variables (coordinates) can be one-way or two-way:

- A *two-way* coordinate is such that the conditions at a given location in that coordinate are influenced by changes in conditions on *either side* of that location.
- A *one-way* coordinate is such that the conditions at a given location in that coordinate are influenced by changes in conditions on *only one side* of that location.

The concept is relative to the PDE being considered. The same idea can be expressed more accurately using the concepts of parabolic, elliptic and hyperbolic PDEs. See [23] for a discussion in the context of CFD problems.

As to our fluid flow and heat transfer problems, space is usually a two-way coordinate. Consider for instance heat conduction equation. The temperature changes in one point influence neighbouring points in all the spatial directions, i.e. if a heat source is applied to a point in a rod, both sides are heated. The same holds for convection-diffusion equations[2]. This is why we have to discretize and solve *all* the spatial domain simultaneously.

Time is always a one-way coordinate. As an example, the temperature at any point is only affected by the current and past temperatures. So we can discretize just a small part of the temporal domain and predict the behavior of the system in the immediate future. The same procedure is repeated at the next time step until the end of the temporal domain is reached.

[2]Space, however, can be a one-way coordinate under certain conditions. This has been used in the parallel algorithm in section 4.7.

2.4.2 Explicit versus implicit discretization. Transition matrices.

The different options to discretize the time coordinate of transient equations can be classified into *explicit* and *implicit*. The differences in both categories of numerical methods are crucial. To illustrate them, we will consider as a model problem a one-dimensional heat conduction equation,

$$\rho c_p \frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2} \tag{2.28}$$

It will be discretized using finite control volume method over a mesh with the nodes equally spaced at $\Delta x = L/N$. Periodical boundary conditions $T(0) = T(L)$ will be used for simplicity. An example of a physical system with such a behavior is an infinitely long rod with a periodical initial temperature, $T(t=0,x) = T(t=0,x+L)$.

For constant ρ and c_p values, the total amount of energy of the system is proportional to $\sum T_i$. Due to the periodic boundary conditions implemented in our example, no heat is lost to the surrounding media, but only redistributed to all the nodes of the system. Thus, we should expect $\sum T_i$ to remain constant.

Due to the presence of the time derivative in the equation, during the discretization process we have to face the problem of finding a discrete approximation for terms such as:

$$\int_t^{t+\Delta t} T_P dt \tag{2.29}$$

A family of approximations for this integral can be introduced as a function of a parameter f :

$$\int_t^{t+\Delta t} T_P dt \approx \left[f T_P^1 + (1-f) T_P^0 \right] \Delta t \tag{2.30}$$

where T_P^0 and T_P^1 are the present (known) and future (to be determined) temperatures of a generic node T_P and f is a parameter between 0 and 1 to be specified. For $f = 0$ we have an *explicit* scheme. Its main feature is that there are no unknown values involved in the approximation. Schemes with $f > 0$ are partially implicit. For $f = 1$, we have a *fully implicit* scheme.

The final discretization equation is a function of f :

$$\frac{1}{Fo}\left(T_P^1 - T_P^0\right) = f\left(T_W^1 - 2T_P^1 + T_E^1\right) + (1-f)\left(T_W^0 - 2T_P^0 + T_E^0\right) \tag{2.31}$$

where $Fo = \frac{k\Delta t}{\rho c_p \Delta x^2}$ is the Fourier number. Equation (2.31) can expressed matricially as

$$AT^1 = BT^0 \tag{2.32}$$

where,

$$A = \left[-f, \frac{1}{Fo} + 2f, -f \right] \tag{2.33}$$

$$B = \left[1-f, \frac{1}{Fo} - 2(1-f), 1-f \right] \tag{2.34}$$

Here stencil notation has been used [86]; central element is the main diagonal value; left and right elements are coefficients of the respective neighbours. The eastern neighbour of the last node is the first node and vice versa. Due to the periodicity of the system, A and B are not exactly tridiagonal, i.e., for $N = 5$,

$$A = \begin{bmatrix} 2f + Fo^{-1} & -f & 0 & 0 & -f \\ -f & 2f + Fo^{-1} & -f & 0 & 0 \\ 0 & -f & 2f + Fo^{-1} & -f & 0 \\ 0 & 0 & -f & 2f + Fo^{-1} & -f \\ -f & 0 & 0 & -f & 2f + Fo^{-1} \end{bmatrix} \tag{2.35}$$

$$B = \begin{bmatrix} Fo^{-1} - 2(1-f) & 1-f & 0 & 0 & 1-f \\ 1-f & Fo^{-1} - 2(1-f) & 1-f & 0 & 0 \\ 0 & 1-f & Fo^{-1} - 2(1-f) & 1-f & 0 \\ 0 & 0 & 1-f & Fo^{-1} - 2(1-f) & 1-f \\ 1-f & 0 & 0 & 1-f & Fo^{-1} - 2(1-f) \end{bmatrix} \tag{2.36}$$

The *transition matrix* [69] P of the discrete equation relates explicitly T^1 to T^0:

$$T^1 = A^{-1}BT^0 = PT^0 \tag{2.37}$$

P cannot be evaluated in a real problem, where $N \approx 10^5, 10^6$, as it involves the computation of A^{-1}. However it is a useful concept for our analysis.

For the explicit scheme, $f = 0$, we have :

$$\begin{aligned} A &= \left[0, \frac{1}{Fo}, 0\right] \\ B &= \left[1, \frac{1}{Fo} - 2, 1\right] \\ P &= \left[Fo, Fo\left(\frac{1}{Fo} - 2\right), Fo\right] \end{aligned} \tag{2.38}$$

As no unknown values of T^1 have been used in the construction of the approximation, A is a *diagonal* matrix. So there is no need to solve any equation system. Moreover, P is a sparse matrix. Only the known values of the immediate neighbours (T_W^0, T_P^0 and T_E^0) have effect over T_P. For the case of higher order spatial schemes involving more points, the stencil would be wider but the matrix would still be sparse. This is excellent news for parallel computing with explicit schemes.

This is not the case for the implicit schemes. In particular, consider the fully implicit scheme with $f = 1$. We have

$$\begin{aligned} A &= \left[-1, 2 + \frac{1}{Fo}, -1\right] \\ B &= \left[0, \frac{1}{Fo}, 0\right] \end{aligned} \tag{2.39}$$

and P is a very ugly *dense* matrix (with $P_{i,j} \neq 0$ for all i,j) [3]. So for each time step, we have to solve a linear system of equations in order to obtain T^1 from T^0.

Apparently, explicit schemes are the best option. However, they have a serious inconvenience: the maximum time step that can be used is limited. If it is too long, the algorithm becomes *unstable*. For our model equation, Von Neumann stability analysis leads [23] to

$$Fo < \frac{1}{2} \rightarrow \Delta t < \frac{\rho c_p \Delta x^2}{2k} \tag{2.40}$$

The same result is obtained in [57] using the rule of the *positive coefficients* (based on physical arguments).

[3] Shown in next section.

To show how catastrophic the consequences of violating equation (2.40) are, consider an example with an initial vector $T^0 = e_k$[4].

According to equation 2.38, the transition expression for an explicit scheme is: $T^1_{k-1} = T^1_{k+1} = Fo$, $T^1_k = Fo\left(\frac{1}{Fo} - 2\right)$ and $T^1 = 0$ for all the other points. For $Fo > \frac{1}{2}$, we have the very unrealistic values $T^1_k < 0$. Indeed, for our physical system we should expect $T^1_k \geq T^1_{k-1} = T^1_{k+1}$. This condition is broken for $Fo > \frac{1}{3}$, even before the stability limit.

The heat lost by node T_k (right hand side of equation 2.28), is evaluated according to the temperature distribution T^0. As T^1_k is decreasing, the heat is overestimated. The higher the time step, the higher the overestimation. Beyond certain values of Fo, the result is not only unaccurated, but also unrealistic and unstable. Note that if spatial accuracy is increased, Δx decreases and the time step has to be reduced.

For the full Navier-Stokes there is no exact stability analysis (at least for the finite-difference formulation [23]). However, the *Courant-Friederichs-Lowry* (CFL) criterion provides a good guidance: Δt must be less than the time a sound wave takes to move from one grid point to the next, i.e.,

$$\delta t < \frac{\delta x}{c} \tag{2.41}$$

where c is the sound velocity.

As **for incompressible flows** , the sound velocity is in practice infinite (section 1.2), **fully explicit algorithms are not possible**. However, parts of the integration (momentum and energy equations) can be done explicitly.

As discussed in previous paragraphs, the transition matrix P is dense for implicit schemes. This does not necessarily mean that the influence of T^0_j is equal for all the nodes in T^1. Consider the transition matrix P of our example for $N = 5$,

$$P = \frac{1}{5Fo^2 + 5Fo + 1} \times \left[\begin{array}{ccccc} Fo^2 + 3\,Fo + 1 & Fo\,(Fo+1) & Fo^2 & Fo^2 & Fo\,(Fo+1) \\ Fo\,(Fo+1) & Fo^2 + 3\,Fo + 1 & Fo\,(Fo+1) & Fo^2 & Fo^2 \\ Fo^2 & Fo\,(Fo+1) & Fo^2 + 3\,Fo + 1 & Fo\,(Fo+1) & Fo^2 \\ Fo^2 & Fo^2 & Fo\,(Fo+1) & Fo^2 + 3\,Fo + 1 & Fo\,(Fo+1) \\ Fo\,(Fo+1) & Fo^2 & Fo^2 & Fo\,(Fo+1) & Fo^2 + 3\,Fo + 1 \end{array}\right] \tag{2.42}$$

For a periodic system such as our model problem[5], the influence of T^0_j over T^1_i is only a function of $|i - j|$, as the matrix is symmetrical and all the rows are equal (only shifted). The further from the main diagonal, the smaller the coefficient.

The central row (or column) of P has been represented for different values of Fo in Fig. 2.2. For small Fo values, the majority of the influence is concentrated in the neighbours of each node, while for large values the influence is extended to all the domain. The area where the influence of T^0_k is important decreases with the time step. This has important implications for implicit integration on parallel computers.

For this particular model (as can be seen for $N = 5$ from equation (2.42)) we have

$$\lim_{Fo \to \infty} P_{i,j} = \frac{1}{N} \tag{2.43}$$

This means that when the steady state is reached, the energy of the system is equally distributed for all the nodes. The initial value of each node influences the rest of the system. The unconditionally stable explicit scheme correctly predicts this feature.

[4]Equal to zero everywhere except for the node k where $T^0_k = 1$.

[5]Periodicity simplifies the analysis but it is not essential. Similar results would be obtained for other cases.

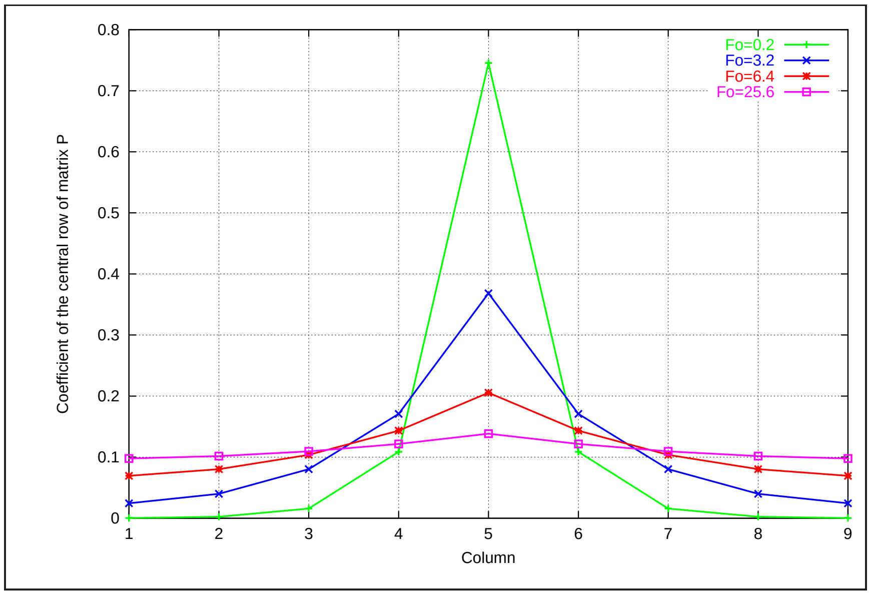

Figure 2.2: Central row coefficients of transition matrix P for different Fo numbers.

2.4.3 Pseudo-Transient method

Time marching can be used for two reasons:

1. To obtain a time accurate solution of a transient flow. The time step is limited by the truncation error associated with the discretization of the time derivatives.

2. To obtain a steady-state solution assuming some initial conditions for the unknowns and then solve the evolution of the system until a steady-state flow is reached. In principle, the time marching procedure used does not have to be time accurate. We will refer to this approach as *pseudo-transient* method [87].

   If implicit schemes are used, the time steps can be much longer. In principle, to solve a single convection-diffusion equation, an arbitrary long time step could be used. This is not the case for the Navier-Stokes equations, because they are coupled and non-linear. If a too long time step is used, the algorithm may diverge or oscillate indefinitely.

   The capability to reach a convergent steady solution with a wide range of time steps is often used to measure the robustness and efficiency of the method to solve the non-linearities and couplings [88].

If an implicit scheme (stable for any Δt) is used, the same code can be used for both pseudo-transient and time accurate types of problems, with only minor changes in the parameters of the algorithm. However, in pseudo-transient problems, the time steps are much longer and this makes both types of problems quite different (with respect to the momentum and energy equations).

For both, $\mathbf{\Phi}^0$ is used as the initial guess of $\mathbf{\Phi}^1$. For pseudo-transient algorithms, if Δt is long, this might not be a very good approximation. Thus, the linearization method and the treatment of the couplings become important parts of the algorithm.

Additionally, for the convection-diffusion equations, the areas of effective influence of each point are larger in pseudo-transient algorithms. If the algebraic equations are iteratively solved, this is an important difference: the shorter the time step, the smaller the number of iterations needed. For long time steps, the use of algorithms that transport information efficiently across the domain (such as MG), becomes important.

The oposite is also valid: if short time steps are used, only the areas of effective influence of pressure correction equation remain large. This the reason why in the example of section 3.3.3, only the pressure correction equation is solved with multigrid, while a single sweep of an iterative solver is enough for transport equations.

Moreover, this is also the reason why for the case of parallel solvers for incompressible flows, the solution of pressure correction equation is the bottleneck. Chapters 5, 6 and 7 are focused on this aspect.

A warning about pseudo-transient method. It is generally accepted that for pseudo-transient method, the time-marching procedure does not have to be time-wise-accurate (i.e., [23] page 152). To be more precise, we should perhaps say that *if the problem has a steady state solution*, the time-marching procedure does not have to be time-wise-accurate.

However, the system (1.1-1.5) does not necessarily have a steady-state solution even for time constant boundary conditions. For instance, consider the well-known driven cavity problem with $Re = 10^4$. The existence of a Hopf bifurcation (from steady state solution to periodic flow) is shown in [89] close to Re=8000. If a situation without steady solution is solved using a pseudo-transient method, with a time step too long to have a time-accurate solution, the relatively small oscillations of the flow can be averaged by the solution algorithm and a *false* (in the same sense as the symmetric solution of section 1.3) steady solution can be obtained. For example, a steady state solution for the problem with $Re = 10^4$ is given in [90].

2.5 Linearization

Assume that after temporal and spatial discretization, we have a set of non-linear equations such as:

$$A\left(\mathbf{\Phi}\right)\mathbf{\Phi} = b\left(\mathbf{\Phi}\right) \tag{2.44}$$

where A and b are a matrix and a vector that depend on $\mathbf{\Phi}$. Here $\mathbf{\Phi} = (u, v, w, p, T)$ contains all the discrete unknowns. Vector b is also a function of the (known) solution of the previous step, $\mathbf{\Phi}^0$, but this is not important here.

Equation (2.44) can not be directly solved as a linear discrete equation would be. To do so, the non-linear problem is decomposed into a sequence of linear problems. It is useful to express the previous equation as:

$$F\left(\mathbf{\Phi}\right) = A\left(\mathbf{\Phi}\right)\mathbf{\Phi} - b = 0 \tag{2.45}$$

We must find a succession $\mathbf{\Phi}^k$ so that,

$$\lim_{k\to\infty} F\left(\mathbf{\Phi}^k\right) = 0 \tag{2.46}$$

And, if possible, choose the succession of faster convergence. Starting from an arbitrary $\mathbf{\Phi}^1$ (usually the previous time step solution, or the initial conditions) there are two main options to construct the sequence:

- Frozen coeffients method[6] uses a zero order approximation to $F\left(\mathbf{\Phi}\right)$:

  $$F\left(\mathbf{\Phi}^{k+1}\right) = A\left(\mathbf{\Phi}^{k+1}\right)\mathbf{\Phi}^{k+1} - b\left(\mathbf{\Phi}^{k+1}\right) \approx A\left(\mathbf{\Phi}^{k}\right)\mathbf{\Phi}^{k+1} - b\left(\mathbf{\Phi}^{k}\right) \tag{2.47}$$

  and then solves the linear problem

  $$A\left(\mathbf{\Phi}^{k}\right)\mathbf{\Phi}^{k+1} = b\left(\mathbf{\Phi}^{k}\right) \tag{2.48}$$

  to obtain $\mathbf{\Phi}^{k+1}$.

[6]Incidentally, this approach has many different names: Picard's method [91, 92, 93], frozen coefficients iteration [94], successive substitution method [95] or standard linearization [96]. This is the approach proposed in [57].

- Newton-Raphson method. Taylor expansion is used to evaluate a first order approximation to $F(\mathbf{\Phi})$,

$$F\left(\mathbf{\Phi}^{k+1}\right) \approx F\left(\mathbf{\Phi}^{k}\right) + J\left(\mathbf{\Phi}^{k}\right)\left(\mathbf{\Phi}^{k+1} - \mathbf{\Phi}^{k}\right) \tag{2.49}$$

where J is the Jacobian of F evaluated at $\mathbf{\Phi}^k$,

$$J\left(\mathbf{\Phi}^{k}\right) = \begin{pmatrix} \frac{\partial F_1}{\partial \mathbf{\Phi}_1} & \frac{\partial F_1}{\partial \mathbf{\Phi}_2} & \cdots & \frac{\partial F_1}{\partial \mathbf{\Phi}_n} \\ \frac{\partial F_2}{\partial \mathbf{\Phi}_1} & \frac{\partial F_2}{\partial \mathbf{\Phi}_2} & \cdots & \frac{\partial F_2}{\partial \mathbf{\Phi}_n} \\ \vdots & & \ddots & \\ \frac{\partial F_n}{\partial \mathbf{\Phi}_1} & \frac{\partial F_n}{\partial \mathbf{\Phi}_2} & \cdots & \frac{\partial F_n}{\partial \mathbf{\Phi}_n} \end{pmatrix} \tag{2.50}$$

A linear equation for $\mathbf{\Phi}^{k+1} - \mathbf{\Phi}^k$ is obtained forcing this approximation of F to be zero,

$$J\left(\mathbf{\Phi}^{k}\right)\left(\mathbf{\Phi}^{k+1} - \mathbf{\Phi}^{k}\right) = -F\left(\mathbf{\Phi}^{k}\right) \tag{2.51}$$

Remarks:

- For any of the two methods, for a general non-linear problem, there is no guarantee that the succession of the solutions of the linear problems is convergent. There is no general method to establish a priori if a given succession will be convergent.

- The convergence of the frozen coefficients method is slower, but their *region of convergence* (the set of points $\mathbf{\Phi}^1$ for which the succession is convergent) is larger. A possible approach is to start the iterations with the frozen coefficients method and, when a certain criteria is satisfied, carry on with the Newton method. In general, if the initial guess is close to the solution of the non-linear problem, the probability to obtain a convergent sequence increases. For pseudo-transient problems, this can be achieved with smaller time steps.

- Sparsity pattern of J matrix is more complex than sparsity pattern of A. This causes problems to the linear solution algorithm. General sparse matrix linear solvers fail to take advantage of the specific patterns of matrix J. The Additive Correction Multigrid method has recently been extended to the solution of the block penta-hepta diagonal matrices arising from Newton-Rapshon method [97].

- In both cases, CPU time can be saved using an approximation to the solution of the linear problem rather than the exact solution. This is, not reaching an accurate solution but doing just a few sweeps with an iterative algorithm. However, this usually deteriorates the convergence.

- The matrix of the frozen coeffients method is easier to evaluate and solve. Numerical derivatives are usually used for Newton-Raphson algorithm. If piecewise functions are used for the high order discretization schemes, such as SMART [98], $F(\mathbf{\Phi})$ can be non derivable, potentially leading to problems.

- Criteria to stop the iterations. The residual of the linear solvers has been used as a criteria (e.g., [6]) to stop the non-linear iterations. In general, this is *not* a valid method, as the exact solution of the $k+1$ iteration, $\mathbf{\Phi}^{k+1} = \left(A\left(\mathbf{\Phi}^{k}\right)\right)^{-1} b\left(\mathbf{\Phi}^{k}\right)$, that has a null residual, may be very different to the solution of the non-linear problem $A(\mathbf{\Phi})\,\mathbf{\Phi} = b(\mathbf{\Phi})$.

  The criteria based on the residuals of the linear equations (i.e. the mass residual), usually work because the the linear solvers typically used are relatively inefficient, so they will only be able to solve the linear equation for $\mathbf{\Phi}^{k+1}$ when it is very close to $\mathbf{\Phi}^k$, i.e., when we are close to the solution of the non-linear problem.

However, according to this approach, the tolerance should be changed for different linear solvers. A better idea is to stop the non-linear iterations when

$$||\mathbf{\Phi}^{k+1} - \mathbf{\Phi}^{k}|| < \epsilon \tag{2.52}$$

or

$$\frac{||\mathbf{\Phi}^{k+1} - \mathbf{\Phi}^{k}||}{||\mathbf{\Phi}^{k+1}||} < \epsilon \tag{2.53}$$

- Relaxation technique is often needed to improve the convergence region of the algorithm. One of the possible formulations is to obtain $\mathbf{\Phi}^{k+1}$ with any approach and then substitute it by $\tilde{\mathbf{\Phi}}^{k+1}$,

$$\tilde{\mathbf{\Phi}}^{k+1} = \mathbf{\Phi}^{k} + \alpha\left(\mathbf{\Phi}^{k+1} - \mathbf{\Phi}^{k}\right) \tag{2.54}$$

  where here α is a relaxation factor taking values from 0 to 1 (no relaxation). Another approach, based on the alteration of the linear problem, can be found in [57]. As aforementioned, time step can also be used as a relaxation parameter. Inefficient linear solvers, effective only to reduce the high frequency components of the error, can in practice act as relaxation methods, providing better convergence at the cost of a much higher CPU time.

- The non-linearity (and difficulty) of the model problem is increased if additional non-linear phenomena (such as turbulence modeling, combustion, free surfaces or thermal radiation) are to be simulated.

The real difficulty associated with the non-linearity of the governing equations is not the obtention of convergent algorithms for a given mesh size: this is almost always possible by relaxation and/or reducing the time step. The main difficulty is the dramatic increase in the complexity of the solutions of the equations (turbulence), and consequently, in the number of unknowns needed to approximate them, as the non-linear terms become more important. This is by far the most cumbersome problem associated with Navier-Stokes equations.

2.6 Coupling

After the previous steps, the solution of the governing equations can be expressed as a sequence of linear algebraic equations, where the unknowns would be velocity components, pressure and temperature at the nodes. All the unknowns are directly or indirectly coupled. With respect to their coupling, the set of equations (1.1-1.5) presents the following particularities:

- Pressure is only explicitly present in momentum equations.
- Temperature field affects velocity and pressure through vertical momentum equation.
- Velocity components are coupled through momentum and continuity equation.

These physical couplings are inherited by the discrete linear equations. Usually, the main troubles come from the pressure-velocity coupling.

2.6.1 Coupled methods

In principle, the solution of the linear system for all the unknowns u, v, w, p, T could be achieved without more considerations, using an appropriate solver. Of course, to find an efficient method to do so is the main problem of the coupled approaches. The set can also be split into smaller subsets. Possible groups to be solved together are:

- Mass conservation and momentum components: SCGS [99], CELS [88], CSIP [100].
- Momentum components [101].

- All the equations [97].

If a pseudo-transient (non time-accurate) approach is used to reach steady state solution, the coupled approach is more robust and allows larger time steps. However, for time accurate simulations, short time steps are usually needed. In this case, momentum equations (but not continuity equation) are coupled mainly at a local level and a coupled algorithm can waste time doing unnecessary work over certain parts of the problem. Therefore, under these conditions, segregated algorithms tend to be faster.

2.6.2 Segregated methods

In segregated methods, a scalar equation for each unknown is solved separately, using the available values of the other unknowns. Each of the visits to all the equations is called an outer iteration. A number of outer iterations (that usually increases with Δt) is done until convergence is achieved.

However, for the coupling between the pressure and the velocity, additional techniques are needed. Due to the incompressibility of the flow, pressure is absent of mass conservation equation but pressure gradient terms are part of source term of the momentum equations.

Thus, there is nothing like a "pressure equation". Quoting [57]: "The pressure field is indirectly specified via the continuity equation. When the correct pressure field is substituted into the momentum equation, the resulting velocity satisfies the continuity equation".

Different segregated algorithms have been proposed. A possible classification is:

- Pseudo-compressibility methods [102].
- Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) [57] and derivates such as the SIMPLE Consistent (SIMPLEC) [59] algorithm.
- Projection methods [60].

An important property of SIMPLE algorithm (and its derivates) and projection methods, is that to impose the incompressibility constraint, they use a symmetric matrix that remains constant along all the iterations and time steps. This is exploited by the parallel algorithms designed in sections 6 and 7.

For problems where the vertical temperature gradient is an important source of momentum, such as the problem model (1.1- 1.5), an extension of SIMPLEC algorithms has been proposed [103].

2.7 Solution of linear algebraic equations

After the previous steps, only the solution of the linear equations remains to obtain the discrete distributions of velocity, pressure and temperature. Spatial discretization is conceptually more complex, but in terms of CPU time, it is usually quite cheap because the unitary cost of the assembly of the coefficient matrix is low [7] and grows linearly with the number of unknowns. This is the reason why it is usually better to use high order methods rather than larger meshes.

On the other hand, the solution of linear equations, which is a trivial problem for small systems, often becomes the bottleneck of the CFD codes due to the huge sizes of the systems to be solved. For a problem with N nodes, the size of the matrix is N^2, so its full storage is totally out of question. For instance, for a small bidimensional 100×100 nodes domain, $N = 10^4$ and the full matrix would have 10^8 numbers. Using double precision real numbers (8 bytes per number), about 760 Mbytes would be needed to store it. This is roughly the maximum memory that a standard main board for

[7] Or at least it should in principle be, but if a single code is used to solve different models, it becomes very complex and difficult to optimize. On the other hand, the use of a different code for each application would cause a huge maintenance effort.

a PC can currently (March 2000) hold (appendix B). However, much larger problems, at least in the range of $N = 10^6, 10^7$ are to be solved for many heat transfer applications (e.g, section 3.3.3).

For incompressible flows using segregated methods, the solution of the discrete linear equation arising from the incompressibility constraint (i.e., pressure correction equation) is the most critical. This is because:

- There are no transient terms that help to increase its diagonal dominance.
- Unless the pressure correction is fixed in a node, it is singular.
- The pressure correction equation has to be solved accurately at each iteration, and there is no good initial guess to do so.

Fortunately, for incompressible flows, the matrix remains constant so more CPU time can be used for a pre-processing stage, if later it is saved in the solution of each individual right hand side. This technique is used in sections 6 and 7.

In this section we will provide an overview of different techniques for the solution of a discrete scalar equation defined in a domain discretized with a structured mesh. Krylov-type solvers, such as CG or GMRES are very shortly considered in section 4.5.2. Specific algorithms for the coupled solution of the coupled system $\mathbf{u} - p$ will be considered in section 3.4. Thus, our problem here is to solve the equation

$$Ax = b \tag{2.55}$$

where A is a penta or hepta diagonal matrix.

In this work, LU decomposition and classic iterative algorithms (Gauss-Seidel, line-by-line Gauss-Seidel, Incomplete LU decompositions) will be briefly presented. We finish this introduction with a quote from the book by J.M.Ortega [104]: "Probably the main driving force for the development of vector and parallel computers has been scientific computing, and one of the most important problems in scientific computing is the solution of linear systems of equations. Even for conventional computers, this has continued to be an active area of research, especially for iterative methods."

2.7.1 Linear system used to benchmark the solution algorithms

A problem model will be introduced to benchmark the different (sequential and parallel) linear solution algorithms presented in this work. Its aim is to model the pressure correction equations on a uniform mesh. Consider the following matrix:

$$\begin{cases} a_P^{\mathbf{i}} = 1 \; ; \; a_{nb}^{\mathbf{i}} = 0 & \text{if } \mathbf{i} \in \text{boundary} \\ a_P^{\mathbf{i}} = \sum a_{nb} \; ; \; a_{nb}^{\mathbf{i}} = 1 & \text{otherwise} \end{cases} \tag{2.56}$$

Here $\mathbf{i}$ is the integer coordinate of each node (i.e., $\mathbf{i} = \{i, j, k\}$ in a three-dimensional mesh). A domain with equal number of nodes in each direction is used. To generate the right hand side vector b, we first generate x_s and then evaluate b as $b = Ax_s$. Then, the iterations are started from $x = 0$. Doing so, we can easily evaluate both the residual $(b - Ax)$ and the error $(x_s - x)$ of our approximate solutions.

The vector x_s is generated as:

$$x_s(i, j) = \cos(2\pi p_1 f(i)) + \cos(2\pi p_2 f(j)) \tag{2.57}$$

in two-dimensional domains and accordingly in three-dimensional domains. Here, $f(i)$ is a linear function that maps the integer values sweeped for i to the segment $[0, 1]$. The coefficients p_1 and p_2 can be changed to alter the error distribution of the initial guess. Low values correspond to low-frequency distributions, difficult for classic iterative algorithms. The lowest frequency $p_1 = p_2 = 0$ is constant. The values used for the problem model are $p_1 = p_2 = 1$, except where mentioned.

Otherwise said, the problem is solved with a precision ϵ that is 10^{-6} of the initial guess residual, i.e.,

$$\epsilon = 10^{-6}||b - Ax^0|| = 10^{-6}||b|| \tag{2.58}$$

There is a dilemma in the use of a problem model to benchmark linear solvers. On one hand, the behavior of the linear equation solvers is dependent on the nature of the flow, due to the spectral distribution of the residuals so it is very difficult to obtain conclusive information from the analysis of an "artificial" problem such as our problem model. On the other hand, if an actual CFD problem is used, there are too many parameters involved, related not only to the flow physics but also to the method, mesh disposition, time step, etc. Thus, it is difficult to provide all the information needed to reproduce the problem.

2.7.2 LU decomposition

LU decomposition is the only direct method considered here. Direct methods take much longer time than each of the sweeps of iterative methods but when they finish, provide an exact solution of the linear equation system (except roundoff errors, that according to our experience are not a problem). Direct methods are not useful for the solution of non-linear problems, where a sequence of linear equations with changing coefficients has to be solved with limited accuracy, but as fast as possible. Their main interest is in the solution of the pressure correction equations, where A remains constant during all the problem and the solutions should be more accurate to enforce incompressibility.

The matrix A can be factored into the product of a lower (L) and a upper (U) triangular matrices using *Crout's algorithm with partial pivoting* [105]. This factorization allows the solution of the system of equations

$$LUx = b \tag{2.59}$$

in two stages. We introduce an auxiliary vector y as $Ux = y$. Thus, our equation becomes:

$$Ly = b \tag{2.60}$$

where $y = (y_1 \cdots y_N)$. This equation can be easily solved using *forward substitution*: y_1 can be directly isolated from the first equation $L_{1,1}\, y_1 = b_1$ and then introduced in the second $L_{1,1}\, y_1 + L_{1,2}\, y_2 = b_2$ to isolate y_2. This procedure goes on until all the vector has been obtained.

Once y is available, we solve for x from

$$Ux = y \tag{2.61}$$

doing a *backward substitution.*

The main problem of LU decomposition is the CPU time of the decomposition and the memory needed to store it. Note that if matrix A is to be used to solve for many right hand sides, the factorization can be done just once. Band LU decomposition [105] reduces considerably the memory and time needed to do the decomposition. In our implementation, if the number of control volumes in each of the axes is different, the domain is rotated to select the minimum band size.

Under certain conditions, LU decomposition is a serious rival for advanced iterative algorithms such as ACM (section 3.2). Its main limitation however is the RAM memory needed. In this work, it has been used as an auxiliary procedure for the Shur decomposition method (section 6).

The CPU time needed to solve the LU system for the problem model has been represented as a function of the problem size in Fig. 2.3. For all the cases, the norm of the residual vector associated with the solution of the LU system was smaller than 10^{-11}. It grows more than linearly, but, if the goal is to solve the equation to machine accuracy, it is very fast compared with other algorithms. The time to evaluate the decomposition has been represented in Fig. 2.4. It is orders of magnitude larger than the solution time, so it is not worth using direct LU except for cases for constant matrices.

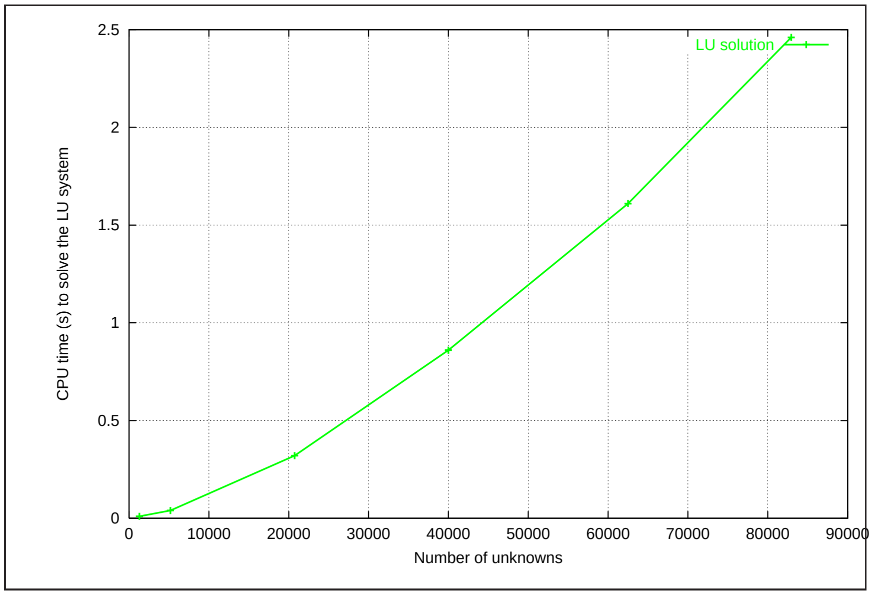

Figure 2.3: CPU time to solve a LU system.

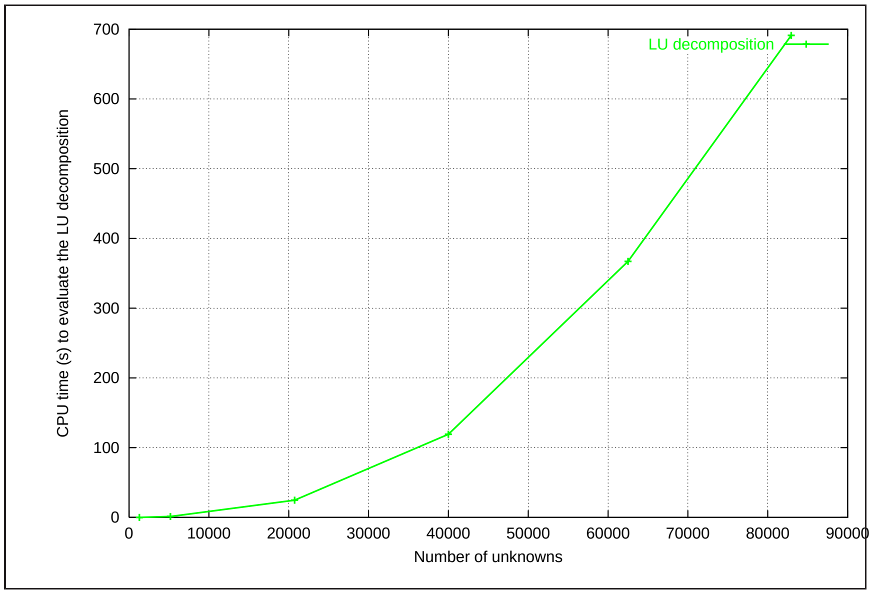

Figure 2.4: CPU time to evaluate the LU decomposition.

2.7.3 Classic iterative methods

Classic (or stationary) iterative methods were the first linear solution algorithms used for CFD. This is mainly due to the reduced amount of memory that they need. For small problems they are very useful, specially for the solution of strongly non-linear problems where a large number of linear subproblems are to be solved with low accuracy. For current computers, with many Mbytes of RAM, they are not the best option, specially for the solution of pressure correction equation where an accurate solution is needed at each iteration.

Gauss-Seidel and line-by-line Gauss-Seidel methods are easier to introduce using the neighbours notation (as in [57]) :

$$a_P\phi_P = \sum_{nb} a_{nb}\phi_{nb} + b \tag{2.62}$$

Each **Gauss-Seidel** iteration sweeps the domain node by node, isolating ϕ_P from each discrete equation.

$$\phi_P^{k+1} = \frac{\sum_{nb} a_{nb}\phi_{nb}^* + b}{a_P} \tag{2.63}$$

Here, the super index $*$ denotes the most recently available values of ϕ. This is, for the neighbouring nodes already visited in this iteration $\phi^* = \phi^{k+1}$ and for the others $\phi^* = \phi^k$.

In the **Jacobi** variant of the algorithm, the values obtained in the previous iteration are used:

$$\phi_P^{k+1} = \frac{\sum_{nb} a_{nb}\phi_{nb}^k + b}{a_P} \tag{2.64}$$

Jacobi is a parallel algorithm while Gauss-Seidel is not: there is a data dependency in equation (2.63). However, Jacobi convergence rate is worse, so more sweeps are needed.

Line-by-line Gauss-Seidel, restricted to structured meshes, sweeps the domain row by row (or column by column). Assuming that it is two dimensional and that it is being explored by rows. For each row, equation (2.62) is expressed as:

$$a_P\phi_P = a_W\phi_W + a_E\phi_E + B \tag{2.65}$$

where the new source term B includes the contribution from nodes in the other rows,

$$B = a_N\phi_N^* + a_S\phi_S^* + b \tag{2.66}$$

Equation (2.65) has a tridiagonal structure and can easily be solved using the Tridiagonal Matrix Algorithm (TDMA) [57, 105]. To enhance the convergence, the domain can be alternatively explored by rows or columns.

Most of the CPU time is spent in the TDMA subroutine. It has a data dependency problem that prevents its vectorization or paralelization. Attempts have been made to develop an efficient parallel version [106].

Like Gauss-Seidel, Line-by-line Gauss-Seidel is not parallel. A Jacobi variant, that uses the same idea as in equation (2.62), can be introduced to parallelize it.

Incomplete LU decompositions. The main idea of ILU algorithms is that to find an exact LU decomposition of A is costly, but to find an approximation to it can be much cheaper. To be more precise, we construct the exact LU decomposition of a matrix $A' \approx A$,

$$L'U' = A' = A + B \tag{2.67}$$

where B is a matrix such that

$$\|B\| \ll \|A\| \tag{2.68}$$

Different versions of the algorithm (SIP [107], MSIP [108, 109]) differ mainly (but not only, e.g. [110]) in how the matrix B is chosen. An algorithm that allows the user to control the amount of fill-in terms through two parameters, and that in the limit becomes an exact LU factorization, has been described in [111].

MSIP is used in our implementations as a smoother (section 2.7.3) for ACM and DDACM. In MSIP, the decomposition depends on a *partial cancellation parameter* α. It is a good idea to tune it for our particular application. The number iterations can be reduced increasing α value but the solver might fail to converge for certain problems if a too high α value is used[8]. Values between 0.4 and 0.94 are typically used. Its influence if used inside as a smoother for a multigrid algorithm is reduced. An automatic method to tune α has been proposed [110].

In order to solve equation (2.55), an iterative sequence is used. After iteration k, a vector x^k is available. Our goal for x^k would be

$$Ax^{k+1} = b \tag{2.69}$$

If we subtract Ax^k at both sides, we have:

$$Ax^{k+1} - Ax^k = b - Ax^k = r^k \tag{2.70}$$

From expression (2.68), we have

$$||B\left(x^{k+1} - x^k\right)|| \ll ||A\left(x^{k+1} - x^k\right)|| \tag{2.71}$$

Thus, we can add $B\left(x^{k+1} - x^k\right)$ to the left hand side of equation (2.70) to obtain :

$$Ax^{k+1} - Ax^k + Bx^{k+1} - Bx^k \approx r^k \tag{2.72}$$

$$(A + B)\,\Delta^{k+1} \approx r^k \tag{2.73}$$

where $\Delta^{k+1} = x^{k+1} - x^k$ is the increment of this iteration, that can be approximately solved as:

$$L'U'\Delta^{k+1} \approx r^k \tag{2.74}$$

Then it is used to evaluate x^{k+1}. As in exact LU decomposition, matrices L' and U' depend only on A and not on the right hand side b so they can be pre-evaluated and stored.

The main problem of the classic iterative algorithms is their degradation as the problem size increases. For large scale problems, thousands of iterations are needed to achieve convergence. This can be seen in Fig 2.5, where the residual of the problem model, solved with MSIP and Line-by-line Gauss-Seidel, has been represented versus the number of iterations for different problem sizes. If the residual norm is represented against CPU time, the results are even worse, as their operation count increases with N. The difference between both algorithms decreases for large N. If the problem is to solve accurately a linear equation, like in the case of the pressure correction equation, both are acceptable only for very coarse meshes.

However, classic iterative algorithms are effective *smoothers*: they can effectively reduce the high frequency components of the error. As an example, the norm of the residual obtained using MSIP for a 128×128 problem model, with different p_1 and p_2 parameters has been represented in Fig. 2.6. The rate of decrease of the residual is larger for the case with $p_1 = p_2 = 5$ distribution. Multigrid algorithms allow the smoothers to keep a high rate of residual reduction during all the convergence process.

[8]And then nobody remembers that the cause might be the α parameter, so a lot of human time can be wasted !.

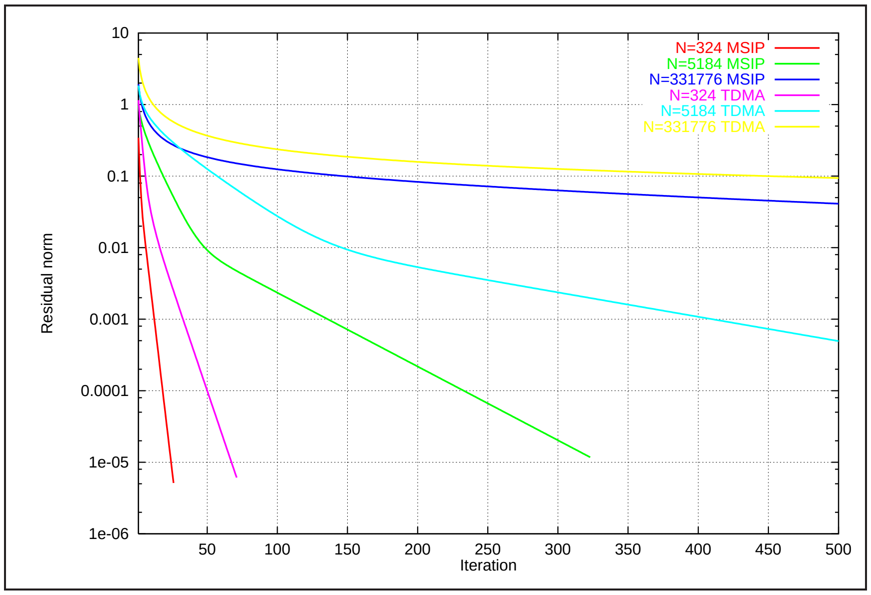

Figure 2.5: Convergence of Line-by-line Gauss Seidel and MSIP algorithms.

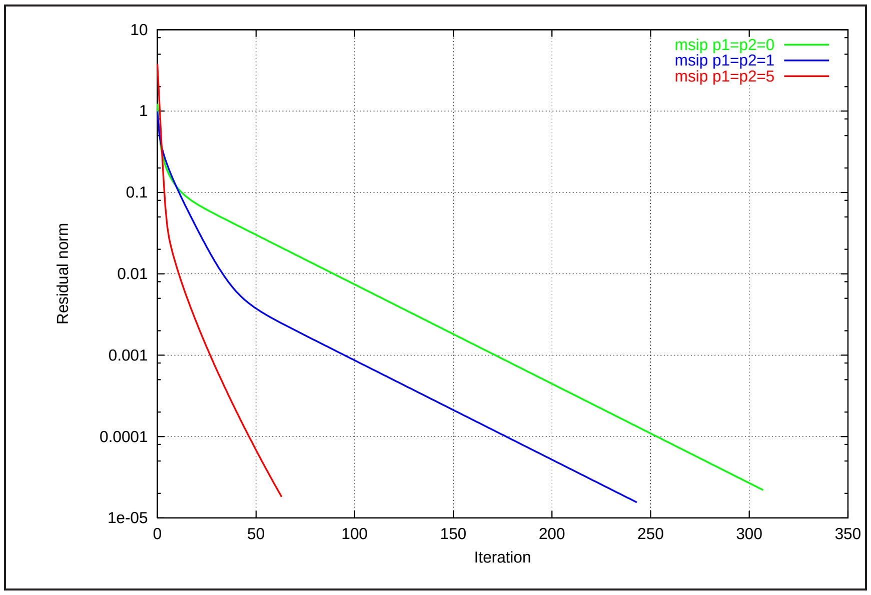

Figure 2.6: Effect of the frequency distribution on the efficiency of MSIP.

2.8 Nomenclature

A	discretization matrix
a	discretization coeff.
b	discretization right-hand-side
c_p	specific heat
B	right hand side matrix
b_{dc}	deferred correction term
B_W	bandwidth
C	mass fraction or concentration
D	diffusion coefficient
F	mass flow rate
f	temporal discretization parameter
Fo	Fourier number
g	gravitational acceleration
L	length
L	lower triangular matrix
N	number of nodes
$\mathbf{u}$	velocity vector
p	dynamic pressure
r	residual
T	temperature
S	source term
$\overline{S}$	averaged source term
$\mathbf{J}$	flux vector
$\mathbf{J}^C$	convective flux vector
$\mathbf{J}^D$	diffusive flux vector
$\mathbf{n}$	normal unit vector
Re	Reynolds number
U	upper triangular matrix
V	volume
$\mathbf{x}$	position vector
x	vector of unknowns in a scalar equation

Greek symbols

α	MSIP cancellation parameter relaxation parameter
Δt	Time step
ϕ	generic unknown
$\mathbf{\Phi}$	vector of all the unknowns
β	thermal volumetric expansion coef.
ϵ	precision
ρ	density
Γ	generalized diffusion coefficient
μ	dynamic viscosity

Subindices

1,2,3	Cartesian components
e	east surface
w	west surface
n	north surface
s	south surface
E	east neighbour
W	west neighbour
N	north neighbour
S	south neighbour
P	discretization node of the CV
WW	far west neighbour
EE	far east neighbour
nb	neighbours

Superindices

0	present
1	future