POLITECNICO DI MILANO

Dipartimento di Scienze e Tecnologie Aerospaziali
(DAER)

Large Eddy Simulation of Turbulent Flows by a Direction Splitting Solver in Complex Geometries

Relatore: Prof. Antonella Abbà
Correlatore: Prof. Franco Auteri

Tesi di laurea di:
Tommaso Zanelli
Matricola: 841366

Anno accademico: 2017 / 2018

Acknowledgements

My deepest gratitude goes to professor Antonella Abbà and professor Franco Auteri of the Dipartimento di Scienze e Tecnologie Aerospaziali, for all their helpfulness and support. Their dedication to this work, especially towards its frenetic end, was extraordinary.
I wish also to thank Luca Baldassarre for making his thesis available to me. The list of all the fellow students and professors that accompanied me through my bachelor and master studies, of all they taught me, of all I owe them, would be too long to report in its entirety, but it is well sculpted in my mind. The same goes for all the friends and family members who supported me in my darkest moments, although I would like to mention Joana, whose never ending optimism and cheerfulness really helped to keep me going.
Obviously, none of this would have ever been possible without my parents shaping me into the person I am, nor without their infinite patience and compassion, especially during these years of studies.

Abstract

The possibility of adding a large eddy simulation turbulence model to a pre-existing solver for the incompressible Navier-Stokes equations is investigated. Preliminary research into turbulence theory and into numerical modelling methods relevant to the solver in use is performed, then a set of Fortran 90 subroutines implementing a Smagorinsky-Lilly closure model is developed. The solver, with the addition of this closure model, is tested performing a simulation of the flow around a sphere at a Reynolds number of 3700. The results are then confronted with those available in the literature.
The solver in use is based on the one originally proposed by Guermond and Minev, which employs finite differences on Cartesian staggered grids, Douglas direction splitting for the momentum equation and an innovative direction splitting technique for the pressure equation. It was developed for optimal scalability on a parallel architecture. The version of the software used for this work differs from the original mainly by the addition of an immersed boundary method. This allows for the placement of a body of arbitrarily complex geometry inside the computational domain, such as the sphere used for the tests peformed.

Contents

List of Figures

List of Tables

Chapter 1

Large Eddy Simulation

1.1 The problem of Turbulence

The behaviour of a Newtonian incompressible fluid is described by the *Navier-Stokes* equations, reported here in indicial notation, in the absence of body forces:

$$\begin{cases} \dfrac{\partial u_i}{\partial t} + \dfrac{\partial\, u_i u_j}{\partial x_j} + \dfrac{1}{\rho}\,\dfrac{\partial p}{\partial x_i} = \nu\,\dfrac{\partial^2 u_i}{\partial x_j \partial x_j}\,, & \text{(1.1a)} \\ \dfrac{\partial u_j}{\partial x_j} = 0\,, & \text{(1.1b)} \end{cases}$$

where summation over repeated indices is understood. Equation (1.1a) represents the conservation of momentum while equation (1.1b) represents the conservation of mass.
The behaviour of their solution is influenced by the Reynolds number: $\mathrm{Re} = \frac{lu}{\nu}$ with l being a characteristic length and u a characteristic speed of the flow considered. Re is a non-dimensional parameter representing the ratio between the advection term $\frac{\partial\, u_i u_j}{\partial x_j}$ and the diffusive term $\nu\frac{\partial^2 u_i}{\partial x_j \partial x_j}$ of (1.1a). For low values of Re the solution is well behaved and the flow is said to be laminar. For greater values of Re, the equations show an increasing sensitivity to small variations in the initial and boundary conditions and the solution starts to behave in a chaotic way. While the deterministic description of the flow provided by (1.1a) and (1.1b) is still valid at high Re, the difficulty of knowing every detail of the initial and boundary conditions with arbitrary precision prompts us to consider instead the solution as a random variable, whose value can not be known univocally since it will be different every time the experiment, measurement or simulation is performed. A random variable can only be described in statistical terms. A flow showing such chaotic behaviour is generally regarded as turbulent.

1.1.1 How turbulence looks like

It is difficult to give a unique definition of turbulence, one offered by S. Corrsin and reported by P. A. Davidson [4, p. 53] is: -"*Incompressible hydrodynamic turbulence is a spatially complex distribution of vorticity which advects itself in a chaotic manner* [...]"- where vorticity is defined as the curl of the velocity field: $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$. Davidson further describes turbuence as a *tangle* of *lumps of vorticity*. Such lumps go by the name of eddy.
The wide variety of scales in which eddies come is a defining characteristic of turbulence. A common view of Turbulence is Richardson's energy cascade: the largest eddies form by subtracting kinetic energy from the mean flow and diverting it to their chaotic motion. Being short-lived structures, they break apart into smaller eddies, then these break apart as well, and so on until the smallest eddies are dissipated by viscosity. The term cascade is used since energy is passed down from the mean flow to eddies of smaller and smaller scale through the breakout process, until molecular viscous processes dissipate it. We can identify three broad ranges of eddy sizes in figure 1.1: the

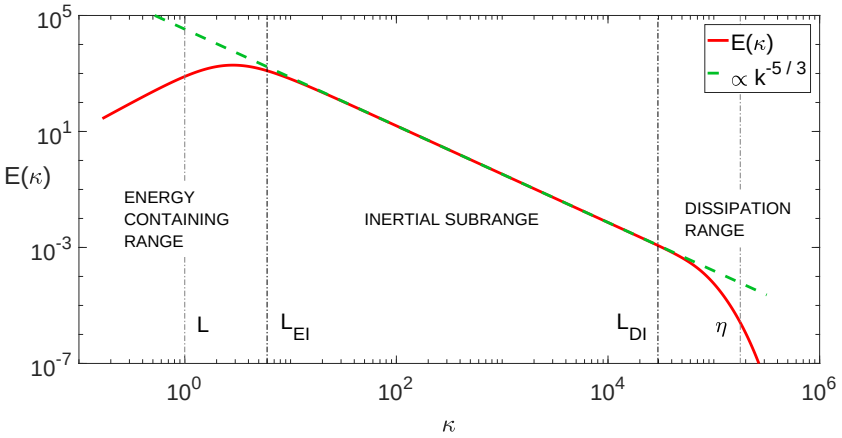


Figure 1.1: Energy Cascade. The Energy spectrum function $E(\kappa)$ is plotted as a function of the wavenumber κ. κ is proportional to the inverse of the size of the eddies, while $E(\kappa)$, defined in (1.9), can be thought of as representative of the energy content of eddies of a certain size.

energy containing range, the inertial subrange and the dissipation range. The first one contains eddies that have sizes and speeds comparable to those of the mean flow, and that are directly influenced by the geometry of the problem. The other two contain eddies with a more universal behaviour. According to an hypothesis first put forward by Kolmogorov, which stands at the base of a great deal of turbulence theory, eddies in the inertial and dissipation range are approximately independent of the geometry and isotropic; according to a further hypothesis, in the inertial range their behaviour is also independent

of viscosity.
Energy must be conserved so the rate of energy entering the cascade, Π, must be equal to the rate of energy exiting it, ε. Π is the energy being passed to the larger eddies, and it can be estimated as $\Pi \sim \frac{u^2}{l/u}$, where u and l are reference values for the velocity and the length scale of the mean flow. ε is the energy being dissipated by viscosity. Since viscosity affects only the smaller eddies, defining their characteristic speed as v and length scale as η, also known as the Kolmogorov scales, it is possible to estimate the rate of dissipation as $\varepsilon \sim \nu \frac{v^2}{\eta^2}$. The Reynolds number of the mean flow is $\mathrm{Re} = \frac{lu}{\nu}$, while the Reynolds number of the smaller eddies is assumed to be $\frac{\eta v}{\nu} \sim 1$, since below this value dissipation is stronger than advection and the energy cascade is arrested.
From the previous relations it is then possible to derive the estimates:

$$\eta \sim l\,\mathrm{Re}^{-3/4} \sim l\,\left(\nu^3/\varepsilon\right)^{1/4}\,, \tag{1.2a}$$

$$v \sim u\,\mathrm{Re}^{-1/4} \sim (\nu\,\varepsilon)^{1/4}\,. \tag{1.2b}$$

1.1.2 The statistical description of turbulence

One characteristic of turbulence is that while the velocity field itself is random, its statistical properties tend to be reproducible from one experiment to another. At the base of the statistical analysis of a turbulent flow is the concept of ensemble average, the average of a flow quantity over multiple experiments. Under the hypothesis that the process is ergodic, such ensemble average is equivalent to the time average:

$$\langle u_i(\boldsymbol{x})\rangle = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} u_i(\boldsymbol{x},\, t)\,\mathrm{d}t\,. \tag{1.3}$$

Since averaging is done over the entire time domain, any time dependence is cancelled. Such averaged velocity field is then meaningful only under the hypothesis that the flow is stationary.
Once an average value for the velocity field is defined, it is possible to define the turbulent fluctuations as:

$$u_i' = u_i - \langle u_i\rangle \tag{1.4}$$

which have, by definition, zero mean. The process of splitting u_i into its well-behaved reproducible component $\langle u_i\rangle$ and its random component u_i', is known as Reynolds' decomposition. It is possible to decompose the average of the kinetic energy of the flow into a component associated with the average velocity field $\langle u_i\rangle$, and one associated with the turbulent fluctuations u_i':

$$\langle \frac{1}{2} u_i u_i\rangle = \frac{1}{2}\left[\langle\langle u_i\rangle\langle u_i\rangle\rangle + \cancel{\langle u_i'\langle u_i\rangle\rangle} + \cancel{\langle\langle u_i\rangle u_i'\rangle} + \langle u_i' u_i'\rangle\right] = \frac{1}{2}\langle u_i\rangle\langle u_i\rangle + \frac{1}{2}\langle u_i' u_i'\rangle\,,$$

The latter is referred to as the turbulent kinetic energy:

$$k = \frac{1}{2} \langle u_i' \, u_i' \rangle \, . \tag{1.5}$$

The correlation function is defined as:

$$R_{ij}(\boldsymbol{x}_1, \, t_1, \, \boldsymbol{x}_2, \, t_2) = \langle u_i(\boldsymbol{x}_1, \, t_1) \, u_j(\boldsymbol{x}_2, \, t_2) \rangle \tag{1.6a}$$

in its most general form, as found, for example, in [15, p. 550]. It is more common to find correlations between either two instants in time or two points in space, such as

$$R_{ij}(\boldsymbol{r}, \, \boldsymbol{x}, \, t) = \langle u_i(\boldsymbol{x}, \, t) \, u_j(\boldsymbol{x} + \boldsymbol{r}, \, t) \rangle \, , \tag{1.6b}$$

which is linked to the turbulent kinetic energy by the relation

$$k = \frac{1}{2} \, R_{ii}(\boldsymbol{r} = 0, \, \boldsymbol{x}, \, t) \, . \tag{1.7}$$

The two-point correlation function's Fourier transform into wavenumber space is:

$$\Phi_{ij}(\boldsymbol{\kappa}, \, \boldsymbol{x}, \, t) = \frac{1}{(2\pi)^3} \iiint R_{ij}(\boldsymbol{r}, \, \boldsymbol{x}, \, t) \, e^{-j \, \boldsymbol{\kappa} \, \boldsymbol{r}} \, \mathrm{d}\boldsymbol{r} \, , \tag{1.8}$$

which can be used to define the energy spectrum function

$$E(\kappa) = \iiint \frac{1}{2} \, \Phi_{ii}(\boldsymbol{\kappa}') \, \delta \left(\left| \boldsymbol{\kappa}' - \kappa \right| \right) \, \mathrm{d}\boldsymbol{\kappa}' = \iint_{S(\kappa)} \frac{1}{2} \, \Phi_{ii}(\boldsymbol{\kappa}) \, \mathrm{d}\boldsymbol{\kappa} \, , \tag{1.9}$$

where $S(\kappa)$ is a sphere of radius κ in wavenumber space. Using (1.7) the turbulent kinetic energy can be expressed as

$$k = \int_0^\infty E(\kappa) \, \mathrm{d}\kappa \, ; \tag{1.10}$$

more generally, $E(\kappa) \, \mathrm{d}\kappa$ is the amount of turbulent kinetic energy associated with motions of size $\sim 1/\kappa$. In the caption to figure 1.1 it was mentioned that $E(\kappa)$ could be thought of as representative of the energy content of eddies of size $\sim 1/\kappa$. While useful for understanding the concept of energy cascade, this is actually an oversimplification textbooks on turbulence generally warn against [4, p. 315]. Eddies are complex structures, and while it is possible to associate a characteristic dimension $r = 1/\kappa$ to them, if one where to compute a single eddy's energy spectrum, they would find it distributed over a wide range of wavenumbers, peaking in the vicinity of κ, rather than entirely concentrated at κ.

1.1.3 The RANS equations and the closure problem

Since the average (1.3) commutes with differentiation, it is possible to take the average of the continuity equation (1.1b) and show that such equation is valid also for the average velocity field and for the fluctuations defined in 1.4.
Taking instead the average of the momentum conservation equation (1.1a) one obtains:

$$\frac{\partial}{\partial x_j}\langle u_i u_j\rangle + \frac{1}{\rho}\frac{\partial \langle p\rangle}{\partial x_i} = \nu \frac{\partial^2 \langle u_i\rangle}{\partial x_j \partial x_j},$$

where the unsteady term disappears as the time derivative of the time average is null. It is possible to retain the unsteady term if the averaging operation is substituted by a time-filtering operation, which limits either the time scale or the frequency bandwidth of the filtered field.
The non-linear term becomes, using Reynolds' decomposition:

$$\langle u_i u_j\rangle = \langle\langle u_i\rangle\langle u_j\rangle\rangle + \cancel{\langle u_i'\langle u_j\rangle\rangle} + \cancel{\langle\langle u_i\rangle u_j'\rangle} + \langle u_i' u_j'\rangle = \langle u_i\rangle\langle u_j\rangle + \langle u_i' u_j'\rangle$$

since the average is an idempotent operator, and, due to linearity, fluctuations have zero mean.
Defining the Reynolds stress tensor as:

$$\tau_{ij}^R = \langle u_i' u_j'\rangle \tag{1.11}$$

and with $\langle \tau_{ij}\rangle = 2\nu \langle S_{ij}\rangle = \nu\left(\frac{\partial\langle u_i\rangle}{x_j} + \frac{\partial\langle u_j\rangle}{\partial x_i}\right)$, it is then possible to write:

$$\begin{cases} \dfrac{\partial \langle u_i\rangle\langle u_j\rangle}{\partial x_j} + \dfrac{1}{\rho}\dfrac{\partial\langle p\rangle}{\partial x_i} = \dfrac{\partial}{\partial x_j}\Big(\langle\tau_{ij}\rangle - \tau_{ij}^R\Big), & \text{(1.12a)} \\ \dfrac{\partial\langle u_j\rangle}{\partial x_j} = 0, & \text{(1.12b)} \end{cases}$$

which are the Reynolds-averaged Navier-Stokes equations, or RANS equations. They describe the behaviour of the average velocity field $\langle u_i\rangle$. They are analogous to the original (1.1a) and (1.1b) but include the additional Reynolds stress tensor (1.11), which adds six more unknowns, since it is symmetric. In the unsteady case, the additional term $\frac{\partial\langle u_i\rangle}{\partial t}$ is included in (1.12a), to obtain the unsteady Reynolds-averaged Navier-Stokes or URANS, that describe the behaviour of the time-filtered velocity field.
It is possible to derive from (1.12a) and (1.12b) additional equations for the components of (1.11), but they would inevitably contain higher order moments in the form $\langle u_i' u_j' u_k'\rangle$, as well as mixed averages such as $\langle p' u_i\rangle$, which would add more unknowns.
While it is possible to derive equations for these additional terms as well, more unknowns will be introduced. Trying to derive a set of equations describing the behaviour of $\langle u_i\rangle$ starting from (1.1a) and (1.1b) alone will always produce more unknowns than equations. This inability to reach a closed set of equations is known as the closure problem of turbulence.

1.2 Turbulence modelling

One way of tackling the problem of turbulence is to simulate all the chaotic motions in a flow up to the smallest scales: this approach is called Direct Numerical Simulation, or DNS. The problem with this approach is that in order to catch the finest turbulent structures the grid elements must have sizes comparable to those of the smallest eddies. From (1.2a) it is apparent that the ratio between the smallest and largest scales decreases as $\mathrm{Re}^{-3/4}$, therefore the number of nodes required for a DNS, in a three-dimensional simulation, would grow as $N \sim \mathrm{Re}^{9/4}$. This is only an estimate of the memory requirement for a single time step of the simulation. The CPU time necessary will probably grow faster with Re since the number of operations required to solve a linear system is proportional to the size of the system only in special cases, such as tridiagonal matrices [17, p. 67], and generally grows faster. Furthermore, the number of time steps required for the simulation will grow as $N \sim \mathrm{Re}^{3/4}$, since the time interval has to be below a certain limit, proportional to the space interval, due to stability and accuracy requirements [4, p. 425]. For these reasons, the DNS approach is feasible only for relatively small values of Re. Most practical engineering applications involve Re having orders of magnitude in the range $10^5 \div 10^7$, with meteorological applications involving Re up to 10^{20} [2, p. 6]. Also, most of the computational effort of a DNS is spent simulating the smallest scales, which are often not of interest for practical applications.

A different approach is that of employing a closure model: the system of equations (1.12a) and (1.12b) is closed by defining a procedure to compute the Reynolds stress tensor (1.11). This model for τ_{ij}^R can not be derived from (1.12a) and (1.12b) alone, but has to be built on additional hypotheses that take into account the nature and physics of the flow under consideration. In simpler eddy viscosity models, τ_{ij}^R is expressed as a function of the spatial derivatives of $\langle u_i \rangle$. Additional information is required to define such function. In more sophisticated Reynolds stress tensor models, a conservation equation for τ_{ij}^R is derived from (1.12a) and (1.12b), as mentioned in 1.1.3, and additional relations are added to express the new unknown terms introduced in this equation as functions of the known terms.

Due to the more regular behaviour of $\langle u_i \rangle$ compared to the velocity field u_i, a much coarser grid than that required by DNS can be used to resolve all its aspects: the resolution of the grid is no longer dependent on the Reynolds number.

All information regarding the turbulent fluctuations u_i', at any scale, is substituted for the closure model. The derivation of a closure model usually involves the determination of a certain number of parameters such that the missing turbulent fluctuations are represented correctly. In general, the values of these parameters have narrow ranges of applicability. Their determination, ideally, requires the information they represent to be known in advance.

The applicability of RANS methodologies to new problems is therefore limited. The parameters can be deduced by similar problems already validated by DNS or experiments, but the reliability of the results remains uncertain. The downside of the RANS approach is that validated results lack in generality, while untested ones lack in reliability.

1.2.1 Eddy viscosity

Eddy viscosity models are based on the hypothesis, first put forward by Boussinesq, that the deviatoric part of the Reynolds Stress Tensor is proportional to the average velocity gradient tensor

$$\langle S_{ij} \rangle = \frac{1}{2} \left(\frac{\partial \langle u_i \rangle}{x_j} + \frac{\partial \langle u_j \rangle}{\partial x_i} \right) . \tag{1.13}$$

This tensor has no isotropic component due to mass conservation (1.12b), but τ_{ij}^R may have one. For this reason the Reynolds Stress Tensor is split into its isotropic part ($\frac{1}{3}\delta_{ij}\tau_{kk}^R$) and deviatoric part, and only the latter is assumed to be proportional to $\langle S_{ij} \rangle$. A turbulent or eddy viscosity coefficient ν_T (in general much greater than the molecular viscosity coefficient ν) is postulated in order to write:

$$\langle u_i' u_j' \rangle - \frac{1}{3}\delta_{ij}\langle u_k' u_k' \rangle = -\nu_T \left(\frac{\partial \langle u_i \rangle}{\partial x_j} + \frac{\partial \langle u_j \rangle}{\partial x_i} \right) = -2\nu_T \langle S_{ij} \rangle . \tag{1.14}$$

It is then possible to rewrite (1.12a) as:

$$\frac{\partial \langle u_i \rangle \langle u_j \rangle}{\partial x_j} + \frac{1}{\rho} \frac{\partial}{\partial x_i} \left(\langle p \rangle + \frac{1}{3}\delta_{ij}\langle u_k' u_k' \rangle \right) = \frac{\partial}{\partial x_j} \left[(\nu + \nu_T) \langle S_{ij} \rangle \right] , \tag{1.15}$$

which, defining a modified pressure $\widetilde{p} = \langle p \rangle + \frac{1}{3}\delta_{ij}\langle u_k' u_k' \rangle$ and an effective viscosity $\nu_{eff} = \nu + \nu_T$ becomes identical to (1.1a), but with a diffusive term of enough magnitude to ensure a well-behaved solution.
The problem then becomes that of determining the value of ν_T.
One approach, suggested by Prandtl, consists in re-writing ν_T as the product of a reference velocity and a reference length:

$$\nu_T = l_m \cdot u^* , \tag{1.16}$$

the latter referred to as the mixing length.
This is in analogy with the estimation of the molecular viscosity in the kinetic theory of gases which, as reported in [4, p. 114], is:

$$\nu = \frac{1}{3}\Lambda \overline{v} , \tag{1.17}$$

where Λ is the mean free path and $\overline{v}$ the root mean square velocity.
One downside of the mixing length theory is that it merely switches the problem to the determination of u^* and l_m. In Prandtl's original formulation for a two-dimensional boundary layer u^* is estimated to be $u^* = l_m \left|\frac{\partial u_x}{\partial y}\right|$, following considerations analogous to those employed to find (1.17) for molecular diffusion, and $l_m = \kappa y$, with y being the distance from the wall and $\kappa = 0.41$ von Kàrmàn's constant. This choice makes the model consistent with the law of the wall (1.45).
The mixing length model is generally considered obsolete and for RANS simulations more sophisticated models are used. These involve expressing ν_T as a function of quantities related to the flow turbulence, such as the turbulent kinetic energy k and the dissipation rate ε. Additional conservation equations are postulated for these quantities. These models usually involve the addition of one or two additional conservation equations. The mixing length model remains of interest as its generalization to a three-dimensional case is the basis for the Smagorinsky-Lilly model frequently employed in Large Eddy Simulation.

1.3 The Large Eddy Simulation approach

As mentioned in 1.1.1, turbulence comes in many scales, and as explained in Section 1.2, the impractical aspect of DNS comes from the need for simulating all scales down to the smallest ones, while the critical difficulty of RANS is finding a closure model that accounts for turbulent fluctuations of any scale with enough precision for a given problem. The Large Eddy Simulation or LES approach attempts to deal with these issues by assuming a separation between the larger scales of the turbulence (the Large Eddies) which are to be simulated directly, and the smaller scales, which are to be accounted for by a closure model. As a consequence, a coarser grid than that required by a DNS is sufficient, since the larger structures only need to be resolved, while the importance of the closure model is reduced compared to a RANS, as it needs to account for the smaller motions only. The method leverages on the first Kolmogorov's hypothesis mentioned in 1.1.1, stating that the smaller structures of turbulence are universal, thus it should be possible for LES closure models to be more general than RANS ones.
The separation between larger and smaller scales is formalized as a spatial filtering operation done on the velocity field:

$$\overline{\boldsymbol{u}}(\boldsymbol{x},\, t) = \iiint G(\boldsymbol{r},\, \boldsymbol{x})\, \boldsymbol{u}(\boldsymbol{x} - \boldsymbol{r},\, t)\, \mathrm{d}\boldsymbol{r}\,, \tag{1.18}$$

where $G(\boldsymbol{r},\, \boldsymbol{x})$ is the Filter Function or Kernel. Such filtering operation allows for the decomposition of the velocity field into its filtered part $\overline{\boldsymbol{u}}(\boldsymbol{x},\, t)$

which, being smoother, corresponds to the larger scales and its residual part

$$\boldsymbol{u}'(\boldsymbol{x},\, t) = \boldsymbol{u}(\boldsymbol{x},\, t) - \overline{\boldsymbol{u}}(\boldsymbol{x},\, t)\,, \tag{1.19}$$

corresponding to the smaller scales. The size of the finest motions that are not filtered out is determined by the filter width Δ, and the grid spacing required to resolve them accurately, h, is generally proportional to it. Very large eddy simulations or VLES are obtained with larger values of Δ, and are essentially analogous to URANS, with the exception that the filtering is spatial instead of temporal.
When Δ is as small as the Kolmogorov scale η, a DNS is essentially performed.
Ideally, the best choice is a Δ slightly smaller than the lower limit of the energy containing range defined in 1.1.1 [21, p. 561].

1.3.1 Filtering

The filtering operation was defined earlier as the convolution of the velocity field and a kernel function. Moving from the physical space to the wavenumber space the filtering operation reduces to the product between the transformed velocity field and the transformed kernel function

$$\widehat{\overline{\boldsymbol{u}}}(\boldsymbol{\kappa},\, t) = \widehat{G}(\boldsymbol{\kappa})\, \widehat{\boldsymbol{u}}(\boldsymbol{\kappa},\, t)\,, \tag{1.20}$$

due to the properties of the Fourier transform.
The kernel function must satisfy the normalization condition

$$\iiint G(\boldsymbol{r},\, \boldsymbol{x})\, \mathrm{d}\boldsymbol{r} = 1\,. \tag{1.21}$$

Filtering is similar to the averaging procedures seen in 1.1.2, as it allows for an analogous decomposition of the velocity field and of the equations of motion. Two important differences are that the filtering operator in general is not idempotent like the average operator, and that the filtered residual is not null. A filter is said to be homogeneous if G depends on $\boldsymbol{r}$ but not on $\boldsymbol{x}$; it is said to be isotropic if it depends only on $|\boldsymbol{r}|$ regardless of direction.
The filtering operation commutes with differentiation in time

$$\overline{\left(\frac{\partial \boldsymbol{u}}{\partial t}\right)} = \frac{\partial\, \overline{\boldsymbol{u}}}{\partial t}\,,$$

but it commutes with differentiation in space only if the filter is homogeneous:

$$\overline{\left(\frac{\partial \boldsymbol{u}}{\partial x_j}\right)} = \left(\frac{\partial\, \overline{\boldsymbol{u}}}{\partial x_j}\right) - \int \frac{\partial G}{\partial x_j}\, \boldsymbol{u}\, \mathrm{d}\boldsymbol{r}\,,$$

in this case, in fact, the term $\frac{\partial G}{\partial x_j}$ is null.
The most commonly used filters, are reported in table 1.1 for the one-dimensional case. The Box filter is essentially a space-averaging over a pre-

	Kernel function $G(r)$	Transfer function $\widehat{G}(\kappa)$
Box filter	$\frac{1}{\Delta}\mathrm{H}\left(\frac{1}{2}\Delta - \lvert r\rvert\right)$	$\frac{\sin\left(\frac{1}{2}\kappa\Delta\right)}{\frac{1}{2}\kappa\Delta}$
Sharp spectral filter	$\frac{\sin\left(\frac{\pi r}{\Delta}\right)}{\pi r}$	$\mathrm{H}(\kappa_c - \lvert\kappa\rvert),\ \kappa_c = \frac{\pi}{\Delta}$
Gaussian filter	$\left(\sqrt{\frac{6}{\pi\Delta^2}}\right) e^{\left(-\frac{6r^2}{\Delta^2}\right)}$	$e^{\left(-\frac{\kappa^2\Delta^2}{24}\right)}$

Table 1.1: Kernel functions and transfer functions of the most commonly used LES filters. H is the Heaviside step function.

defined interval of size Δ; the Sharp spectral filter works similarly but in wavenumber space. The Gaussian filter has the same bell shape of a normal distribution, in both the physical and wavenumber space.
A three-dimensional version of these filters can be easily obtained considering $|\boldsymbol{r}|$ instead of r, plus slight modifications to satisfy the normalization condition (1.21). All such filters are homogeneous and isotropic.
Another possible generalization of the box filter in three dimensions, uniform but not isotropic, is:

$$G(\boldsymbol{r}) = \prod_{j=1}^{3} \frac{1}{\Delta_j}\mathrm{H}\left(\frac{1}{2}\Delta_j - |r_j|\right) \tag{1.22}$$

which corresponds to a spatial average over a rectangular prism of edges Δ_1, Δ_2 and Δ_3.
Usually, the filtering operation is not applied explicitly for deriving the filtered equations of motion nor for formulating the closure model: it is instead implicit in the discretization of the domain. This is equivalent to having a box filter with the shape of the "box" equal to the local cell of the grid. In such case the filtered velocity field is referred to as resolved, while the residual one as sub-grid scale or SGS.
One problem of the implicit filtering is that, while for uniform rectangular structured grids the filter is essentially (1.22), for non-uniform ones the filter is generally no longer homogeneous. This causes filtering and space differentiation to not commute anymore. Anyway, the error introduced assuming commutativity is usually limited, while the difficulties in deriving the filtered equations of motion without this assumption are non-trivial. Therefore, commutativity is usually assumed even on non-uniform grids. This assumption, however, may not be reasonable in the presence of solid boundaries [2, p. 17].

1.3.2 Filtered equations of motion

Applying filtering to equation (1.1b), and assuming commutativity between the filtering operation and spatial differentiation, one obtains:

$$\overline{\left(\frac{\partial u_j}{\partial x_j}\right)} = \frac{\partial \overline{u}_j}{\partial x_j} = 0$$

Which means that both the filtered field $\overline{\boldsymbol{u}}$ and the residual field $\boldsymbol{u}'$ have zero divergence.
Applying filtering to (1.1a), with the same commutativity assumption, brings:

$$\frac{\partial \overline{u}_i}{\partial t} + \frac{\partial \overline{u_i u_j}}{\partial x_j} + \frac{1}{\rho}\frac{\partial \overline{p}}{\partial x_i} = \nu \frac{\partial^2 \overline{u}_i}{\partial x_j \partial x_j} = \frac{\partial \overline{\tau}_{ij}}{\partial x_j}$$

Which is analogous to its Reynolds-averaged equivalent. Once again, the impossibility of obtaining a closed set of equations for the filtered velocity field $\overline{u}_i$ is due to the non-linear term $\overline{u_i u_j}$.
Similarly to the Reynolds stress tensor (1.11), the residual stress tensor or SGS stress tensor is introduced

$$\tau_{ij}^R = \overline{u_i u_j} - \overline{u}_i \overline{u}_j \tag{1.23}$$

by summing and subtracting the space derivative of $\overline{u}_i \overline{u}_j$ from equation (1.3.2). τ_{ij}^R is then decomposed into its isotropic and deviatoric parts:

$$\tau_{ij}^R = \frac{1}{3}\delta_{ij}\tau_{kk}^R + \tau_{ij}^r \,.$$

The isotropic part can be added to the pressure to obtain the modified pressure:

$$\widetilde{p} = \overline{p} + \frac{1}{3}\tau_{kk}^R \,, \tag{1.24}$$

so that the equations of motion for the filtered velocity field can be rewritten as:

$$\begin{cases} \dfrac{\partial \overline{u}_i}{\partial t} + \dfrac{\partial \overline{u}_i \overline{u}_j}{\partial x_j} + \dfrac{1}{\rho}\dfrac{\partial \widetilde{p}}{\partial x_i} = \dfrac{\partial}{\partial x_j}\left(\overline{\tau}_{ij} - \tau_{ij}^r\right), & \text{(1.25a)} \\ \dfrac{\partial \overline{u}_j}{\partial x_j} = 0 \,. & \text{(1.25b)} \end{cases}$$

Alternatively to the decomposition into its isotropic and deviatoric part, another decomposition of the residual stress tensor is:

$$\tau_{ij}^R = \overline{u_i u_j} - \overline{u}_i \overline{u}_j = \mathcal{L}_{ij} + \mathcal{C}_{ij} + \mathcal{R}_{ij} \,, \tag{1.26}$$

which is called Leonard decomposition: it is analogous to Reynolds' decomposition, but does not simplify since the filtering operation is in general not idempotent. The terms

$$\mathcal{L}_{ij} = \overline{\overline{u}_i\overline{u}_j} - \overline{u}_i\overline{u}_j \,, \tag{1.27a}$$

$$\mathcal{C}_{ij} = \overline{\overline{u}_i u'_j} + \overline{u'_i\overline{u}_j} \,, \tag{1.27b}$$

$$\mathcal{R}_{ij} = \overline{u'_i u'_j} \,, \tag{1.27c}$$

are called Leonard stresses, cross stresses and SGS Reynolds stresses respectively. u'_i is the residual, defined here as $u'_i = u_i - \overline{u}_i$.
Such quantities, however, are not Galilean invariant, that is: they are not invariant to rotations or reflections of the coordinates. A different decomposition into components that are Galilean invariant is the one proposed by Germano:

$$\tau^R_{ij} = \overline{u_i u}_j - \overline{u}_i\overline{u}_j = \mathcal{L}^0_{ij} + \mathcal{C}^0_{ij} + \mathcal{R}^0_{ij} \,, \tag{1.28}$$

where:

$$\mathcal{L}^0_{ij} = \overline{\overline{u}_i\overline{u}_j} - \overline{\overline{u}}_i\overline{\overline{u}}_j \,, \tag{1.29a}$$

$$\mathcal{C}^0_{ij} = \overline{\overline{u}_i u'_j} + \overline{u'_i\overline{u}_j} - \overline{\overline{u}}_i\overline{u'}_j - \overline{u'}_i\overline{\overline{u}}_j \,, \tag{1.29b}$$

$$\mathcal{R}^0_{ij} = \overline{u'_i u'_j} - \overline{u'}_i\overline{u'}_j \,. \tag{1.29c}$$

1.3.3 The Smagorinsky-Lilly model

The Smagorinsky-Lilly model is a closure model of the eddy viscosity (1.2.1) type, which works as a three-dimensional generalization of the mixing length model.
The deviatoric part of the SGS stress tensor is assumed to be proportional to the filtered rate of strain $\overline{S}_{ij}$ in the same way as the viscous stress tensor:

$$\tau^r_{ij} = -2\nu_r \cdot \overline{S}_{ij} \,, \tag{1.30}$$

with $\overline{S}_{ij} = \frac{1}{2}\left(\frac{\partial\overline{u}_i}{\partial x_j} + \frac{\partial\overline{u}_j}{\partial x_i}\right)$ and:

$$\nu_r = l_S^2 \cdot \overline{S} \,, \tag{1.31}$$

where $\overline{S}$ is the characteristic filtered rate of strain, defined as $\overline{S} = \sqrt{2\overline{S}_{ij}\overline{S}_{ij}}$. l_S is the Smagorinsky lengthscale, defined as

$$l_S = C_S\Delta \,, \tag{1.32}$$

where Δ is the filter width, in the three-dimensional case it can be estimated as $\sqrt[3]{\Delta_1\Delta_2\Delta_3}$. C_S is called Smagorinsky constant. Its value can be estimated, at least for the case of fully developed turbulence at very high Re, as:

$$C_S \approx 0.17 \,.$$

The procedure for this estimation can be found, for example, in [21, p. 588] and [2, p. 75].
This value tends to be slightly too large to fit experimental or DNS data. Also, near a solid boundary stronger dissipative effects smooth turbulent fluctuations. A damping function may be employed so as progressively reduce C_S to zero at the boundary. One such function, proposed by Van Driest is

$$C_S(y^+) = C_S^0 \left(1 - e^{-\frac{y^+}{A}}\right), \tag{1.33}$$

where $C_S^0 = 0.17$, $A = 25$ is the Van Driest constant and y^+ is the non-dimensional distance from the boundary, defined as $y^+ = y/\delta_v$. δ_v is the viscouls legthscale, defined in 1.3.6.

1.3.4 Backscatter

As mentioned in 1.1.1, energy flows on average from larger to smaller scales of motion. RANS eddy viscosity models capture this behaviour by adding a dissipative term which subtracts energy from the average velocity field, however, statistical likelihood does not equal certainty. It may happen that, locally, energy flows from smaller to larger scales of motion. This phenomenon is called backscatter. While this reversed energy flow can not happen between the residual and the average velocity fields of RANS, due to their statistical nature, it can happen between the SGS and the filtered velocity fields of LES. The Smagorinsky-Lilly model, however, since it introduces an always positive turbulent viscosity coefficient, does not allow for backscatter to take place. The phenomenon of backscatter is invisible to the Smagorinsky-Lilly model. More advanced closure models do allow for backscatter to take place [2, p. 103].

1.3.5 Germano dynamic model

The estimate of the Smagorinsky constant $C_S = 0.17$ is not always the optimal one. Different values have been applied successfully to various types of flow, from $C_S = 0.10$ [6][8] to $C_S = 0.23$ [8]. Also, C_S needs to get to zero near walls and when the flow is laminar [8][21, p. 619]. It would be of interest, in order to refine the Smagorinsky-Lilly model, to couple it with a technique to find values for C_S that are optimal for the flow considered. One such technique is the Germano dynamic model.
The first step of this technique is defining a new filtering operation. This new filter, called test filter, has a width $\widetilde{\Delta}$ that is larger than the width of the original one $\overline{\Delta}$, usually double. The velocity filtered by the test filter is indicated as $\widetilde{\boldsymbol{u}}$, in the same way as the velocity filtered by the original one is indicated as $\overline{\boldsymbol{u}}$. By applying both filters to the velocity field, $\widetilde{\overline{\boldsymbol{u}}}$ is obtained.

The residual stress tensor after the double filtering would be:

$$T_{ij} = \widetilde{\overline{u_i}\,\overline{u}_j} - \widetilde{\overline{u}}_i\,\widetilde{\overline{u}}_j \tag{1.34}$$

in the same way as $\tau_{ij}^R = \overline{u_i\,u}_j - \overline{u}_i\,\overline{u}_j$. Now, applying the test filter to τ_{ij}^R it is possible to define the resolved stress tensor $\mathcal{L}_{ij}$ as

$$\mathcal{L}_{ij} = T_{ij} - \widetilde{\tau_{ij}^R} = \widetilde{\overline{u}_i\overline{u}_j} - \widetilde{\overline{u}}_i\,\widetilde{\overline{u}}_j. \tag{1.35}$$

This relation is known as Germano identity [21, p. 621]. $\mathcal{L}_{ij}$ represents the stresses due to the smaller motions that are still resolved in the filtered velocity field $\overline{\boldsymbol{u}}$ (but not in the field obtained combining both filters). As such, it depends only on $\overline{\boldsymbol{u}}$ and not on the unknown term $\overline{u_i\,u}_j$.
It is possible to re-write the Smagorinsky-Lilly model as

$$\tau_{ij}^r = -2\,c_s\,\overline{\Delta}^2\,\overline{S}\,\overline{S}_{ij}, \tag{1.36}$$

with $c_s = C_s^2$. The Smagorinsky-Lilly model can also be used to estimate the deviatoric part of the residual stress tensor after the double filtering

$$T_{ij}^d = T_{ij} - \frac{1}{3}\,\delta_{ij}\,T_{kk} = -2\,c_s\,\widetilde{\overline{\Delta}}^2\,\widetilde{\overline{S}}\,\widetilde{\overline{S}}_{ij}\,, \tag{1.37}$$

where $\widetilde{\overline{\Delta}}$ is the equivalent width of the two combined filters. With the previous two expressions it is possible to estimate the deviatoric portion of $\mathcal{L}_{ij}$ as:

$$\mathcal{L}_{ij}^S = T_{ij}^d - \widetilde{\tau_{ij}^r}. \tag{1.38}$$

Under the assumption that c_s is uniform, it can be taken out of the filtering operation to write:

$$\mathcal{L}_{ij}^S = c_s\,\left(2\,\overline{\Delta}^2\,\widetilde{\overline{S}\,\overline{S}_{ij}} - 2\,\widetilde{\overline{\Delta}}^2\,\widetilde{\overline{S}}\,\widetilde{\overline{S}}_{ij}\right) = c_s\,M_{ij} \tag{1.39}$$

with $M_{ij} = 2\,\overline{\Delta}^2\,\widetilde{\overline{S}\,\overline{S}_{ij}} - 2\,\widetilde{\overline{\Delta}}^2\,\widetilde{\overline{S}}\,\widetilde{\overline{S}}_{ij}$. The deviatoric part of the resolved stresses estimated by the Smagorinsky-Lilly model, $\mathcal{L}_{ij}^S$, can then be confronted with its actual value

$$\mathcal{L}_{ij}^d = \mathcal{L}_{ij} - \frac{1}{3}\,\delta_{ij}\,\mathcal{L}_{kk}, \tag{1.40}$$

obtained from $\overline{\boldsymbol{u}}$ as in (1.35). The error of the model in estimating $\mathcal{L}_{ij}^d$ is quantified as

$$\epsilon = \left(\mathcal{L}_{ij}^d - \mathcal{L}_{ij}^S\right)^2 = \left(\mathcal{L}_{ij}^d - c_s\,M_{ij}\right)^2\,. \tag{1.41}$$

The error is minimized for

$$c_s = \frac{\mathcal{L}_{ij}\,M_{ij}}{M_{kl}\,M_{kl}}, \tag{1.42}$$

since $\mathcal{L}_{ij}\, M_{ij} = \mathcal{L}^d_{ij}\, M_{ij}$ [21, p. 622].
There is, however, an inconsistency at the base of the procedure above: c_s is assumed uniform when filtering τ^r_{ij} but then is defined in (1.42) as a function of time and space. c_s computed in this way presents large fluctuations [21, p. 623], also due to the possibility of M_{kl} going to zero locally [8]. Such fluctuations render LES computations unstable. It is possible to reduce the fluctuations by computing c_s from average values of $\mathcal{L}_{ij}\, M_{ij}$ and of $M_{kl}\, M_{kl}$. For channel flows, it is possible to average these terms over planes parallel to the solid boundaries [8]. c_s would then have constant values over such planes, but would vary with the distance from the wall. Different local spatial or temporal averages may be used in other situations.
One alternative is not making the assumption that c_s is uniform. In this case it will not be possible to collect c_s in (1.39), since it will remain inside the test filter. The problem then becomes that of finding the function $c_s(\boldsymbol{x})$ that minimizes the error $\left|\mathcal{L}^d_{ij} - \mathcal{L}^S(c_s\,(\boldsymbol{x}))_{ij}\right|$ over the whole domain. This approach is known as localized dynamic model.

1.3.6 Boundary conditions

In the presence of a solid wall, two boundary conditions are applied to the velocity field: the impermeability condition

$$\boldsymbol{u}\,\boldsymbol{n} = 0, \tag{1.43a}$$

and, for a viscous fluid, the no-slip condition

$$\boldsymbol{u} - (\boldsymbol{u}\,\boldsymbol{n})\,\boldsymbol{n} = 0. \tag{1.43b}$$

These, combined, imply that the velocity field is null at the wall.
For large eddy simulations, there are two possible approaches to treating a solid boundary. The first one is near wall-resolution (LES-NWR), in which the near-wall motions are simulated directly. In this case conditions (1.43a) and (1.43b) can be applied directly to the filtered velocity field $\overline{\boldsymbol{u}}$. It has to be noted, however, that to simulate the near-wall motions with enough precision the grid spacing at the wall must be on the order of the viscous lengthscale δ_v, which approximately decreases with Re as

$$\frac{\delta_v}{\delta} = \mathrm{Re}^{-0.88}, \tag{1.44}$$

where δ is the flow lengthscale, a characteristic size of the flow [21, p. 598]. This implies that the number of grid points necessary to resolve the near-wall motions grows as $\mathrm{Re}^{1.76}$, making this approach infeasible for high Re problems. Still, LES-NWR can be seen as a less expensive DNS when only the near wall turbulence is of interest, as it allows for a reduction in grid refinement far from it.

The second approach is near-wall modeling (LES-NWM) in which the near-wall motions are not simulated directly, but modelled instead through suitable boundary conditions. The impermeability condition (1.43a) is generally left unaltered, while the no-slip one is substituted by the condition that the tangential part of the velocity field adheres to a wall function, such as the log law:

$$\overline{u}_{\parallel} = u_\tau \frac{1}{\kappa} \ln\left(y^+\right) + B, \tag{1.45}$$

with $\kappa = 0.41$ (von Kàrmàn's constant), $u_\tau^2 = \tau_{wall}/\rho$, $B = 5.2$ and $y^+ = y/\delta_v$. δ_v is the viscouls legthscale, defined as $\delta_v = \nu/u_\tau$. These conditions can be applied at a certain distance from the boundary, usually they are applied at the first grid node away from the wall.
Alternatively, it is possible to implement hybrid methods: the domain is divided into a region close to solid walls, where a RANS is performed, and a region far from the walls, where a LES is performed. The advantage is that RANS models tend to behave better than LES ones close to the walls. The price is that communication between the results of the two methods at their interfaces is difficult, since the average velocity field of the RANS and the filtered velocity field of the LES have a very different nature.

Chapter 2

Numerical solution

It is seldom possible to obtain a solution to a differential problem in closed form, especially if partial differential equations (PDEs) rather than ordinary differential equations (ODEs) are involved. An approximate numerical solution can be obtained by substituting the continuous differential problem with a finite set of discrete algebraic equations, which can be solved automatically by a computer. In wavenumber (or frequency) space differential and integral relations become algebraic, so a discrete approximation can be obtained by considering a finite number of modes. This spectral approach is very accurate [21, p. 344], but is not considered here. In physical space, it is possible to divide the domain into a finite number of elements, then consider algebraic approximations for the differential or integral operations appearing in the problem on this discrete domain.
It is useful to distinguish between boundary value problems (BVP), in which boundary conditions are given on the frontier of the domain, and initial value problems (IVP), in which all boundary conditions are given at a single point. Equations that model unsteady physical processes, such as the Navier-Stokes equations as presented in Chapter 1, have characteristics of both: they can be seen as an initial value problem in time and a boundary value problem in space. For initial value problems, numerical solution techniques usually involve finite differences; for boundary value problems, one of three numerical schemes is generally used: finite differences, finite elements or finite volumes. The approach pursued here involves finite differences.

2.1 Consistency, stability, convergence

A numerical scheme for solving a differential problem has a finite dimension N, which can be, for example, the number of values used to define the approximate solution. N is also an indication of the computational effort required by the scheme. The main objective of a numerical scheme is to produce an approximated solution so that an increase in N would bring

it closer (in some sense) to the actual solution of the original problem. If this is the case the scheme is said to be convergent. Convergence is linked to two other important properties of a numerical scheme: consistency and stability. Quarteroni [23, p. 4] provides the following formal definitions of these properties. A generic differential problem can be summarized as

$$\mathcal{P}(u,\, g) = 0\,, \tag{2.1}$$

where u is the unknown function, g a set of on which the solution depends and $\mathcal{P}$ a function of u and its derivatives. The discrete problem obtained applying a numerical method to (2.1) is

$$\mathcal{P}_N(u_N,\, g_N) = 0\,, \tag{2.2}$$

u_N and g_N being discrete approximations of u and g, and $\mathcal{P}_N$ being the discrete counterpart of $\mathcal{P}$ obtained substituting the differential operators in it with discrete approximations. Convergence can be defined as:

$$\|u - u_N\| \longrightarrow 0 \qquad \text{for} \qquad N \longrightarrow \infty\,. \tag{2.3}$$

A numerical method is said to be consistent if, for N going to infinity, the real solution u and the real g become solutions of the discrete problem:

$$\mathcal{P}_N(u,\, g) \longrightarrow 0 \qquad \text{for} \qquad N \longrightarrow \infty\,. \tag{2.4}$$

Consistency can be seen an indication of how well the discrete differential operators in $\mathcal{P}_N$ approximate the real ones in $\mathcal{P}$. It is possible to define an order of convergence, and an order of consistency of a numerical method depending on the order of the limits (2.3) and (2.4). The order of convergence may depend on which norm is used for (2.3). Stability relates instead to the discrete problem's sensitivity to variations in the data. A numerical method is stable if, given a perturbation of (2.2):

$$\mathcal{P}_N(u_N + \delta u_N,\, g_N + \delta g_N) = 0\,, \tag{2.5}$$

it is possible to write:

$$\forall\, \varepsilon > 0\, \exists\, \delta > 0 \quad \text{such that} \quad \|\delta g_N\| < \delta \implies \|u_N\| \leq \varepsilon \quad \forall\, N\,. \tag{2.6}$$

Stability is important for keeping in check the unavoidable rounding errors in the data, and to avoid the amplification of computational errors in time marching schemes.
A fundamental result in numerical analysis is the equivalence theorem, which states that if a numerical method is consistent, then it is convergent if and only if it is stable [23, p. 5]. This result is useful since consistency and stability are generally easier to prove than convergence. However, sometimes it is necessary to study the stability of the method directly. Other times this is excessively difficult, and the convergence of the method has to be tested empirically on problems whose solution is known in advance.

2.2 Finite Differences

In order to build a numerical scheme for approximating the solution to a differential problem, it is necessary to find a discrete substitute for derivation. For sufficiently regular functions, and for sufficiently small h, the expressions

$$\frac{\mathrm{d}u}{\mathrm{d}x} \approx \frac{u\left(x+h\right)-u\left(x\right)}{h}\,, \tag{2.7a}$$

$$\frac{\mathrm{d}u}{\mathrm{d}x} \approx \frac{u\left(x\right)-u\left(x-h\right)}{h}\,, \tag{2.7b}$$

can be used. In fact, the definition of derivative is the limit of their right-hand side for $h \to 0$. (2.7a) and (2.7b) are called forward and bacward finite differences respectively. It is important to know how accurately they approximate $\frac{\mathrm{d}u}{\mathrm{d}x}$. The Taylor series expansions of u around x are:

$$u(x+h) = u(x) + \frac{\mathrm{d}u}{\mathrm{d}x}\,h + \frac{1}{2}\,\frac{\mathrm{d}^2u}{\mathrm{d}x^2}\,h^2 + \frac{1}{6}\,\frac{\mathrm{d}^3u}{\mathrm{d}x^3}h^3 + \frac{1}{24}\,\frac{\mathrm{d}^4u}{\mathrm{d}x^4}h^4 + ... \tag{2.8a}$$

$$u(x-h) = u(x) - \frac{\mathrm{d}u}{\mathrm{d}x}\,h + \frac{1}{2}\,\frac{\mathrm{d}^2u}{\mathrm{d}x^2}\,h^2 - \frac{1}{6}\,\frac{\mathrm{d}^3u}{\mathrm{d}x^3}h^3 + \frac{1}{24}\,\frac{\mathrm{d}^4u}{\mathrm{d}x^4}h^4 - ... \tag{2.8b}$$

By rearranging the terms in (2.8a) and (2.8b) it is possible to express the errors committed by (2.7a) and (2.7b) as $\frac{1}{2}\,\frac{\mathrm{d}^2u}{\mathrm{d}x^2}\,h + \frac{1}{6}\,\frac{\mathrm{d}^3u}{\mathrm{d}x^3}h^2 + ...$ and $\frac{1}{2}\,\frac{\mathrm{d}^2u}{\mathrm{d}x^2}\,h - \frac{1}{6}\,\frac{\mathrm{d}^3u}{\mathrm{d}x^3}h^2 + ...$ respectively. For $h \to 0$ the dominant term in the error is, using the big-o notation, $\mathrm{O}(h)$. This means the error decreases linearly with decreasing h. This approximation is said to be first order accurate. Subtracting (2.8a) and (2.8b) and dividing the result by $2\,h$ expression

$$\frac{\mathrm{d}u}{\mathrm{d}x} = \frac{u(x+h)-u(x-h)}{2\,h} + \mathrm{O}\left(h^2\right) \tag{2.9}$$

is found. (2.9) is known as centred finite difference. It is a second order accurate approximation for the derivative of $u(x)$, since the dominant term in the error is $\mathrm{O}\left(h^2\right)$. Summing (2.8a) and (2.8b) instead, and dividing the result by h^2, a second order centred finite difference approximation for the second order derivative of $u(x)$ can be obtained:

$$\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = \frac{u(x+h)-2\,u(x)+u(x-h)}{h^2} + \mathrm{O}\left(h^2\right)\,. \tag{2.10}$$

More generally, a finite difference approximation can be obtained for the derivative of any order, with any order of accuracy, at any point, by combining a sufficient number of Taylor expansions around it. For example, it is possible to derive non centred second order accurate approximations of the first order derivative of $u(x)$ in x:

$$\frac{\mathrm{d}u}{\mathrm{d}x} = \frac{-3\,u(x)+4\,u(x+h)-u(x+2\,h)}{2\,h} + \mathrm{O}\left(h^2\right)\,, \tag{2.11a}$$

$$\frac{\mathrm{d}u}{\mathrm{d}x} = \frac{u(x) - 4\,u(x-h) + 3\,u(x-2\,h)}{2\,h} + \mathrm{O}\left(h^2\right)\,. \tag{2.11b}$$

The points at which the function is evaluated need not be uniformly spaced. More general expressions for centred finite difference approximations to the first and second order derivatives are:

$$\left.\frac{\mathrm{d}u}{\mathrm{d}x}\right|_{x_i} \approx \frac{1}{h_i + h_{i-1}} \left[\frac{h_i\,(u(x_i) - u(x_{i-1}))}{h_{i-1}} + \frac{h_{i-1}\,(u(x_{i+1}) - u(x_i))}{h_i}\right], \tag{2.12a}$$

$$\left.\frac{\mathrm{d}^2u}{\mathrm{d}x^2}\right|_{x_i} \approx \frac{2}{h_i + h_{i-1}} \left[\frac{u(x_{i+1}) - u(x_i)}{h_i} - \frac{u(x_i) - u(x_{i-1})}{h_{i-1}}\right], \tag{2.12b}$$

as reported by Keating [13, p. 14], with $h_i = x_{i+1} - x_i$ and $h_{i-1} = x_i - x_{i-1}$. While (2.12a) is second order accurate, (2.12b) is only first order. It regains second order accuracy if the grid is quasi-uniform, that is, if the grid spacing varies sufficiently slowly along x [13].

The number of different values of u necessary to compute a finite difference approximation increase with the order of the derivative and with the order of accuracy desired. The coefficients of the finite difference are obtained solving a linear system that combines the Taylor series of u so that all "unwanted" derivatives cancel out. This linear system, however, becomes very badly conditioned with large numbers of nodes [17, p. 11].

The concept of finite differences presented above can easily be extended to multidimensional derivatives. For example, an approximation to the two-dimensional Laplacian operator is:

$$\nabla^2 u\big|_{x_i,y_j} \approx \frac{u(x_{i-1},y_i) + u(x_i,y_{i-1}) - 4\,u(x_i,y_i) + u(x_{i+1},y_i) + u(x_i,y_{i+1})}{h^2}\,, \tag{2.13}$$

assuming equal spacing h in all directions. The set of points used to compute

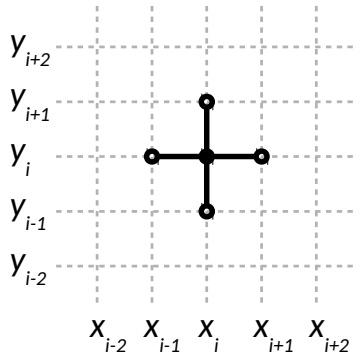


Figure 2.1: The five-point stencil.

a finite difference is referred to as stencil. The one used for (2.13) is called

the five-point stencil [17, p. 61], shown in figure 2.1.
When using finite differences, it is customary to use subscripts to identify the position in space and superscripts to indicate the instant in time [17, p. 120]. For example, $u^n_{i,\,j,\,k}$ would be the value of the approximate solution at position $\{x_i,\, y_j,\, z_k\}$ and at time t^n.

2.2.1 Initial value problems

An initial value problem takes the form

$$\begin{cases} \dfrac{\mathrm{d}u}{\mathrm{d}t} = f(u(t),\, t)\,, \qquad t > t_0 & \text{(2.14a)} \\ u(t_0) = u_0 & \text{(2.14b)} \end{cases}$$

A time-marching numerical scheme to find an approximate solution to it can be obtained by substituting the derivative of u on the left-hand side with either (2.7a) or (2.7b):

$$\frac{u^{n+1} - u^n}{\Delta t} = f\left(u^n,\, t^n\right)\,, \qquad u^0 = u_0\,, \tag{2.15a}$$

$$\frac{u^{n+1} - u^n}{\Delta t} = f\left(u^{n+1},\, t^{n+1}\right)\,, \qquad u^0 = u_0\,. \tag{2.15b}$$

These solution methods are called forward Euler and bacward Euler methods respectively. They are both first order accurate in time. (2.15a) is an explicit method, as the unknown term u^{n+1} can be expressed immediately as a function of u's value at the previous step, while (2.15b) is an implicit method, as the unknown term appears also inside the forcing term f. Implicit schemes are usually more demanding computationally, but they tend to present advantages in terms of stability requirements [27, p. 91], [13, p. 28].
A scheme that is second order accurate in time is the so-called leapfrog method

$$\frac{u^{n+1} - u^{n-1}}{2\,\Delta t} = f\left(u^n,\, t^n\right)\,, \tag{2.16}$$

which uses a centred difference for the time derivative. The leapfrog method is an example of a multi-step method, since it references the value of u not only at t^n but also at the previous step. A multi-step method has certain disadvantages in terms of memory, since additional terms have to be kept in storage, and it is not obvious how to initialize them, since for the first few time steps they refer to undefined values of u. Another second order accurate scheme, which is implicit but does not require any additional values of u, is the trapezoidal [17, p. 121] or Crank-Nicholson [27, p. 4] method:

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}\left[f\left(u^{n+1},\, t^{n+1}\right) + f\left(u^n,\, t^n\right)\right]\,, \tag{2.17}$$

where the centred difference in time is evaluated at $t^{n+\frac{1}{2}}$, and the value of f at $t^{n+\frac{1}{2}}$ is approximated with an average between its values at n and $n+1$. This averaging is second order accurate in time, as long as f depends sufficiently regularly from u and t [17, p. 116]. Of course, it is possible to use higher order approximations for the time derivative to obtain more accurate multi-step schemes, either implicit or explicit. An alternative route to obtaining higher order accurate multi-step schemes is looking for an approximation of the integral of f, rather than one of the derivative of u. Two important families of methods are obtained in this way: the explicit Adams-Bashforth methods and the implicit Adams-Moulton methods. The two- and three-step Adams-Bashforth methods are:

$$u^{n+1} = u^n + \frac{\Delta t}{2}\left[3\, f\left(u^n,\, t^n\right) - f\left(u^{n-1},\, t^{n-1}\right)\right] , \tag{2.18a}$$

$$u^{n+1} = u^n + \frac{\Delta t}{12}\left[23 f\left(u^n, t^n\right) - 16 f\left(u^{n-1}, t^{n-1}\right) + 5 f\left(u^{n-2}, t^{n-2}\right)\right] . \tag{2.18b}$$

The two-step Adams-Moulton method is:

$$u^{n+1} = u^n + \frac{\Delta t}{12}\left[5 f\left(u^{n+1}, t^{n+1}\right) + 8 f\left(u^n, t^n\right) - f\left(u^{n-1}, t^{n-1}\right)\right] . \tag{2.19}$$

2.15a is the one-step Adams-Bashforth method, 2.17 is the one-step Adams-Moulton method [17, p. 132]. N-step Adams-Bashforth schemes have accuracy of order N [24, p. 401] while N-step Adams-Moulton schemes have accuracy of order $N+1$ [24, p. 402].
An alternative to multi-step schemes is multi-stage schemes [17, p. 124], of which the more common are the Runge-Kutta methods. These methods use multiple estimations of the solution during a single time step, progressively refining the value at the end of the step. A simple example is the two-stage method:

$$u^* = u^n + \frac{\Delta t}{2} f\left(u^n,\, t^n\right) , \tag{2.20a}$$

$$u^{n+1} = u^n + \Delta t\, f\left(u^*,\, t^{n+\frac{1}{2}}\right) , \tag{2.20b}$$

Where (2.20a) estimates the value of u at $t^{n+\frac{1}{2}}$ with a forward Euler step as in (2.15a), then (2.20b) uses this estimate to find u^{n+1} with a leapfrog step, as in (2.16). The method is second order accurate in time [17, p. 125]. The general form of a q stage Runge-Kutta method is

$$u_i^* = u^n + \Delta t \sum_{j=1}^{q} a_{ij}\, f\left(u_j^*,\, t^n + c_j\, \Delta t\right) , \qquad i = 1, ..., q \tag{2.21a}$$

$$u^{n+1} = u^n + \Delta t \sum_{j=1}^{q} b_j\, f\left(u_j^*,\, t^n + c_j\, \Delta t\right) . \tag{2.21b}$$

In this form the method is fully implicit, as each stage depends on all the unknowns u_i^*. If the summation on the left-hand side of (2.21a) is stopped at i, then the method is said to be diagonally implicit, as each stage is implicit only in its unknown. If the summation is stopped at $i-1$, then each stage is explicit. Parameters a_{ij}, b_j, c_j must adhere to certain conditions for the method to be consistent [17, p. 126]:

$$\sum_{j=1}^{q} a_{ij} = c_i \,, \qquad i = 1, ..., q \tag{2.22}$$

$$\sum_{j=1}^{q} b_j = 0 \,. \tag{2.23}$$

Further conditions on the parameters exist to ensure the order of accuracy of the method.

2.2.2 Domain discretization

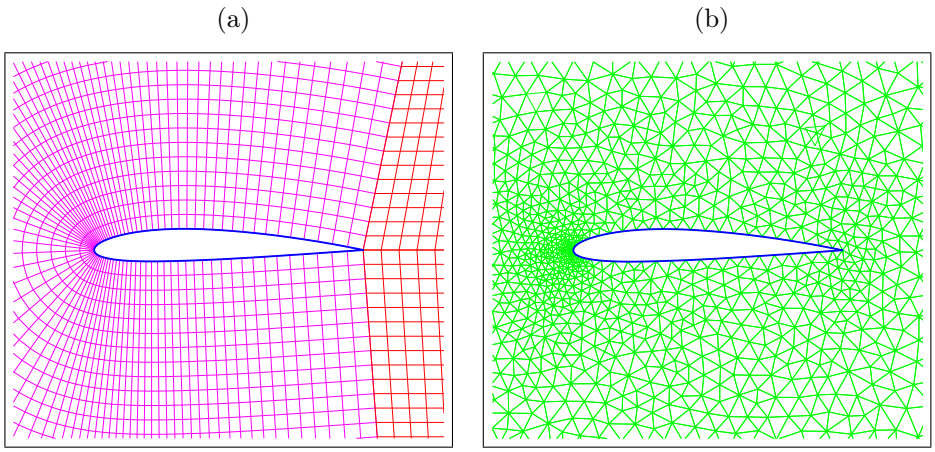


Figure 2.2: (a) A two-dimensional structured grid adapted around a NACA 2412 airfoil. (b) A two-dimensional unstructured grid made of triangular elements around a NACA 2412 airfoil.

In order to obtain a discrete representation of the computational domain it is necessary to divide it into a finite number of parts. This subdivision is referred to as grid or mesh. Grids can be structured or unstructured [23, p. 145]. Structured grids (figure 2.2a) are made of quadrangular elements, ordered so that it is possible to map their positions to sequences of integers. They are memory efficient, as they only require a set of indexes to identify a cell. Unstructured grids (figure 2.2b) are generated filling the domain with geometric elements that have no pre-determined order. These elements can

have various shapes, the simplest one being a triangle, and vary in size, although they must satisfy certain regularity constraints. Unstructured grids are very versatile, they can fill complex geometries and be refined locally, but require explicit storage of the position of each one of their nodes, so they are not as memory efficient as structured grids. Structured grids can be adapted to general domain geometries by applying a coordinate transformation to a Cartesian grid, but they require the domain boundaries to be sufficiently regular. They allow for local refinements, but only in "bands" rather than at specific points. Multiple structured grids can be used to fill more complex domains.
It is common to associate finite difference methods (FDM) to structured grids and finite element methods (FEM) with unstructured grids. However, finite element methods can be implemented on structured grids. It is also possible to implement a finite difference scheme on an unstructured grid, although this is rarely done in practice [27, p. 383].
There is another approach to the inclusion of a complex boundary in a structured grid. Instead of transforming the grid's geometry, the effects of the boundary (the boundary conditions) are imposed inside the computational domain. A boundary treated in this way is said to be immersed. This approach allows for the retention of the grid's orthogonal Cartesian structure, in which finite differences can be expressed in a very simple and efficient form.

2.2.3 Immersed boundaries

One possible approach to recreating the effects of complex boundaries inside the computational domain is boundary fitting. It consists of finding where the immersed boundary crosses the Cartesian grid, and moving the first node inside the boundary at the crossing point. Finite difference expressions in the form of (2.12a) and (2.12b) are modified locally to account for the reduced intervals. They, however, no longer commute, that is, Schwarz's theorem no longer applies to the discrete operators [13, p. 16]. Roache advises against this approach [27, p. 322].
A different approach consists in recreating the values of the velocity field at the immersed boundary from the surrounding points of the Cartesian grid without altering its shape, and using this recreation to impose the boundary conditions.
The immersed geometry is generally provided as an unstructured mesh of elements. For each element, the point of the Cartesian grid that is closest to it can be found, and a set of points around it can be used for the reconstruction of the desired quantity at the boundary through a form of interpolation to be defined. An immersed boundary procedure can be explicit, when the velocity at the boundary is extrapolated from known values and used to apply a correction on the field, for example, in the form of a forcing term.

It can be implicit, when the reconstruction procedure is used to translate the boundary conditions on the immersed geometry into conditions on the surrounding Cartesian points, as is done in the ghost cell methods used in [18] and in [26].
A list of approaches to immersed boundaries can be found in [13, p. 3].

2.2.4 Solution of boundary value problems

On a discrete domain, functions can be represented by their values at the nodes of the grid. These values can be arranged into vectors. Spatial derivative operators can be obtained combining the coefficients of finite difference expressions like (2.7a), (2.10) or (2.12a) into matrices. Derivatives are computed multiplying these matrices with the vector representations of the functions. These matrices are singular by construction.
A generic boundary value problem, such as the elliptic equation

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} = f\,, & x \in [0,\,1]\,, \qquad (2.24a) \\ u(0) = u_0\,, \qquad u(1) = u_f\,, & \qquad (2.24b) \end{cases}$$

can be turned into the linear system of equations

$$D_{ij}\, u_j = f_i\,, \qquad (2.25)$$

where $\{D_{ij}\}$ is the matrix representing the operator $\frac{\partial^2}{\partial x^2}$, modified to account for boundary conditions. If the problem is well posed, this modified matrix should be invertible. There are many different solution techniques for linear systems of equations. The brute force approach, Gaussian elimination, requires, in the more general case, a number of operations that is $\mathrm{O}\left(N^3\right)$ [17, p. 67], with N being the size of the problem. It can be noted, however, that for a one-dimensional problem the matrix $\{D_{i,\,j}\}$ is banded. Specifically using centred finite differences in the form (2.10), it is tridiagonal. For a tridiagonal matrix (or, more in general, a banded matrix with bandwidth $\ll N$, [24, p. 96]) there is a technique called Thomas algorithm [24, p. 95], that requires a number of operations that is only $\mathrm{O}\left(N\right)$ [17, p. 67]. However, for multi-dimensional boundary value problems, in the presence of derivatives along different directions, it is not possible to obtain a solution matrix with a bandwidth $\ll N$, regardless of the order chosen for the solution vector. The matrix would still be sparse, since each finite difference requires a limited number of adjacent points, and many efficient techniques exist for the solution of a system of this kind. However, Thomas algorithm on a limited bandwidth matrix remains one of the simplest, most efficient and scalable [9] techniques available. It would be advantageous to reformulate the problem so that only one-directional derivative operators need to be solved at one time.

2.2.5 Alternating direction implicit method

A parabolic equation in the form

$$\frac{\partial u}{\partial t} = Lu\,, \tag{2.26}$$

where L a generic differential operator, can be solved in two or more passages through a technique called operator splititng. If it is possible to write

$$Lu = L_1 u + L_2 u\,,$$

then the solution can be found in two passages:

$$\frac{u^* - u^n}{\Delta t} = L_1 u^*\,, \tag{2.27a}$$

$$\frac{u^{n+1} - u^*}{\Delta t} = L_2 u^{n+1}\,. \tag{2.27b}$$

Solving 2.27b for u^* and inserting it into 2.27a brings:

$$\frac{u^{n+1} - u^n}{\Delta t} = L_1 u^{n+1} + L_2 u^{n+1} - \Delta t L_1 L_2 u^{n+1}\,,$$

which is a forward Euler approximation for (2.26), plus a term proportional to Δt. As a consequence, solving (2.27a) and (2.27b) at each time step produces an approximate solution to (2.26) that is first order accurate in time. The same approach can be used to obtain more accurate schemes, for a variety of time-marching problems. Considering the heat equation

$$\frac{\partial u}{\partial t} = \alpha\, \nabla^2 u\,, \tag{2.28}$$

in two dimensions, the alternating direction implicit (ADI) method is an implicit operator splitting scheme that allows for one-directional problems only to be solved at each step:

$$\frac{u^* - u^n}{\Delta t} = \frac{\alpha}{2}\left(\frac{\delta^2 u^*}{\delta x^2} + \frac{\delta^2 u^n}{\delta y^2}\right)\,, \tag{2.29a}$$

$$\frac{u^{n+1} - u^*}{\Delta t} = \frac{\alpha}{2}\left(\frac{\delta^2 u^*}{\delta x^2} + \frac{\delta^2 u^{n+1}}{\delta y^2}\right)\,. \tag{2.29b}$$

$\frac{\delta^2}{\delta x^2}$ and $\frac{\delta^2}{\delta y^2}$ are the finite difference approximations of the spatial derivative operators. Rearranging the terms, (2.29a) and (2.29b) can be re-written as

$$\left(\boldsymbol{I} - \frac{\alpha\,\Delta t}{2}\,\frac{\delta^2}{\delta x^2}\right) u^* = \left(\boldsymbol{I} + \frac{\alpha\,\Delta t}{2}\,\frac{\delta^2}{\delta y^2}\right) u^n\,, \tag{2.30a}$$

$$\left(\boldsymbol{I} - \frac{\alpha\,\Delta t}{2}\,\frac{\delta^2}{\delta y^2}\right) u^{n+1} = \left(\boldsymbol{I} + \frac{\alpha\,\Delta t}{2}\,\frac{\delta^2}{\delta x^2}\right) u^*\,. \tag{2.30b}$$

If centred finite differences were used, both matrices $\left(\boldsymbol{I} - \frac{\alpha\,\Delta t}{2}\,\frac{\delta^2}{\delta x^2}\right)$ and $\left(\boldsymbol{I} - \frac{\alpha\,\Delta t}{2}\,\frac{\delta^2}{\delta y^2}\right)$ can be rearranged to be tridiagonal since they involve derivatives in one direction only. As mentioned in 2.2.4, this allows for a particularly efficient solution. By combining (2.30a) and (2.30b) to bypass u^*, it can be shown that the method is second order accurate in time. The method is also unconditionally stable [27, p. 101]. It can be shown that u^* is a first order accurate approximation of the value of u at $t^{n+\frac{1}{2}}$ [17, p. 199]. This is advantageous, since equation (2.30a) requires boundary conditions, and the values of $u^{n+\frac{1}{2}}$ can be used. According to LeVeque, by using these boundary conditions the second order accuracy of the complete method is retained despite u^* being only a first order accurate approximation for $u^{n+\frac{1}{2}}$ [17, p. 199]. The extension of (2.29) to a three-dimensional situation is not obvious. A three-dimensiononal ADI method was first proposed by Douglas [7] in 1962:

$$\frac{u^* - u^n}{\Delta t} = \frac{\alpha}{2}\,\frac{\delta^2}{\delta x^2}\,(u^* + u^n) + \alpha\,\frac{\delta^2 u^n}{\delta y^2} + \alpha\,\frac{\delta^2 u^n}{\delta z^2}\,, \tag{2.31a}$$

$$\frac{u^{**} - u^n}{\Delta t} = \frac{\alpha}{2}\,\frac{\delta^2}{\delta x^2}\,(u^* + u^n) + \frac{\alpha}{2}\,\frac{\delta^2}{\delta y^2}\,(u^{**} + u^n) + \alpha\,\frac{\delta^2 u^n}{\delta z^2}\,, \tag{2.31b}$$

$$\begin{aligned}\frac{u^{n+1} - u^n}{\Delta t} = &\frac{\alpha}{2}\,\frac{\delta^2}{\delta x^2}\,(u^* + u^n) + \frac{\alpha}{2}\,\frac{\delta^2}{\delta y^2}\,(u^{**} + u^n) + \\ &+ \frac{\alpha}{2}\,\frac{\delta^2}{\delta z^2}\,\left(u^{n+1} + u^n\right).\end{aligned} \tag{2.31c}$$

This method is, again, second order accurate and unconditionally stable [27, p. 104]. In this case, each intermediate step is a first order accurate approximation of the value of u at the end of the step. As such, the values of u^{n+1} at the boundaries can be used for boundary conditions to (2.31a) and (2.31b). However, this might result in a loss of accuracy. A more detailed analysis of the problem of boundary conditions for intermediate steps in ADI schemes can be found in [16].

In the presence of an additional forcing term in equation (2.28)

$$\frac{\partial u}{\partial t} = \alpha\,\nabla^2 u + f\,(u,\,t)\,,$$

it is sufficient to add $f^{n+\frac{1}{2}}$ to each step of (2.31). With some algebra, [10][13, p. 42], the scheme becomes:

$$\frac{u^* - u^n}{\Delta t} = \alpha\,\nabla^2 u^n + f^{n+\frac{1}{2}}\,, \tag{2.32a}$$

$$\frac{u^{**} - u^*}{\Delta t} = \frac{\alpha}{2}\,\frac{\delta^2}{\delta x^2}\,(u^{**} - u^n)\,, \tag{2.32b}$$

$$\frac{u^{***} - u^{**}}{\Delta t} = \frac{\alpha}{2}\,\frac{\delta^2}{\delta y^2}\,(u^{***} - u^n)\,, \tag{2.32c}$$

$$\frac{u^{n+1} - u^{***}}{\Delta t} = \frac{\alpha}{2}\,\frac{\delta^2}{\delta z^2}\,\left(u^{n+1} - u^n\right). \tag{2.32d}$$

The form of the scheme in (2.31) is recovered by summing each step of (2.32) to all its previous steps.
(2.32) can be adapted straightforwardly to the treatment of the diffusion terms for each component of the momentum conservation equation (2.33a), collecting all other terms in f. Viscous diffusive terms have more demanding stability requirements than the advection terms when treated explicitly [13, p. 29]. Treating them with an implicit, unconditionally stable method is advantageous.

2.3 Numerical solution of the incompressible Navier-Stokes equations

When dealing with numerical methods for their approximate solutions, equations (1.1a) and (1.1b) are generally presented in vector notation:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \left(\boldsymbol{u}\, \boldsymbol{u}^T \right) + \nabla p - \dfrac{1}{\mathrm{Re}} \nabla^2 \boldsymbol{u} = \boldsymbol{f}\,, & \text{(2.33a)} \\ \nabla \cdot \boldsymbol{u} = 0\,, & \text{(2.33b)} \end{cases}$$

where $\boldsymbol{u} = \{u,\, v,\, w\}^T$ and $\boldsymbol{f}$ is a generic forcing term. All quantities have been made non-dimensional considering a reference velocity U_0, a reference length $\mathcal{L}_0$, a reference pressure $q = \rho U_0^2$ and a reference time scale $\tau = \mathcal{L}_0 / U_0$.
Equations (2.33a) and (2.33b) are defined over a domain $\Omega \times [0,\, T]$, with $\Omega \in \mathbb{R}^d$, $d = 2,\, 3$. The system of equations (2.33) will have to be completed with initial and boundary conditions. The initial condition for $\boldsymbol{u}$ will be in the form:

$$\boldsymbol{u}|_{t=0} = \boldsymbol{u}_0 \qquad \text{in } \Omega\,. \tag{2.34}$$

Boundary conditions can be

$$\begin{cases} \boldsymbol{u} = \boldsymbol{b} & \text{in } \Gamma_D \times [0,\, T]\,, & \text{(2.35a)} \\ \dfrac{1}{\mathrm{Re}} \nabla \boldsymbol{u} \cdot \boldsymbol{n} - p\, \boldsymbol{n} = \boldsymbol{\sigma} & \text{in } \Gamma_N \times [0,\, T]\,, & \text{(2.35b)} \end{cases}$$

where Γ_D and Γ_N are two non-overlapping portions of the domain boundary $\partial\Omega$. Dirichlet boundary conditions in the form (2.35a) are also called essential, they are used to impose conditions (1.43a) and (1.43b) at a solid wall, or, for example, to prescribe a velocity profile at an inlet. Neumann boundary conditions in the form (2.35b) define a surface force $\boldsymbol{\sigma}$ acting on the fluid at the boundary. If Neumann boundary conditions only are prescribed, the problem is underdetermined. For the pressure the situation is less clear. Pressure appears only through its gradient in equation (2.33a), so it is not determined uniquely. It can be determined by artificially setting its value at a point, or by adding the condition that its average over the domain

is null [23, p. 447]
In three dimensions, existence and uniqueness of strong solutions for 2.33a and 2.33b have been formally proven to exist only for very small times [23, p. 444]. Nonetheless, numerical solutions have been routinely computed for decades.

2.3.1 Staggered grids

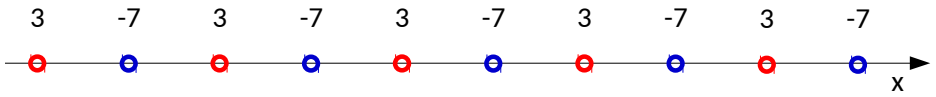


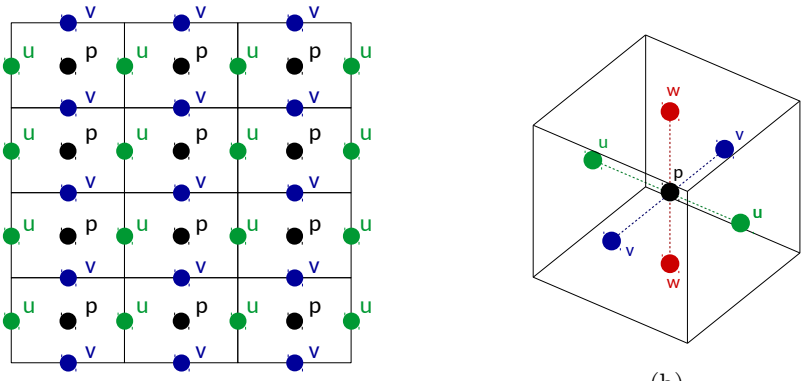
Figure 2.3: Values arranged in an alternating pattern along the x axis.

In the centred finite difference (2.9), the value of the function at the point in which the derivative is evaluated, is absent. This means that applying (2.9) to values arranged in an alternating pattern such as that in figure 2.3 would produce a null derivative, despite its obvious non-uniformity. The same situ-

Figure 2.4: Values arranged in a checkerboard pattern in two dimensions.

ation can happen in more than one dimension, where checkerboard patterns such as that in figure 2.4 would be invisible to the (2.9) operator [20, p. 116]. The problem is that this kind of pattern may emerge in the pressure field during a time-marching simulation, without affecting the momentum equation, which only contains first order derivatives of it. A similar problem may affect the continuity equation (2.33b).
A remedy for this particular kind of instability is the placement of flow variables on different grids, positioned so that first order derivative centred finite differences are evaluated between two adjacent nodes. These grids are said to be staggered, the velocity components and the pressure are positioned as

in figure 2.5. The use of staggered grids for the solution of the equations of motion (2.33) was first suggested by Harlow and Welch for their MAC (Marker and Cell) method [12]. The three components of the momentum



(a) (b)

Figure 2.5: (a): Staggered cells in two dimensions. (b): A staggered cell in three dimensions.

conservation equation (2.33a) are evaluated at the nodes where the corresponding components of $\boldsymbol{u}$ are placed. This allows for simple expressions for the diffusion terms

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{i+\frac{1}{2},j,k} \approx \frac{u_{i+\frac{3}{2},j,k} + 2\,u_{i+\frac{1}{2},j,k} + u_{i-\frac{1}{2},j,k}}{h_x^2},$$

$$\left.\frac{\partial^2 u}{\partial y^2}\right|_{i+\frac{1}{2},j,k} \approx \frac{u_{i+\frac{1}{2},j+1,k} + 2\,u_{i+\frac{1}{2},j,k} + u_{i+\frac{1}{2},j-1,k}}{h_y^2},$$

$$\left.\frac{\partial^2 u}{\partial z^2}\right|_{i+\frac{1}{2},j,k} \approx \frac{u_{i+\frac{1}{2},j,k+1} + 2\,u_{i+\frac{1}{2},j,k} + u_{i+\frac{1}{2},j,k-1}}{h_z^2},$$

and the pressure forces

$$\left.\frac{\partial p}{\partial x}\right|_{i+\frac{1}{2},j,k} \approx \frac{p_{i+1,j,k} - p_{i,j,k}}{h_x},$$

with analogous expressions for the other components v and w. The evaluation of the continuity equation (2.33b) at the pressure nodes is similarly easy:

$$\nabla \cdot \boldsymbol{u}|_{i,j,k} \approx \frac{u_{i+\frac{1}{2},j,k} - u_{i-\frac{1}{2},j,k}}{h_x} + \frac{v_{i,j+\frac{1}{2},k} - v_{i,j-\frac{1}{2},k}}{h_y} + \frac{w_{i,j,k+\frac{1}{2}} - w_{i,j,k-\frac{1}{2}}}{h_z}.$$

The evaluation of the advection term is slightly more complicated, as it requires the values of certain variables at positions in which they are absent.

The remedy is using averages between adjacent points, which are second order accurate in space. For example, the estimation of the value of v in one of the u nodes would require an average between four points:

$$v_{i+\frac{1}{2},j,k} \approx \frac{v_{i+1,j+\frac{1}{2},k} + v_{i,j+\frac{1}{2},k} + v_{i+1,j-\frac{1}{2},k} + v_{i,j-\frac{1}{2},k}}{4}.$$

In general, when dealing with non-linear terms, Roache [27, p. 214] suggests making the products of averages, rather than taking the averages of products.
With staggered grids, some velocity components will not be precisely defined at the boundaries. In order to enforce boundary conditions on these components, Harlow and Welch [12] suggested the use of ghost cells outside the boundary. In these cells the velocity is defined so that its average with its adjacent value inside the domain is the desired boundary value. For example, at a solid wall normal to the x direction, the value of $u_{N+\frac{1}{2},j,k}$ in the gost cell outside the boundary is set to be $-u_{N-\frac{1}{2},j,k}$, so that their average $u_{N,j,k}$ is null. See figure 2.6.

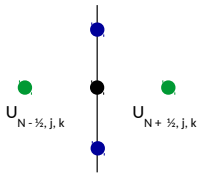


Figure 2.6: Ghost cell at the boundary.

2.3.2 The pressure term

The system (2.33) contains four equations for the four unknowns u, v, w and p. It would be tempting to solve all of them together, arranging their discrete (linearized) equivalents into a single linear system of equations. This approach, however, is impractical. The resulting matrix, in fact, would be difficult to solve numerically [13, p. 30]. The origin of the problem is more readily seen reformulating (2.33) in functional form, as is done, for example, in [23, p. 451]. The solution of (2.33) among all possible values of $\boldsymbol{u}$ and p is a saddle point of the functional, which is more difficult to find than an actual extremum point: it is a minimum in $\boldsymbol{u}$ and a maximum in p. In order to avoid the saddle-point problem, it would be desiderable to separate the search for $\boldsymbol{u}$ and the search for p in two different steps.

The Poisson equation for pressure

As pointed out in [20, p. 114], the pressure field p is implicitly specified by the continuity equation (2.33b). A proper pressure field substituted into

(2.33a) would result in a divergence-free velocity field. The question is how to determine this proper pressure field. Commonly, this is done by taking the divergence of the momentum conservation equation, which results in a Poisson equation for the pressure. In perfect arithmetic, taking the divergence of (2.33a) most terms cancel out due to (2.33b), and the resulting equation is:

$$\nabla^2 p = -\nabla \boldsymbol{u} : \nabla \boldsymbol{u} \,. \tag{2.36}$$

Pressure in the Marker and Cell method

In the original MAC method [12], the pressure was computed from the velocity field at the previous time step, but without assuming this field to be divergence-free. In fact, it may not have been entirely, due to the finite precision of the computer arithmetic, or due to an incomplete solution of the Poisson equation at the previous step, which was generally solved iteratively [27, p. 211]. Defining $D = \nabla \cdot \boldsymbol{u}$ the equation for pressure in the MAC method was

$$\nabla^2 p = -\frac{\partial D}{\partial t} - \nabla \cdot \left[\nabla \cdot \left(\boldsymbol{u}\, \boldsymbol{u}^T\right)\right] + \frac{1}{\mathrm{Re}} \nabla^2 D \,, \tag{2.37}$$

with $\frac{\partial D}{\partial t}$ approximated as $\frac{D^{n+1} - D^n}{\Delta t}$. The satisfaction of the continuity equation (2.33b) at the next step was imposed by setting $D^{n+1} = 0$ in (2.37). The same treatment of the pressure was used in [6]. In the last decades this method has fallen out of favour and projection methods have taken its place.

Projection methods

Projection methods can be seen as an application of operator splitting, as seen in 2.2.5. The earliest projection method was proposed by Chorin and Temam. In the first stage the momentum conservation equation is solved without the pressure term:

$$\frac{\widetilde{\boldsymbol{u}}^{n+1} - \boldsymbol{u}^n}{\Delta t} = -\nabla \cdot \left(\boldsymbol{u}^* \, \boldsymbol{u}^{**T}\right) + \frac{1}{\mathrm{Re}} \nabla^2 \widetilde{\boldsymbol{u}}^{n+1} + \boldsymbol{f}^{n+1} \,, \qquad \widetilde{\boldsymbol{u}}^{n+1}\big|_{\partial\Omega} = 0 \,. \tag{2.38}$$

Homogeneous boundary conditions have been assumed for simplicity. The advection term can be treated either explicitly or implicitly [23, p. 475]. In the second stage the following system is solved:

$$\begin{cases} \dfrac{\boldsymbol{u}^{n+1} - \widetilde{\boldsymbol{u}}^{n+1}}{\Delta t} = -\nabla p^{n+1} \,, \qquad \boldsymbol{u}^{n+1}\, \boldsymbol{n}\big|_{\partial\Omega} = 0 \,, & \text{(2.39a)} \\ \nabla \cdot \boldsymbol{u}^{n+1} = 0 \,. & \text{(2.39b)} \end{cases}$$

This is done by taking the divergence of (2.39a) and solving the resulting Poisson equation for the pressure

$$\nabla^2 p^{n+1} = \frac{1}{\Delta t}\, \widetilde{\boldsymbol{u}}^{n+1} \,, \qquad \left.\frac{p^{n+1}}{\partial n}\right|_{\partial\Omega} = 0 \,. \tag{2.40}$$

Once the pressure field is known, the velocity field $\boldsymbol{u}^{n+1}$ is found with

$$\boldsymbol{u}^{n+1} = \widetilde{\boldsymbol{u}}^{n+1} - \Delta t \, \nabla p^{n+1} \,. \tag{2.41}$$

$\widetilde{\boldsymbol{u}}^{n+1}$ is, in general, not divergence-free but, according to Ladyzhenskaya's theorem [22, p. 178] it admits a unique decomposition into a solenoidal part and an irrotational-part. (2.41) returns its divergence-free part $\boldsymbol{u}^{n+1}$, and is therefore referred to as the projection step, since it projects $\widetilde{\boldsymbol{u}}^{n+1}$ on a solenoidal functional space.
By deriving $\widetilde{\boldsymbol{u}}^{n+1}$ from (2.39a) and substituting it into (2.38) a forward or bacward Euler approximation of (2.33a) is found, with some additional $\mathrm{O}(\Delta t)$ terms if the advection or diffusion terms are treated implicitly. The order of consistency of the method is $\mathrm{O}(\Delta t)$, however, the order of convergence considering both pressure and velocity is only $\mathrm{O}\left(\Delta t^{1/2}\right)$ [10, Section 3.1]. This is due to the artificial Neumann boundary condition in (2.40). All error estimates in [10] are actually for the application of these methods to the Stokes equations, however, they should apply to the full Navier-Stokes equations since the advection term does not influence the error due to operator splitting [10, Section 2].
A variation on the original Chorin-Temam scheme, with the aim of improving accuracy, is the incremental scheme, obtained adding an explicit approximation of $p^{n+\frac{1}{2}}$ to (2.38), like, for example, its value at the previous step:

$$\begin{cases} \dfrac{\widetilde{\boldsymbol{u}}^{n+1} - \boldsymbol{u}^{n}}{\Delta t} = -\nabla \cdot \left(\boldsymbol{u}^{*}\, \boldsymbol{u}^{**T}\right) - \nabla p^{n-\frac{1}{2}} + \dfrac{1}{\mathrm{Re}} \nabla^2 \widetilde{\boldsymbol{u}}^{n+1} + \boldsymbol{f}^{n+1} \,, \\ \widetilde{\boldsymbol{u}}^{n+1}\Big|_{\partial\Omega} = 0 \,. \end{cases} \tag{2.42}$$

Equation (2.40) is now solved for the pressure increment $\phi^{n+\frac{1}{2}} = p^{n+\frac{1}{2}} - p^{n-\frac{1}{2}}$:

$$\nabla^2 \phi^{n+\frac{1}{2}} = \frac{1}{\Delta t}\, \widetilde{\boldsymbol{u}}^{n+1} \,, \qquad \left.\frac{\phi^{n+\frac{1}{2}}}{\partial n}\right|_{\partial\Omega} = 0 \,, \tag{2.43}$$

with the projection step

$$\boldsymbol{u}^{n+1} = \widetilde{\boldsymbol{u}}^{n+1} - \Delta t \, \nabla \phi^{n+\frac{1}{2}} \,. \tag{2.44}$$

In this case it is easy to see that

$$\left.\frac{\partial p^{n+\frac{1}{2}}}{\partial n}\right|_{\partial\Omega} = \left.\frac{p^{-\frac{1}{2}}}{\partial n}\right|_{\partial\Omega} \,;$$

$p^{-\frac{1}{2}}$ is the value used to initialize the pressure. If it is chosen carefully, the problem of the artificial Neumann boundary condition affecting the original Chorin-Temam scheme should be avoided [10, Section 3.2].

Another variation of the projection algorithm is the rotational scheme. (2.42) is re-written as

$$\begin{cases} \dfrac{\widetilde{\boldsymbol{u}}^{n+1} - \boldsymbol{u}^n}{\Delta t} = -\nabla \cdot \left(\boldsymbol{u}\,\boldsymbol{u}^T\right)\Big|^{n+\frac{1}{2}} - \nabla \widehat{p}^{\,n+\frac{1}{2}} + \dfrac{1}{\mathrm{Re}}\nabla^2 \widetilde{\boldsymbol{u}}^{n+\frac{1}{2}} + \boldsymbol{f}^{n+\frac{1}{2}}, \\ \widetilde{\boldsymbol{u}}^{n+1}\Big|_{\partial\Omega} = 0\,, \end{cases} \tag{2.45}$$

where $\widetilde{\boldsymbol{u}}^{n+\frac{1}{2}}$ and $\boldsymbol{u}^{n+\frac{1}{2}}$ are approximated as $\frac{1}{2}\left(\widetilde{\boldsymbol{u}}^{n+1} + \widetilde{\boldsymbol{u}}^{n}\right)$ and $\frac{1}{2}\left(\boldsymbol{u}^{n+1} + \boldsymbol{u}^{n}\right)$ respectively, and $\nabla \cdot \left(\boldsymbol{u}\,\boldsymbol{u}^T\right)\Big|^{n+\frac{1}{2}}$ and $\widehat{p}^{\,n+\frac{1}{2}}$ are second order accurate approximations of the advection term and of the pressure. $\widehat{p}^{\,n+\frac{1}{2}}$ needs to be an explicit approximation. The pressure increment is re-defined as

$$\phi^{n+\frac{1}{2}} = p^{n+\frac{1}{2}} - \widehat{p}^{\,n+\frac{1}{2}} + \frac{\chi}{\mathrm{Re}}\nabla \cdot \widetilde{\boldsymbol{u}}^{n+\frac{1}{2}}\,. \tag{2.46}$$

For simplicity, the case for $\chi = 1$ is considered. The second stage is derived, again, from the system

$$\begin{cases} \dfrac{\boldsymbol{u}^{n+1} - \widetilde{\boldsymbol{u}}^{n+1}}{\Delta t} = -\nabla \phi^{n+\frac{1}{2}}\,, \qquad \boldsymbol{u}^{n+1}\,\boldsymbol{n}\big|_{\partial\Omega} = 0\,, & \text{(2.47a)} \\ \nabla \cdot \boldsymbol{u}^{n+1} = 0\,. & \text{(2.47b)} \end{cases}$$

Making use of the vector identity

$$\nabla^2 \boldsymbol{u} = \nabla\left(\nabla \cdot \boldsymbol{u}\right) - \nabla \times \left(\nabla \times \boldsymbol{u}\right)\,, \tag{2.48}$$

and of the fact that, due to the curl of a gradient being null, (2.47a) implies

$$\nabla \times \left(\nabla \times \widetilde{\boldsymbol{u}}^{n+1}\right) = \nabla \times \left(\nabla \times \boldsymbol{u}^{n+1}\right)\,, \tag{2.49}$$

it is possible to sum (2.45) to (2.47a), substitute (2.46) and, after some algebra, obtain

$$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\Delta t} = -\nabla \cdot \left(\boldsymbol{u}\boldsymbol{u}^T\right)\Big|^{n+\frac{1}{2}} - \nabla p^{\,n+\frac{1}{2}} - \frac{1}{\mathrm{Re}}\nabla \times \left(\nabla \times \boldsymbol{u}^{n+\frac{1}{2}}\right) + \boldsymbol{f}^{n+\frac{1}{2}}\,. \tag{2.50}$$

Due to (2.48) and (2.47b), $\nabla \times \left(\nabla \times \boldsymbol{u}^{n+\frac{1}{2}}\right) = -\nabla^2 \boldsymbol{u}^{n+\frac{1}{2}}$, so the rotational scheme is second order consistent in time. The advantage of the rotational scheme over the previous ones is that the artificial Neumann boundary condition for the pressure, which can be obtained multiplying (2.50) by $\boldsymbol{n}$ at the boundary, is consistent [10, Section 3.3]. The rotational scheme converges with order $\mathrm{O}\left(\Delta t^{3/2}\right)$ [13, p. 35]. A more general formulation can be found in [10, Section 3.4].

In the classical formulation of projection methods, $\widetilde{\boldsymbol{u}}^{n+1}$ is considered an intermediate value with no particular significance, however, it can be shown [10, Section 3.5] that it converges to the real solution with the same order of

accuracy as $\boldsymbol{u}^{n+1}$. While not entirely divergence-free, $\widetilde{\boldsymbol{u}}^{n+1}$ has the advantage, over $\boldsymbol{u}^{n+1}$, of adhering entirely to the prescribed boundary conditions. Due to how (2.47a) is formulated, in fact, only the normal component of $\boldsymbol{u}^{n+1}$ adheres to the prescribed boundary conditions. It would be possible to chose $\widetilde{\boldsymbol{u}}^{n+1}$ instead of $\boldsymbol{u}^{n+1}$ as the end result of the scheme, and, with some algebra, bypass $\boldsymbol{u}^n$ entirely. Considering the original Chorin-Temam scheme, it is possible to sum (2.38) and (2.39a) to obtain

$$\frac{\widetilde{\boldsymbol{u}}^{n+1} - \widetilde{\boldsymbol{u}}^{n}}{\Delta t} = \nabla \cdot \left(\boldsymbol{u}^{*}\, \boldsymbol{u}^{**T}\right) - \nabla p^{n} + \frac{1}{\mathrm{Re}}\, \nabla^2 \widetilde{\boldsymbol{u}}^{n+1} + \boldsymbol{f}^{n+1}\,. \tag{2.51}$$

Since the procedure to compute the advection term was not defined earlier, it is legitimate to compute it from the field $\widetilde{\boldsymbol{u}}$, either implicitly or explicitly, and substitute $-\nabla \cdot \left(\boldsymbol{u}^{*}\, \boldsymbol{u}^{**T}\right)$ for $-\nabla \cdot \left(\widetilde{\boldsymbol{u}}^{*}\, \widetilde{\boldsymbol{u}}^{**T}\right)$. This first stage computes the velocity using the pressure from the previous time step. The second stage is the pressure update (2.40). The two passages are:

$$\begin{cases} \dfrac{\widetilde{\boldsymbol{u}}^{n+1} - \widetilde{\boldsymbol{u}}^{n}}{\Delta t} - \nabla \cdot \left(\widetilde{\boldsymbol{u}}^{*} \widetilde{\boldsymbol{u}}^{**T}\right) - \nabla p^{n} + \dfrac{1}{\mathrm{Re}}\, \nabla^2 \widetilde{\boldsymbol{u}}^{n+1} + \boldsymbol{f}^{n+1}\,,\ \widetilde{\boldsymbol{u}}^{n+1}\Big|_{\partial\Omega} = 0\,, & \text{(2.52a)} \\ \Delta t\, \nabla^2 p^{n+1} = \widetilde{\boldsymbol{u}}^{n+1}\,, \qquad \dfrac{\partial p^{n+1}}{\partial n}\Big|_{\partial\Omega} = 0\,. & \text{(2.52b)} \end{cases}$$

This scheme can be re-interpreted as a first order accurate approximation of the system

$$\begin{cases} \dfrac{\partial \boldsymbol{u}_\epsilon}{\partial t} + \nabla \cdot \left(\boldsymbol{u}_\epsilon\, \boldsymbol{u}_\epsilon^{T}\right) + \nabla p_\epsilon - \dfrac{1}{\mathrm{Re}}\, \nabla^2 \boldsymbol{u}_\epsilon = \boldsymbol{f}\,, \qquad \boldsymbol{u}_\epsilon|_{\partial\Omega} = 0\,, & \text{(2.53a)} \\ \nabla \cdot \boldsymbol{u}_\epsilon - \epsilon\, \nabla^2 p_\epsilon = 0\,, \qquad \dfrac{\partial p_\epsilon}{\partial n}\Big|_{\partial\Omega} = 0\,, & \text{(2.53b)} \\ \boldsymbol{u}_\epsilon|_{t=0} = \boldsymbol{u}_0\,, \qquad p_\epsilon|_{t=0} = p_0\,, & \text{(2.53c)} \end{cases}$$

which can be seen as a singular perturbation of the Navier-Stokes equations (2.33), with $\epsilon = \Delta t$ as perturbation parameter. Methods such as (2.53) are also called pressure stabilization or pseudo-compressibility methods. The additional perturbation term decreases as $\mathrm{O}\left(\Delta t\right)$ in this case. The same line of reasoning can be applied to the rotational scheme. Summing (2.45) and (2.47), with $\widehat{p}^{\,n+\frac{1}{2}} = p^{n-\frac{1}{2}}$, equation

$$\frac{\widetilde{\boldsymbol{u}}^{n+1} - \widetilde{\boldsymbol{u}}^{n}}{\Delta t} = -\nabla \cdot \left(\widetilde{\boldsymbol{u}}\, \widetilde{\boldsymbol{u}}^{T}\right)\Big|^{n+\frac{1}{2}} - \nabla \left(p^{n-\frac{1}{2}} + \phi^{n-\frac{1}{2}}\right) + \frac{1}{\mathrm{Re}} \nabla^2 \widetilde{\boldsymbol{u}}^{n+\frac{1}{2}} + \boldsymbol{f}^{n+\frac{1}{2}}\,, \tag{2.54}$$

is obtained, where, again, the advection term was reformulated in terms of $\widetilde{\boldsymbol{u}}$. It can be proven [13, p. 37] that $p^{n-\frac{1}{2}} + \phi^{n-\frac{1}{2}} = p^{n+\frac{1}{2}} + \mathrm{O}\left(\Delta t^2\right)$, so it is possible to define the pressure predictor

$$\widetilde{p}^{\,n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \phi^{n-\frac{1}{2}}\,, \tag{2.55}$$

then, the entire scheme is obtained combining (2.54), (2.43) and (2.46):

$$\begin{cases} \dfrac{\widetilde{\boldsymbol{u}}^{n+1}-\widetilde{\boldsymbol{u}}^{n}}{\Delta t} = -\nabla\cdot\left(\widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}}^{T}\right)\Big|^{n+\frac{1}{2}} - \nabla\widetilde{p}^{n+\frac{1}{2}} + \dfrac{1}{\mathrm{Re}}\nabla^2\widetilde{\boldsymbol{u}}^{n+\frac{1}{2}} + \boldsymbol{f}^{n+\frac{1}{2}}\,, \\ \widetilde{\boldsymbol{u}}^{n+1}\Big|_{\partial\Omega} = 0\,, & (2.56a) \\ \nabla^2\phi^{n+\frac{1}{2}} = \dfrac{1}{\Delta t}\,\widetilde{\boldsymbol{u}}^{n+1}\,, \qquad \left.\dfrac{\phi^{n+\frac{1}{2}}}{\partial n}\right|_{\partial\Omega} = 0\,, & (2.56b) \\ p^{n+\frac{1}{2}} = \phi^{n+\frac{1}{2}} + p^{n-\frac{1}{2}} - \dfrac{\chi}{\mathrm{Re}}\,\nabla\cdot\widetilde{\boldsymbol{u}}^{n+\frac{1}{2}}\,, & (2.56c) \end{cases}$$

This scheme is a second order accurate approximation of the perturbed system

$$\begin{cases} \dfrac{\partial\boldsymbol{u}_\epsilon}{\partial t} + \nabla\cdot\left(\boldsymbol{u}_\epsilon\,\boldsymbol{u}_\epsilon^{T}\right) + \nabla p_\epsilon - \dfrac{1}{\mathrm{Re}}\,\nabla^2\boldsymbol{u}_\epsilon = \boldsymbol{f}\,, \qquad \boldsymbol{u}_\epsilon|_{\partial\Omega} = 0\,, & (2.57a) \\ \nabla\cdot\boldsymbol{u}_\epsilon - \epsilon\,\nabla^2\phi_\epsilon = 0\,, \qquad \left.\dfrac{\phi_\epsilon}{\partial n}\right|_{\partial\Omega} = 0\,, & (2.57b) \\ \boldsymbol{u}_\epsilon|_{t=0} = \boldsymbol{u}_0\,, \qquad p_\epsilon|_{t=0} = p_0\,, & (2.57c) \\ \phi_\epsilon = \epsilon\,\dfrac{\partial p_\epsilon}{\partial t} + \dfrac{\chi}{\mathrm{Re}}\,\nabla\cdot\boldsymbol{u}_\epsilon\,, & (2.57d) \end{cases}$$

where, again, $\epsilon = \Delta t$. It can be shown that the pressure increment $\phi^{n+\frac{1}{2}}$ is $\mathrm{O}\left(\Delta t\right)$, so in this case the perturbation term is $\mathrm{O}\left(\Delta t^2\right)$.
The solution to system (2.53) converges to the solution to the original Navier-Stokes equation as

$$\|\boldsymbol{u}_\epsilon - \boldsymbol{u}\|_{\mathrm{L}^\infty(0,\,T;\,\mathrm{H}^1(\Omega))} + \|p_\epsilon - p\|_{\mathrm{L}^\infty(0,\,T;\,\mathrm{L}^2(\Omega))} \leq C\,\Delta t^{\frac{1}{2}}\,, \tag{2.58}$$

while system (2.57) converges to it as

$$\|\boldsymbol{u}_\epsilon - \boldsymbol{u}\|_{\mathrm{L}^2(0,\,T;\,\mathrm{H}^1(\Omega))} + \|p_\epsilon - p\|_{\mathrm{L}^2(0,\,T;\,\mathrm{L}^2(\Omega))} \leq C\,\Delta t^{\frac{3}{2}}\,, \tag{2.59}$$

As long as $0 < \chi \leq 1$. Proofs of convergence can be found in [25] and in [31]. The norms in equations (2.58) and (2.59) are defined as:

$$\|a\|^2_{\mathrm{L}^\infty(0,\,T;\,\mathrm{M})} = \max_{n\in[0,\,N]}\|a^n\|^2_{\mathrm{M}} \tag{2.60a}$$

$$\|b\|^2_{\mathrm{L}^2(0,\,T;\,\mathrm{M})} = \Delta t\sum_{n=0}^{N}\|b^n\|^2_{\mathrm{M}} \tag{2.60b}$$

$$\|\boldsymbol{v}\|^2_{\mathrm{H}^1(\Omega)} = \int_\Omega \boldsymbol{v}\cdot\boldsymbol{v}\,\mathrm{d}\Omega + \int_\Omega \nabla\boldsymbol{v} : \nabla\boldsymbol{v}\,\mathrm{d}\Omega \tag{2.60c}$$

$$\|q\|^2_{\mathrm{L}^2(\Omega)} = \int_\Omega q^2\,\mathrm{d}\Omega \tag{2.60d}$$

$$\|q\|^2_{\mathrm{L}^\infty(\Omega)} = \max_{\boldsymbol{x}\in\Omega}|\,q\,|^2\,. \tag{2.60e}$$

A more detailed summary of functional spaces and norms can be found in [23, Chapter 2].

Alternating direction for the pressure equation

Projection schemes are very effective in solving the Navier-Stokes equations. However, especially when a highly efficient technique such as the ADI scheme of section 2.2.5 is used for solving the momentum conservation equation at the first stage, the subsequent Poisson equation for pressure represents a performance bottleneck. A workaround first proposed by Guermond and Minev [9] consists of substituting the Laplacian operator in the pressure equation of a singular perturbation method with a different operator that can be split into a sequence of one-dimensional problems. This allows for the pressure equation to be solved by successive applications of the Thomas algorithm, in the same way as the divergence term of the momentum conservation equation. This is advantageous not only due to the greater efficiency of Thomas algorithm compared to the most common solution techniques for solving a multi-dimensional Poisson equation, such as fast Fourier transform methods, whose applicability is limited to homogeneous flows, or multigrid methods, but also due to its better scalability on a parallel computer architecture.
The Laplacian operator $-\nabla^2$ is substituted by a generic operator A in the system (2.57):

$$\begin{cases} \dfrac{\partial \boldsymbol{u}_\epsilon}{\partial t} + \nabla \cdot \left(\boldsymbol{u}_\epsilon \, \boldsymbol{u}_\epsilon^T\right) + \nabla p_\epsilon - \dfrac{1}{\mathrm{Re}} \nabla^2 \boldsymbol{u}_\epsilon = \boldsymbol{f} \,, \quad \boldsymbol{u}_\epsilon|_{\partial\Omega} = 0 \,, & \text{(2.61a)} \\ \nabla \cdot \boldsymbol{u}_\epsilon + \Delta t \, \mathrm{A} \phi_\epsilon = 0 \,, \quad \phi_\epsilon \in \mathcal{D}\left(\mathrm{A}\right) \,, & \text{(2.61b)} \\ \boldsymbol{u}_\epsilon|_{t=0} = \boldsymbol{u}_0 \,, \qquad p_\epsilon|_{t=0} = p_0 \,, & \text{(2.61c)} \\ \phi_\epsilon = \Delta t \dfrac{\partial p_\epsilon}{\partial t} + \dfrac{\chi}{\mathrm{Re}} \nabla \cdot \boldsymbol{u}_\epsilon \,. & \text{(2.61d)} \end{cases}$$

Operator A is defined as

$$\mathrm{A} = \left(1 - \frac{\partial^2}{\partial x^2}\right) \left(1 - \frac{\partial^2}{\partial y^2}\right) \left(1 - \frac{\partial^2}{\partial z^2}\right) , \tag{2.62}$$

and its definition domain $\mathcal{D}\left(\mathrm{A}\right)$ is the set of functions q such that

$$\begin{cases} \left.\dfrac{\partial q}{\partial z}\right|_{z \in \partial\Omega} = 0 \,, & \text{(2.63a)} \\ \left(1 - \dfrac{\partial^2}{\partial z^2}\right) \left.\dfrac{\partial q}{\partial y}\right|_{y \in \partial\Omega} = 0 \,, & \text{(2.63b)} \\ \left(1 - \dfrac{\partial^2}{\partial y^2}\right) \left(1 - \dfrac{\partial^2}{\partial z^2}\right) \left.\dfrac{\partial q}{\partial x}\right|_{x \in \partial\Omega} = 0 \,. & \text{(2.63c)} \end{cases}$$

Operator A induces the bilinear form

$$a\left(p, \, q\right) = \int_\Omega q \, \mathrm{A}p \, \mathrm{d}\Omega \,. \tag{2.64}$$

It is observed in [9, Section 2.2] that, if operator A satisfies the properties

$$a\,(p,\,q) = a\,(q,\,p)\;, \qquad \forall\, p,\, q \in \mathcal{D}\,(\mathrm{A})\;, \tag{2.65a}$$

$$a\,(q,\,q) \geq \|\nabla q\|^2_{\mathrm{L}^2(Omega)}\;, \qquad \forall\, q \in \mathcal{D}\,(\mathrm{A})\;, \tag{2.65b}$$

then the error estimate (2.59) for the solution of (2.57) applies also to (2.61). In [9, Section 2.4] it is proven that (2.62) does in fact satisfy properties (2.65a) and (2.65b). If A is substituted into (2.53c) instead, its solution will still satisfy error estimate (2.58).
The full method presented in [9, Section 4] is composed of the following stages:

1. Pressure predictor:
   Pressure is initialized by setting $p^{-3/2} = p_0$ and $p^{-1/2} = p_0$. For all subsequent steps:
   $$\widetilde{p}^{n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \phi^{n-\frac{1}{2}}\,. \tag{2.66}$$

2. Velocity update:
   Equation (2.61a) is solved applying the ADI scheme (2.32) of section 2.2.5:
   $$\begin{aligned}\frac{\boldsymbol{u}^* - \boldsymbol{u}^n}{\Delta t} =& -\nabla\cdot(\boldsymbol{u}\,\boldsymbol{u})|^{n+\frac{1}{2}} - \nabla\widetilde{p}^{n+\frac{1}{2}} + \\ &+\frac{1}{\mathrm{Re}}\,\nabla^2\boldsymbol{u}^n + f^{n+\frac{1}{2}}\,, \qquad \boldsymbol{u}^n|_{\partial\Omega} = 0\,, \end{aligned}\tag{2.67a}$$

   $$\frac{\boldsymbol{u}^{**} - \boldsymbol{u}^*}{\Delta t} = \frac{1}{2\,\mathrm{Re}}\,\frac{\delta^2}{\delta x^2}\,(\boldsymbol{u}^{**} - \boldsymbol{u}^n)\;, \quad \boldsymbol{u}^{**}|_{x\in\partial\Omega} = 0\,, \tag{2.67b}$$

   $$\frac{\boldsymbol{u}^{***} - \boldsymbol{u}^{**}}{\Delta t} = \frac{1}{2\,\mathrm{Re}}\,\frac{\delta^2}{\delta y^2}\,(\boldsymbol{u}^{***} - \boldsymbol{u}^n)\;, \quad \boldsymbol{u}^{***}|_{y\in\partial\Omega} = 0\,, \tag{2.67c}$$

   $$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^{***}}{\Delta t} = \frac{1}{2\,\mathrm{Re}}\,\frac{\delta^2}{\delta z^2}\,\left(\boldsymbol{u}^{n+1} - \boldsymbol{u}^n\right)\;, \quad \boldsymbol{u}^{n+1}\big|_{z\in\partial\Omega} = 0\,. \tag{2.67d}$$

   The advection term treatment is left unspecified for the moment and will be considered in 2.3.3.

3. Pressure increment:
   Equation (2.61b) is solved one direction at the time:
   $$\psi - \frac{\partial^2\psi}{\partial x^2} = -\frac{1}{\Delta t}\,\nabla\cdot\boldsymbol{u}^{n+1}\,, \qquad \left.\frac{\partial\psi}{\partial x}\right|_{x\in\partial\Omega} = 0\,, \tag{2.68a}$$

   $$\varphi - \frac{\partial^2\varphi}{\partial y^2} = \psi\,, \qquad \left.\frac{\partial\varphi}{\partial y}\right|_{y\in\partial\Omega} = 0\,, \tag{2.68b}$$

   $$\phi^{n+\frac{1}{2}} - \frac{\partial^2\phi^{n+\frac{1}{2}}}{\partial z^2} = \varphi\,, \qquad \left.\frac{\partial\phi^{n+\frac{1}{2}}}{\partial z}\right|_{z\in\partial\Omega} = 0\,. \tag{2.68c}$$

   This returns the value of the pressure increment $\phi^{n+\frac{1}{2}}$ to be used for the final sage

4. Pressure update:
   The current value of the pressure is obtained from equation (2.56c):

$$p^{n+\frac{1}{2}} = \phi^{n+\frac{1}{2}} + p^{n-\frac{1}{2}} - \frac{\chi}{\mathrm{Re}} \nabla \cdot \widetilde{\boldsymbol{u}}^{n+\frac{1}{2}} . \tag{2.69}$$

2.3.3 The advection term

A fully implicit treatment of the advection term $\nabla \cdot \left(\boldsymbol{u}\, \boldsymbol{u}^T \right)$ is difficult, since it requires a method for the solution of a system of non-linear equations, such as Newton's method. While this method is in use for finding solutions to steady problems [9, Section 5.2], for time-marching problems the cost of adding an iterative procedure to each step may prove excessive.
It is possible to use a semi-implicit approximation of the value of $\nabla \cdot \left(\boldsymbol{u}\, \boldsymbol{u}^T \right)$ at $t^{n+\frac{1}{2}}$

$$\nabla \cdot \left(\boldsymbol{u}\, \boldsymbol{u}^T \right) \Big|^{n+\frac{1}{2}} \approx \nabla \cdot \left(\boldsymbol{u}^{n+1}\, \boldsymbol{u}^{n^T} \right) , \tag{2.70}$$

which is second order accurate in time, is linear in the unknown term $\boldsymbol{u}^{n+1}$ and, according to [13, p. 55], has a wider stability region than explicit approximations. In [13], however, the second order accurate Adams-Bashforth scheme (2.18a) is preferred:

$$\nabla \cdot \left(\boldsymbol{u}\, \boldsymbol{u}^T \right) \Big|^{n+\frac{1}{2}} \approx \frac{1}{2} \nabla \cdot \left(3\, \boldsymbol{u}^n\, \boldsymbol{u}^{n^T} - \boldsymbol{u}^{n-1}\, \boldsymbol{u}^{n-1^T} \right) . \tag{2.71}$$

This method is conditionally stable, it requires that the Courant–Friedrichs–Lewy condition $\Delta t \leq \frac{1}{2} \frac{h}{|\boldsymbol{u}|}$ be satisfied [13, p. 55]. In [26], instead, a three-stage Runge Kutta method is used, in which, at each stage, the advection term is treated explicitly, while the diffusive and pressure terms are treated implicitly with the ADI methodology seen in 2.2.5.

2.3.4 The sub-grid scale stresses

The eddy viscosity coefficient for the Smagorinsky-Lilly model (1.31) can be expressed in non-dimensional form in terms of the non-dimensional gradient of the (filtered) velocity field $\overline{\boldsymbol{u}}$:

$$\nu_r^{nd} = C_S^2 \left(\Delta^{nd} \right)^2 \sqrt{\frac{1}{2} \left(\nabla \overline{\boldsymbol{u}} + \nabla \overline{\boldsymbol{u}}^T \right) : \left(\nabla \overline{\boldsymbol{u}} + \nabla \overline{\boldsymbol{u}}^T \right)} . \tag{2.72}$$

Where Δ^{nd} is the non-dimensional filter width obtained assuming the grid spacings are expressed in terms of the reference length $\mathcal{L}_0$. Defining a total non-dimensional viscosity coefficient

$$\nu^{tot} = \frac{1}{\mathrm{Re}} + \nu_r^{nd} , \tag{2.73}$$

the momentum and mass conservation equations for $\overline{\boldsymbol{u}}$ could be written as:

$$\begin{cases} \dfrac{\partial \overline{\boldsymbol{u}}}{\partial t} + \nabla \cdot \left(\overline{\boldsymbol{u}}\, \overline{\boldsymbol{u}}^T \right) + \nabla \overline{p} - \nabla \cdot \left[\nu^{tot} \left(\nabla \overline{\boldsymbol{u}} + \nabla \overline{\boldsymbol{u}}^T \right) \right] = \overline{\boldsymbol{f}}\,, & \text{(2.74a)} \\ \nabla \cdot \overline{\boldsymbol{u}} = 0\,, & \text{(2.74b)} \end{cases}$$

Where $\overline{p}$ is the non-dimensional filtered pressure, modified to include the isotropic part of the sub-grid scale stresses as illustrated in 1.3.2. The forcing term $\overline{\boldsymbol{f}}$ has been filtered too. The viscosity coefficient ν^{tot} is non-uniform in space so it cannot be taken out of the divergence operator applied to the stress tensor. This is one reason why the term $\nabla \overline{\boldsymbol{u}}^T$ is retained inside the expression. The other reason is that, as explained in 2.3.2, the continuity equation (2.74b) is hardly ever imposed directly, and, depending on the method used, $\nabla \cdot \left(\nabla \overline{\boldsymbol{u}}^T \right)$, may not entirely cancel out.
Regarding the advancement in time, the eddy viscosity coefficient (2.72) is best handled in an explicit manner, due to its non-linearity. In [6] it is computed from the velocity field at the previous step; since a leapfrog scheme (2.16) is used for time advancement, second order accuracy in time is preserved. In more general situations, if second order accuracy is of interest and memory requirements allow it, it is possible to approximate ν_r^{nd} at $t^{n+\frac{1}{2}}$ with a multi-step method, similarly to (2.71). However, stability concerns may be more pressing than time accuracy ones. The three components of the viscous and sub-grid scale forces are, expressed in terms of the velocity components $\overline{u}$, $\overline{v}$, $\overline{w}$:

$$\begin{array}{lll} \dfrac{\partial}{\partial x}\left(2\,\nu^{tot}\dfrac{\partial \overline{u}}{\partial x}\right) & +\dfrac{\partial}{\partial y}\left[\nu^{tot}\left(\dfrac{\partial \overline{u}}{\partial y}+\dfrac{\partial \overline{v}}{\partial x}\right)\right] & +\dfrac{\partial}{\partial z}\left[\nu^{tot}\left(\dfrac{\partial \overline{u}}{\partial z}+\dfrac{\partial \overline{w}}{\partial x}\right)\right] \\ \dfrac{\partial}{\partial x}\left[\nu^{tot}\left(\dfrac{\partial \overline{v}}{\partial x}+\dfrac{\partial \overline{u}}{\partial y}\right)\right] & +\dfrac{\partial}{\partial y}\left(2\,\nu^{tot}\dfrac{\partial \overline{v}}{\partial y}\right) & +\dfrac{\partial}{\partial z}\left[\nu^{tot}\left(\dfrac{\partial \overline{v}}{\partial z}+\dfrac{\partial \overline{w}}{\partial y}\right)\right] \\ \dfrac{\partial}{\partial x}\left[\nu^{tot}\left(\dfrac{\partial \overline{w}}{\partial x}+\dfrac{\partial \overline{u}}{\partial z}\right)\right] & +\dfrac{\partial}{\partial y}\left[\nu^{tot}\left(\dfrac{\partial \overline{w}}{\partial y}+\dfrac{\partial \overline{v}}{\partial z}\right)\right] & +\dfrac{\partial}{\partial z}\left(2\,\nu^{tot}\dfrac{\partial \overline{w}}{\partial z}\right) \end{array}$$

In general, it is possible to apply the ADI method to a situation in which the diffusion coefficient is non-uniform [7]. However, the cross terms that couple the three components of the momentum equation (2.74a), such as $\frac{\partial}{\partial y}\left(\nu^{tot}\frac{\partial \overline{v}}{\partial x}\right)$ or $\frac{\partial}{\partial x}\left(\nu^{tot}\frac{\partial \overline{u}}{\partial z}\right)$, cannot be included in the tridiagonal formulation of (2.32). Aside from the fact that different components of velocity are involved, mixed derivatives require at least a four-point stencil. In [26], the cross terms

$$\frac{\partial}{\partial y}\left(\nu^{tot}\frac{\partial \overline{v}}{\partial x}\right)+\frac{\partial}{\partial z}\left(\nu^{tot}\frac{\partial \overline{w}}{\partial x}\right) \qquad \text{(2.75a)}$$

$$\frac{\partial}{\partial z}\left(\nu^{tot}\frac{\partial \overline{w}}{\partial y}\right)+\frac{\partial}{\partial x}\left(\nu^{tot}\frac{\partial \overline{u}}{\partial y}\right) \qquad \text{(2.75b)}$$

$$\frac{\partial}{\partial x}\left(\nu^{tot}\frac{\partial \overline{u}}{\partial z}\right)+\frac{\partial}{\partial y}\left(\nu^{tot}\frac{\partial \overline{v}}{\partial z}\right) \qquad \text{(2.75c)}$$

are treated explicitly, in the same way as the advection term, and only the terms

$$2\,\frac{\partial}{\partial x}\left(\nu^{tot}\frac{\partial \overline{u}}{\partial x}\right)+\frac{\partial}{\partial y}\left(\nu^{tot}\frac{\partial \overline{u}}{\partial y}\right)+\frac{\partial}{\partial z}\left(\nu^{tot}\frac{\partial \overline{u}}{\partial z}\right) \tag{2.75d}$$

$$\frac{\partial}{\partial x}\left(\nu^{tot}\frac{\partial \overline{v}}{\partial x}\right)+2\,\frac{\partial}{\partial y}\left(\nu^{tot}\frac{\partial \overline{v}}{\partial y}\right)+\frac{\partial}{\partial z}\left(\nu^{tot}\frac{\partial \overline{v}}{\partial z}\right) \tag{2.75e}$$

$$\frac{\partial}{\partial x}\left(\nu^{tot}\frac{\partial \overline{w}}{\partial x}\right)+\frac{\partial}{\partial y}\left(\nu^{tot}\frac{\partial \overline{w}}{\partial y}\right)+2\,\frac{\partial}{\partial z}\left(\nu^{tot}\frac{\partial \overline{w}}{\partial z}\right) \tag{2.75f}$$

are included in the implicit ADI scheme.
Alternatively, one can treat molecular diffusion and turbulent diffusion separately, using the ADI method for the former and a fully explicit treatment for the latter. This has some advantages in terms of simplicity, especially if the turbulence model is added to a pre-existing code, but it may add further stability constraints to the scheme.

Chapter 3

The program

The software that this work is meant to expand upon is based on the original code developed by Guermond and Minev to test their direction splitting technique for the pressure equation proposed in [9]. It is written in Fortran 90 and makes use of the MPI Message passing interface libraries for parallelization. As explained in Chapter 2, direction splitting techniques can be used to break down a multi-dimensional problem into a sequence of one-dimensional ones. These, if centred finite differences on a Cartesian grid are employed, can be solved very efficiently by means of Thomas' algorithm. The latter, in turn, can be easily split into a sequence of independent tasks using the Schur's complement technique.
The original software could solve both two-dimensional and three-dimensional problems, however, some later additions, especially the immersed boundary, are not compatible with the two-dimensional formulation. In this work, only the three-dimensional formulation is considered.

3.1 Parallel computing and MPI

An algorithm can be seen as the breakdown of a problem into a sequence of operations. A computer, or, more specifically, its central processing unit, can be seen a machine built to execute this sequence of operations. In the simplest case, one operation at a time is executed. Central processing units, or CPUs, run a certain number of cycles per second. Each cycle generally consists of an instruction retrieval or "fetching" step and an interpretation step, after which a read, write, logical or algebraic operation performed on a piece of data, as dictated by that instruction. General purpose CPUs are usually capable performing only very basic operations during a single cycle. Purpose-built processing units, such as GPUs, are optimized instead to perform specific, complex tasks repeatedly.
The number of cycles per second a CPU performs is referred to as its clock speed or frequency. Ever since the introduction of integrated circuits, manu-

facturers of microprocessors have been trying to increase the maximum clock speed their products could reach. However, the increase in clock speed seems to have plateaued around values in the order of the Giga-Hertz (10^9 cycles per second) at the turn of the century. This is due to physical constraints: the faster a CPU runs, the more power it consumes. This power is converted into heat by the Joule effect, and, for the higher performing processors, has to be actively extracted. After a certain limit the energy expenditure for heat extraction becomes impractical, and it seems that this limit has been reached.

The solution employed by CPU vendors to keep improving performance while maintaining the same clock speed is parallelization: essentially, building systems containing more than one CPU, that can run at the same time. The introduction of parallel architectures in consumer electronics is relatively recent, but has already a long history in high performance applications, with vector processors being a staple of supercomputers since the 1970s. Nowadays, supercomputers with massively parallel architectures, and numbers of cores ranging in the thousands, are routinely found in the TOP500 list.

It has to be noted, however, that breaking an algorithm into a series of independent, or semi-independent tasks that can be run in parallel is a non-trivial problem, and sometimes even impossible. These tasks will, most of the time, require a certain amount of communication between one another, which is time-consuming. The lesser amount of communication is necessary for a particular parallel algorithm, the more scalable the algorithm is said to be, and the closer the speed multiplication factor brought by running it in parallel instead of serially will be to the number of processes.

When it comes to parallelization, two different approaches are possible: shared memory, in which all processes have access to all of the data and message passing, in which each process has its own set of data, and communication between processes is obtained through a finite amount of communication events, called messages. It must be pointed out that the shared memory approach does not entirely remove the cost of inter-process communication: while in some parallel architectures CPUs have a shared memory cache, whose access is almost instantaneous, most of the shared data will be in main memory. Both inter-processor communication and communication between a processor and main memory represent performance bottlenecks.

The MPI libraries, as the name suggests, are an implementation of the message passing approach to parallelization. With MPI, a single program is written and run on each processor: the code is the same, but the behaviour of each process is different, depending on the portion of data it handles and on its position in the "hierarchy" of processes.

In MPI, the relations between processes are defined by communicators. The default communicator, *MPI_ COMM_ WORLD*, simply assigns a number, or rank, to each process, from 0 to $N-1$, with N the total number of processes. A process can communicate with another by referencing its rank. This com-

municator, however, is useful only for a small number of processes, as it is not representative of either the actual system architecture or of the topology of the problem considered.
In the software used for this work, a Cartesian communicator is employed: processes are arranged along N orthogonal axes, and assigned an N-dimensional rank corresponding to the set of coordinates defining their position along the Cartesian axes. This way the physical domain of the problem can be split into blocks, and each block assigned to the process corresponding to its position in the Cartesian decomposition. The boundaries between the processes will then coincide with the boundaries between the sub-domains.
In MPI, sub-communicators, handling communication between subsets of processes, can also be defined. In the case of Cartesian communication, sub-communicators can be defined to handle communication along a single Cartesian direction.
Message passing in MPI can be either synchronous or asynchronous. In asynchronous communication, when a process sends a message, execution continues regardless of whether the target process has received it or not. In synchronous communication, when a message is sent, execution stops until the message is received. All processes share the same code, so if a "send" command is followed by a "receive" command, execution freezes. All processes start waiting for their counterpart to receive their message, before ever checking if they were sent one! However, if the domain is not periodic, there will be a process at the end of it with no one to send a message to. This process can be made to bypass the "send" command, for example, by making all processes check their rank to determine their position in the domain, and send a message only if they are not at the end of it. The last process will step immediately to the "receive" command, unlocking the processor next to it which can then go to its "receive" line. The receiving and sending of messages cascades in this fashion to the other end of the domain. In the software in use, synchronous communication only is employed.

3.2 Schur's complement technique

As explained in Chapter 2, the solution to a one-dimensional diffusion problem amounts to the solution of a tridiagonal linear system of equations. In order to understand how the solution of a tridiagonal system can be split into a series of semi-independent tasks, it is necessary to introduce the concept of the Schur complement. Given a matrix $\boldsymbol{M}$, split into blocks:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}, \tag{3.1}$$

the Schur complement of $\boldsymbol{D}$ is defined as:

$$\boldsymbol{M}/\boldsymbol{D} = \boldsymbol{A} - \boldsymbol{B}\,\boldsymbol{D}^{-1}\,\boldsymbol{C}\,, \tag{3.2a}$$

while the Schur complement of $\boldsymbol{A}$ is defined as:

$$\boldsymbol{M}/\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{C}\,\boldsymbol{A}^{-1}\,\boldsymbol{B}\,. \tag{3.2b}$$

Using the Schur complement, a linear system in the form:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{Bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{Bmatrix}, \tag{3.3}$$

Can be split into the two independent problems:

$$\begin{cases} \boldsymbol{x} = (\boldsymbol{M}/\boldsymbol{D})^{-1}\,\left(\boldsymbol{a} - \boldsymbol{B}\,\boldsymbol{D}^{-1}\,\boldsymbol{b}\right)\,, & \text{(3.4a)} \\ \boldsymbol{y} = (\boldsymbol{M}/\boldsymbol{A})^{-1}\,\left(\boldsymbol{b} - \boldsymbol{C}\,\boldsymbol{A}^{-1}\,\boldsymbol{a}\right)\,. & \text{(3.4b)} \end{cases}$$

This, however, is not particularly useful for geneic matrices: while it is true that the problem is split into smaller, independent tasks, the number of necessary matrix inversions increases. Figure 3.1 illustrates the structure

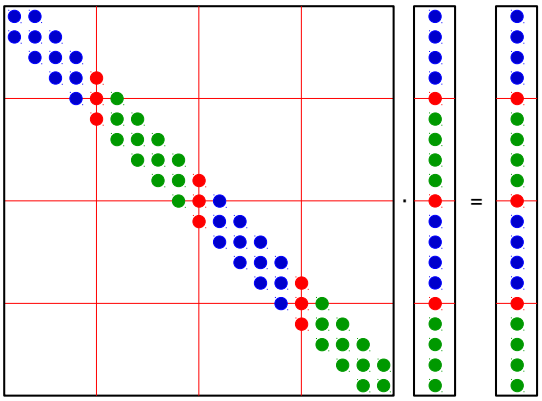


Figure 3.1: Tridiagonal system split between different processes.

of a tridiagonal problem, whose domain is divided among a certain amount of processes: the red dots represent interface values, which are shared by different processes. They are the critical part of the problem since their equation references two values, which are known respectively by one process but not by the other. Figure 3.2 shows the same problem, but rearranged so that all interface values are concentrated at the bottom. The problem can be divided, as in (3.2), between its internal and its interface portions:

$$\begin{bmatrix} \boldsymbol{A}_{ii} & \boldsymbol{A}_{ie} \\ \boldsymbol{A}_{ei} & \boldsymbol{A}_{ee} \end{bmatrix} \begin{Bmatrix} \boldsymbol{x}_i \\ \boldsymbol{x}_e \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{f}_i \\ \boldsymbol{f}_e \end{Bmatrix}. \tag{3.5}$$

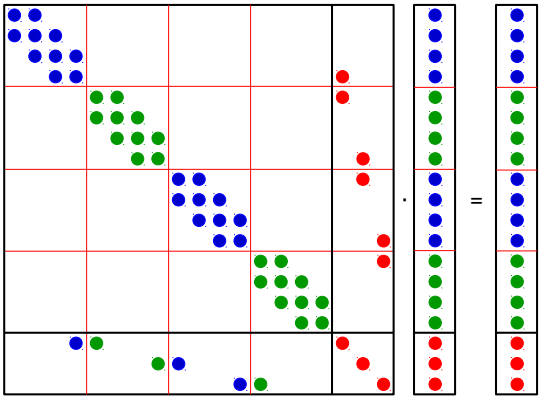


Figure 3.2: Tridiagonal system rearranged in order to separate its internal and interface parts.

It is easy to see from figure 3.2 that matrix $\boldsymbol{A}_{ii}$ is block-diagonal:

$$\boldsymbol{A}_{ii} = \begin{bmatrix} \boldsymbol{A}_{ii}^{(1)} & 0 & \cdots & 0 \\ 0 & \boldsymbol{A}_{ii}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{A}_{ii}^{(N_p)} \end{bmatrix}, \tag{3.6}$$

with each block corresponding to one process. The software proceeds as follows: the Schur complement

$$\boldsymbol{A}/\boldsymbol{A}_{ii} = \left(\boldsymbol{A}_{ee} - \boldsymbol{A}_{ei}\, \boldsymbol{A}_{ii}^{-1}\, \boldsymbol{A}_{ie}\right) \tag{3.7}$$

is computed only once at the beginning. It does not need to be re-computed as long as matrix $\boldsymbol{A}$ does not vary, which is the case in the original formulation. $\boldsymbol{A}/\boldsymbol{A}_{ii}$ is stored in the so-called "master process", the process having zero rank in the default communicator. At each time step, each process computes its independent portion of $\boldsymbol{A}_{ii}^{(p)\,-1}\, \boldsymbol{f}_i^{(p)}$, using Thomas' algorithm. Then, each process sends its portion of vector $\boldsymbol{A}_{ei}^{(p)}\, \boldsymbol{A}_{ii}^{(p)\,-1}\, \boldsymbol{f}_i^{(p)}$ to the master process. The latter assembles vector $\boldsymbol{A}_{ei}\, \boldsymbol{A}_{ii}^{-1}\, \boldsymbol{f}_i$ and computes $\boldsymbol{x}_e$ as:

$$\boldsymbol{x}_e = (\boldsymbol{A}/\boldsymbol{A}_{ii})^{-1}\, \left(\boldsymbol{f}_e - \boldsymbol{A}_{ei}\, \boldsymbol{A}_{ii}^{-1}\, \boldsymbol{f}_i\right), \tag{3.8}$$

a passage analogous to (3.4b). The solution of (3.8) is simple, since the Schur complement is diagonal. At this point, the master process broadcasts vector $\boldsymbol{x}_e$ to each process, which then computes:

$$\boldsymbol{x}_i^{(p)} = \boldsymbol{A}_{ii}^{(p)\,-1} \left(\boldsymbol{f}_i^{(p)} - \boldsymbol{A}_{ie}^{(p)}\, \boldsymbol{x}_e\right) \tag{3.9}$$

The entire amount of data exchanged between each process and the master process for one time step amounts to two vectors of the same size as $\boldsymbol{x}_e$.

The size of $\boldsymbol{x}_e$ is the number of domain subdivisions along the direction of the one-dimensional problem being solved, times the number of points in the other two directions. For a three-dimensional problem, the amount of data that needs to be exchanged grows as N^2.

3.3 Data storage

Fortran 90 allows for the definition of multi-dimensional arrays, which are an easy and intuitive way of representing physical fields over sequences of indexed points, like those of a structured Cartesian grid. The computer, however, stores each array as a one-dimensional sequence of values in memory, so the multi-dimensional structure of the data is lost. A three-dimensional array defined as $f_{i,j,k}$ in the code will actually be stored as f_n in memory, with $n = i + N_i\,(j-1) + N_i\,N_j\,(k-1)$. In this case, i is the fastest running direction, in the sense that adjacent values along direction i are the only ones actually adjacent in memory, while adjacent values along directions j and k will have, in memory, a distance of N_i and $N_i\,N_j$ respectively. Whether the fastest running direction is the one indexed first, such as i in the case above, or the last, may depend on the programming language or even on the compiler used.
It is important to know which is the fastest running direction when writing high performance code, since sequences of adjacent values in memory can be accessed faster than sequences of distant ones.
The software considered for this work, at least for its core functionalities, makes no use of multi-dimensional arrays: each quantity is stored as a one-dimensional array, indexed as explained above, directly in the code. The fastest running direction is chosen case by case to be the optimal one for a particular quantity. Three possible orderings are considered:

1. x fastest running, y second fastest running, z slowest running:
   $n = N_x\,N_y\,(i_z-1) + N_x\,(i_y-1) + i_x$

2. y fastest running, z second fastest running, x slowest running:
   $n = N_y\,N_z\,(i_x-1) + N_y\,(i_z-1) + i_y$

3. z fastest running, x second fastest running, y slowest running:
   $n = N_z\,N_x\,(i_y-1) + N_z\,(i_x-1) + i_z$

Quantities are stored as structures having two fields: a one-dimensional array of size $N_x\,N_y\,N_z$, and an integer called "direction" having a value of either 1, 2 or 3, depending on which one of the orderings above is used.
Vector fields, such as the velocity field, are generally ordered so that, for each component, the fastest running direction is that component's direction. Scalar fields are generally ordered with x as the fastest running direction. A subroutine called *reorder* can be used to change the ordering of an array.

The data structures for storing arrays are defined in the source file *./SOURCES/types.f90*. In the same file the structure *mesh_ types* is defined. It contains all information regarding the computational domain, both global (shared by all processes) and local. Other important global information such as the time step, the Reynolds number and the final time is stored in the *my_ data* structure, defined in the source file *./CODE/read.f90*. For the immersed boundary, the structures used are *SurfaceGrid*, defined in the source file *./SOURCES/surface.f90*, *IBGeoData* and *postProcessingData*, defined in the source file *./SOURCES/IBData.f90*.

3.4 The main cycle

The program's source code is organized into four directories: *./CODE/*, *./SOURCES/*, *./tools/* and *./UTILITIES/*. The main executable's source code, *pCNST.f90*, can be found in the *./CODE/* folder. It contains:

1. A list of all the modules included:

   ```
   PROGRAM NSt

    USE MPI
    USE types
    USE assembling

       ...
   ```

   "Module" is the term used in Fortran to refer to external libraries. With the exception of the MPI library, all external modules are compiled from source files contained in the four directories mentioned above. The program is essentially self-contained.

2. A list of variable declarations. These are usually preceded, in modern Fortran code, by the statement:

   ```
    IMPLICIT NONE
   ```

   This prevents the use of undeclared variables in the code. The compiler, in fact, may otherwise assign default types to undeclared variables, potentially different than those intended by the programmer, and this can cause conversion errors difficult to spot.

3. All information necessary to set up a simulation, except for that concerning the immersed boundary is contained in a file called *data*, and in three additional *.dat* files containing the coordinates of the grid along each Cartesian direction. The first subroutines called by the program read all this information and start building the mesh and the MPI communicators:

   ```
    CALL MPI_INIT(code)
   ```

```
CALL read_my_data(mesh)
CALL create_mesh_weights(mesh)
CALL create_cart_comm(mesh%Ndom,comm_cart,comm_one_d,coord_cart)

   ...
```

4. The following block builds the solution matrices and computes the Schur complement (3.7).

5. Following the computation of the Schur complement, the arrays containing all the relevant quantities are allocated.

6. Once allocated, all of these quantities are initialized. This block either reads their value from a set of restart files, or computes them from the initial and boundary conditions defined in the source file *./CODE/solutions.f90*. Velocity is initialized both at the initial time step, $t = 0$, and at the previous one $-\Delta t$. Pressure is initialized at $-\frac{1}{2}\Delta t$ and at $-\frac{3}{2}\Delta t$ as required by the initialization of (2.66). The advection term at $-\Delta t$ is also computed in this phase.

7. The following block contains a series of calls to the subroutines that handle the loading and initialization of the immersed boundary:

```
WRITE(*,*) "Initiatilization␣of␣the␣IB␣started"

CALL IBAllocateIB(mesh, IBGeo) ! Allocator for velocity

CALL IBAllocateSF(mesh, IBGeo)   ! Allocator for surface forces

CALL IBAllocatePP(mesh, IBGeo, IBPostProc)

  ...
```

8. After the immersed boundary is initialized, the main time stepping cycle begins: it is a DO WHILE construct, executed until the current time exceeds the imposed limit:

```
DO WHILE (t<tend-1D-8)
```

   (a) The very first line inside the DO loop increases the current time: $t = t + \Delta t$. The second, increases an integer counting how many time steps have elapsed:

```
t = t + dt
i_time_step = i_time_step + 1
```

   (b) The following block is executed by the master processor only, and only at certain time steps: it outputs the current time:

```
IF ( (MOD(i_time_step,w_time_steps) .EQ. 0) .OR. t<5*dt
    ) THEN
  IF ( rank .EQ. 0) THEN
    WRITE(*,*) '␣
        ------------------------------------------␣'
```

```
        WRITE(*,*) ' i/n     =' , i_time_step , '/' ,
            n_time_steps
        WRITE(*,*) ' t/tend  =' , t , '/' , tend
      END IF
    END IF
```

(c) Next, a DO loop is started over each component of the momentum equation:

```
DO j = 1, dim
```

The right-hand side of equation (2.67a), called rhs in the code, is initialized as null:

```
rhs%direction = vn(j)%direction
rhs%f1 = 0d0
```

then, one by one all the explicit terms of the equation are added to it. First the source term:

```
IF (param%src) THEN
!         CALL compute_sponge (rhs,mesh,t-0.5d0*dt,vn)
  CALL compute_source (rhs,mesh,t-0.5d0*dt)
END IF
```

then the gradient of the predicted pressure ((2.66)),

```
CALL grad(rhs, pPred, mesh,one_d_rank(j)+1)
```

then the advection term:

```
IF (param%advect) THEN
   rhs%f1=rhs%f1+0.5d0*advn(j)%f1
   advn(j)%direction=rhs%direction
   advn(j)%f1=0d0
   CALL advection(advn(j), vn, mesh, t-dt)
   rhs%f1=rhs%f1-1.5d0*advn(j)%f1
END IF
```

The latter one is projected at time step $t + \frac{1}{2}\Delta t$ as in (2.71): first rhs is added $\frac{1}{2}\nabla \left(\boldsymbol{u}^{n-1}\, \boldsymbol{u}^{n-1\,T}\right)_j$ from the previous step, then $\nabla \left(\boldsymbol{u}^{n}\, \boldsymbol{u}^{n\,T}\right)_j$ is computed, multiplied by $\frac{3}{2}$, subtracted from rhs and stored to be used on the next step.
Some of the terms computed, like the advection term and the predicted pressure gradient, involve finite differences that, at the internal boundaries, may reference values on different processes. However, no inter-process communication takes place in this phase. At the boundary, each process computes its side of the finite differences. When solving the linear system, all interface values of rhs must be sent to the master process in order for it to compute (3.8). At that point the incomplete finite differences are summed up.

(d) In the next block the momentum equation is solved by applying scheme (2.67). Actually, the scheme is slightly reformulated to

work with velocity increases:

$$\Delta u_j^0 = \text{rhs} - \Delta t\,\nu\,\nabla^2 u_j^n\,, \tag{3.10a}$$

$$\left(1 - \Delta t\,\nu\,\frac{1}{2},\frac{\partial^2}{\partial x^2}\right)\Delta u_j^1 = \Delta u_j^0\,, \tag{3.10b}$$

$$\left(1 - \Delta t\,\nu\,\frac{1}{2},\frac{\partial^2}{\partial y^2}\right)\Delta u_j^2 = \Delta u_j^1\,, \tag{3.10c}$$

$$\left(1 - \Delta t\,\nu\,\frac{1}{2},\frac{\partial^2}{\partial z^2}\right)\Delta u_j^3 = \Delta u_j^2\,. \tag{3.10d}$$

With Δu_j^3 being the actual velocity increase used to update u_j^n to u_j^{n+1}:

```
rhs%f1 = dt*rhs%f1
CALL predictor(rhs,vn(j),mesh,mat_vel,coord_cart,t-dt,
    param%dt,param%nu)
DO ii = 1, dim
   CALL reorder(rhs,vin(j),mesh)

   !=== Impose boundary conditions ===!
   CALL bcs(rhs,mesh,t,j,param%dt,param%nu)
   i = rhs%direction
   vin(j)%direction=rhs%direction
     nb_arrows = nNodes/(mesh%nint(i)+2)
   CALL solve_seq(comm_one_d(i),sch_vel(i,j),mat_vel(i
       ,j),rhs,vin(j),nb_arrows,mesh,i)
   rhs%f1 = vin(j)%f1
END DO
```

The *predictor* step corresponds to passage (3.10a), while the inner loop performs (3.10b), (3.10c) and (3.10d) with rhs and vin alternating their role as the Δu_j^k terms. Boundary conditions are also imposed at each passage.

(e) at the end of the DO loop over all components of the momentum equation, the velocity field is updated and its value corrected with the application of the boundary conditions:

```
! === Recover the solution ==!
DO j = 1, dim
   vnm1(j)%f1 = vn(j)%f1
   vn(j)%f1 = vn(j)%f1 + vin(j)%f1

   !=== Impose bcs on second boundary plane/line in in
       each direction except
   !=== solution direction=!
   CALL bcs_end(vn(j),mesh,t)
END DO
```

(f) the following block handles the pressure update as in (2.68):

```
! ========== Pressure loop==!
rhs%direction = 1
rhs%f1=0d0
CALL div(rhs,vn,mesh)
```

```
! === Compute divergence(u(n)+u(n+1))/2
vin(1)%f1 = 0.5d0*(divN%f1+rhs%f1)
divN%f1 = rhs%f1
rhs%f1 = -rhs%f1/dt
DO i = 1, dim
   nb_arrows = nNodes/(mesh%nint(i)+2)
   CALL solve_seq(comm_one_d(i),sch_press(i),mat_press(i)
       ,rhs,pPred,nb_arrows,mesh,i)
   CALL reorder(pPred,rhs,mesh)
   rhs%direction = pPred%direction
   rhs%f1 = pPred%f1
END DO
rhs%f1 = pn%f1
```

Plus the pressure increase (2.69):

$$p^{n+\frac{1}{2}} = \phi^{n+\frac{1}{2}} + p^{n-\frac{1}{2}} - \frac{\chi}{\mathrm{Re}} \nabla \cdot \widetilde{\boldsymbol{u}}^{n+\frac{1}{2}}$$

```
pn%f1 = pn%f1+pPred%f1-param%xi*vin(1)%f1
```

And the pressure prediction step: (2.66) $\widetilde{p}^{n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \phi^{n-\frac{1}{2}}$

```
! === Prediction = p(n-1/2) + phi(n-1/2)
pPred%f1 = pn%f1+pPred%f1
```

(g) At this point the subroutines that take care of the immersed boundary are called: the velocity correction necessary to satisfy the boundary conditions on the immersed geometry is applied explicitly at this point in the computation:

```
! ========== Immersed boundary procedure ========== !
  ! ========== Computation stencil, phi, cl and point on
      the border ========== !
  CALL IBComputeInterpolationData(mesh, IBGeo, comm_one_d,
      t, dt)
  ! ========== END Computation stencil, phi, cl and point
      on the border ========== !

  ! ========== Compute Surf velocity ========== !
  CALL SurfVelocity(IBGeo, IBPostProc, i_time_step, t, dt,
      V_surf, Vo_surf, Voo_surf, VA, 1/param%nu)
  ! ========== END Compute Surf velocity ========== !

  ! ========== IB velocity corrector ========== !
  CALL IBCorrectVelocity(vn, IBGeo, comm_one_d, mesh, t, dt
      , V_surf)
  ! ========== END IB velocity corrector ========== !

  ! ========== End immersed boundary procedure ========== !
```

(h) A further block handles the computations for a passive scalar, if its inclusion was requested in the file *data*.

(i) Before the end of the time stepping cycle, a set of subroutines is called to compute the stresses on the immersed surface. The resulting force on the body is output on the screen at each time step:

```
! ========== Force computation ========== !
! Computes the force on the external side
CALL IBInitForceCalc(mesh,IBGeo,comm_one_d)
CALL IBComputeStressTensor(param%nu, vn, pn, IBGeo,
    IBPostProc, mesh, comm_one_d)
CALL IBComputeForces(IBGeo, IBPostProc, forces)
IF (rank==0) THEN
   DO i = 1, SIZE(IBGeo)
     IF(IBGeo(i)%forceflag == 1 .OR. IBGeo(i)%forceflag
         == 2)  WRITE(*,*) 'Force␣on␣surface␣', i, '=',
         forces(:,1,i)
     IF(IBGeo(i)%forceflag == 3) WRITE(*,*) 'Force␣on␣
         surface␣', i, 'side␣1␣=', forces(:,1,i), 'side␣2
         ␣=', forces(:,2,i)
   END DO
END IF
```

9. After the time stepping cycle, there is the output block. The computed fields are exported as *.vtk* files, readable by post-processing software such as paraview, and also stored as raw ASCII files, which is the format the program uses for restarts.

10. The last few lines take care of deallocating the arrays used for the computation.

3.5 The immersed boundary

The treatment of the immersed boundary inside the program deserves a closer look. The implementation, and especially its parallelization aspects, are very complex, and they are still a work in progress, so only a description of the core methodology will be provided.
The immersed geometry is passed to the program as a triangulated mesh in the Gts (Gnu triangulated surface) format. A Gts file starts with a line specifying the numbers of points, segments and triangular elements contained in it. After this line, points are listed as triplets of coordinates in floating point format. Subsequently, the segments are listed as couples of integer values identifying their extremes from the previous list of points. Then, the triangular elements are listed as triplets of integers identifying their sides from the previous list of segments.
In the program, each triangular element is assigned a representative position, obtained as an average of its vertices, and referred to as Lagrangian point. This is to distinguish these points from the ones in the Cartesian grid used for the fluid dynamics computation, which are referred to as Eulerian points. In fact, the motion (or lack thereof) of the immersed boundary is described by the position of its elements, which is a Lagrangian approach, while the motion of the fluid is described by the evolution of fluid flow quantities at specific "stations", the grid points, which is an Eulerian approach.
The immersed boundary procedure is based on the one described in [5],

with some differences. In this type of method, the velocity correction is generally applied on the intermediate velocity of a projection scheme of the type described in 2.3.2, however, since in the model in use no intermediate velocity is present, the correction is applied explicitly at the end of each step to the current velocity field.
The immersed boundary method proceeds as follows:

1. The velocity at the Lagrangian points of the immersed geometry is computed from its value at (a subset of) the Eulerian points:

$$\widehat{u}_i^E \longrightarrow U_i^L$$

2. The corrective forcing term at each Lagrangian point is obtained as the difference between the local fluid velocity and the local immersed body velocity (null if the object is stationary):

$$F_i^L = \frac{V_i^L - U_i^L}{\Delta t}$$

3. the forcing term at (the same subset of) the Eulerian points is computed from its value at the Lagrangian points:

$$F_i^L \longrightarrow f_i^E$$

4. the corrective forcing term is applied to the velocity field:

$$u_i^E = \widehat{u}_i^E + \Delta t \, f_i^E$$

At the core of the method is the mapping between the values of a quantity at the Lagrangian points and its values at the Eulerian points: for each Lagrangian point, the closest Eulerian point is found. A stencil of seven points is considered for each Lagrangian point, consisting of the closest Eulerian point, plus its two surrounding Eulerian points along each Cartesian direction.
For each seven-point stencil, a set of four coefficients, $\boldsymbol{a}$, is defined, so that the velocity can be expressed as a linear function of position:

$$U_i(\boldsymbol{x}) = \boldsymbol{p}^T \boldsymbol{a}\,, \tag{3.11}$$

where $\boldsymbol{p}^T(\boldsymbol{x}) = \{1, x, y, z\}$. These coefficients are found minimizing the functional:

$$J = \sum_{n=1}^{7} W\left(\boldsymbol{x} - \boldsymbol{x}^{(n)}\right) \left[\boldsymbol{p}^T\left(\boldsymbol{x}^{(n)}\right) \boldsymbol{a} - u_i^{(n)}\right]^2 \tag{3.12}$$

Which is the sum of the squares of the errors committed applying (3.11) to estimate the values $u_i^{(n)}$ of the velocity at the seven stencil points $\boldsymbol{x}^{(n)}$. W

is a weigh function, such that points in the stencil closer to $\boldsymbol{x}$ contribute more to the error. Solving (3.12) for $\boldsymbol{a}$ as a function of $u_i^{(n)}$ brings the linear mapping between the velocity at the Lagrangian point $\boldsymbol{X}$ and the velocity at the seven stencil points associated to it:

$$U_i^L(\boldsymbol{X}) = \sum_{n=1}^{7} \phi^{(n)}(\boldsymbol{X})\, u_i^{(n)} . \tag{3.13}$$

Coefficients $\phi^{(n)}(\boldsymbol{X})$ are used also for the inverse relation, mapping the values of the forcing term each an Eulerian point that is part of at least one seven-point stencil and the values of the forcing term at the Lagrangian points corresponding to the N_l stencils it belongs to:

$$f_i^{(n)} = \sum_{l=1}^{N_l} c_L\, \phi^{(n)}(\boldsymbol{X}_l)\, F_i^{(l)} . \tag{3.14}$$

The scaling factors c_L are computed so that the Eulerian and the Lagrangian forcing terms in (3.14), multiplied by their associated volumes, coincide.
This approach to immersed boundaries is called moving least squares. "Moving" refers to the possibility of movement of the boundary, in which case the coefficents of (3.13) are recomputed at each step, and "least squares" to the minimization of the square error used to compute them.

Chapter 4

Implementation of the model

4.1 Preliminary tests

Both the original Smagorinsky model and the dynamic model require knowledge of the characteristic strain rate $\overline{S}$. Prior to the implementation of a LES closure model inside the program, a strategy to compute $\overline{S}$ was developed in Fortran 90 and tested on the velocity field:

$$\left\{ \begin{array}{c} u \\ v \\ w \end{array} \right\} = \left\{ \begin{array}{c} \sin(x)\,\sin(y+z) \\ -\cos(x)\,\cos(y+z) \\ 2\,\cos(x)\,\cos(y+z) \end{array} \right\} . \tag{4.1}$$

The spatial derivatives of (4.1) are easy to compute by hand. It can be readily verified that (4.1) satisfies the continuity equation (2.33b), and computing its gradient is also straightforward. The analytic expression for the characteristic strain rate of (4.1) is:

$$\overline{S} = \sqrt{5\,\cos(y+z)^2\,\sin(x)^2 + 13\,\sin(y+z)^2\cos(x)^2}\,, \tag{4.2}$$

which, along with analogous expressions for the components of the velocity gradient, provides an easy reference for testing the accuracy of the method. When computing the velocity gradient with centred finite differences on staggered grids, its diagonal components are naturally placed on the pressure nodes, while the off-diagonal components are placed at the corners of the cells. The situation is illustrated in figure 4.1. The values of the off-diagonal components of the velocity gradient at the pressure nodes can be obtained by taking the averages of their values at the four surrounding points:

$$\left.\frac{\partial u}{\partial y}\right|_{i,\,j} = \frac{1}{4}\left(\left.\frac{\partial u}{\partial y}\right|_{i+\frac{1}{2},\,j+\frac{1}{2}} + \left.\frac{\partial u}{\partial y}\right|_{i+\frac{1}{2},\,j-\frac{1}{2}} + \left.\frac{\partial u}{\partial y}\right|_{i-\frac{1}{2},\,j+\frac{1}{2}} + \left.\frac{\partial u}{\partial y}\right|_{i-\frac{1}{2},\,j-\frac{1}{2}} \right) ; \tag{4.3}$$

this procedure is second order accurate on a uniform grid, but it may not be on a non-uniform grid. As shown in figure 4.2, in fact, on a non-uniform

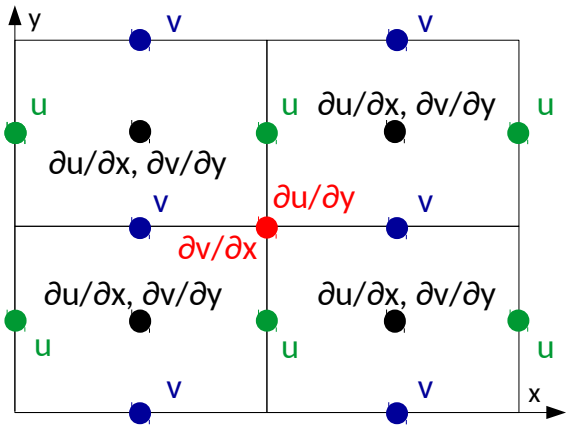


Figure 4.1: Velocity gradient components on uniform staggered grids.

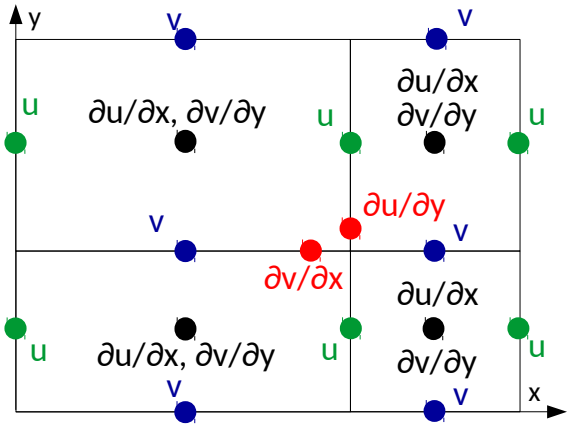
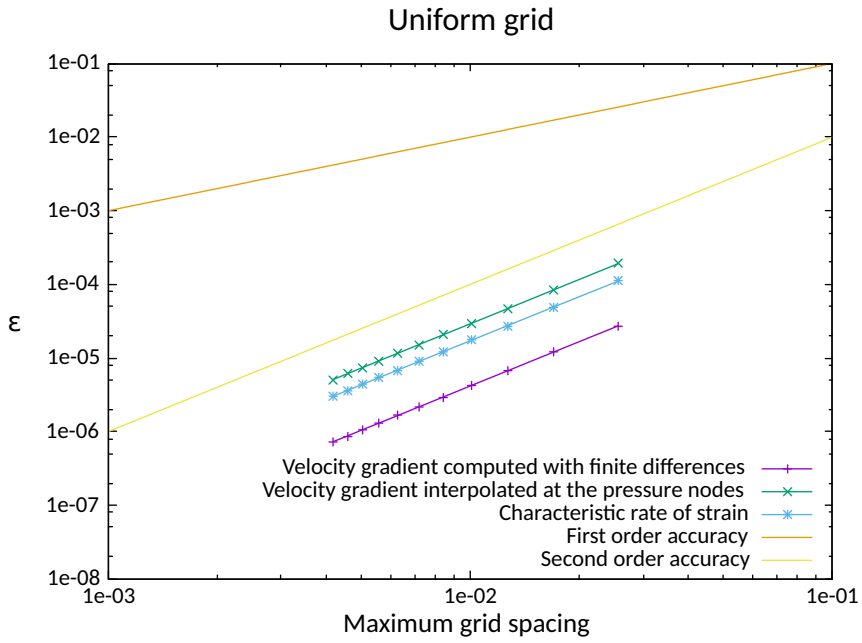
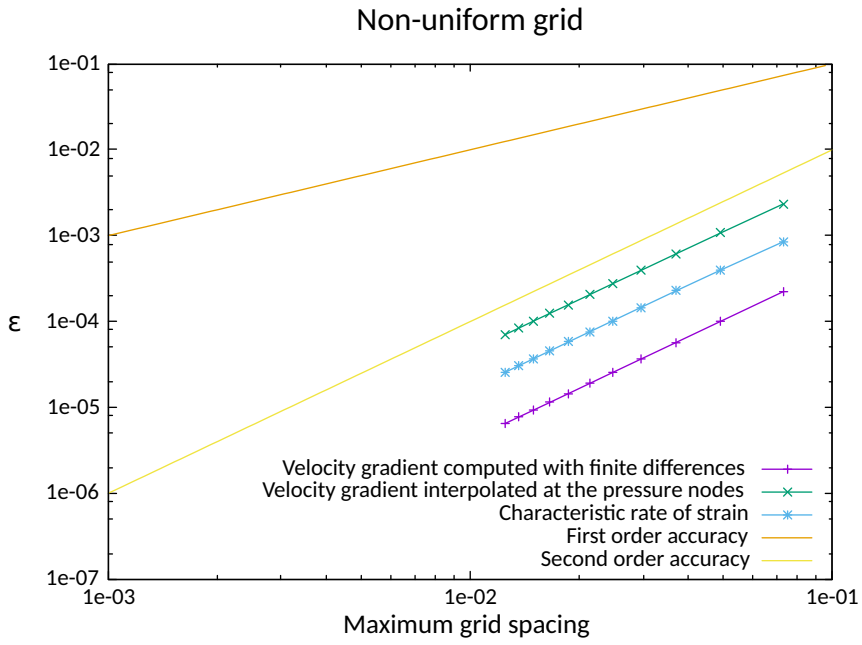


Figure 4.2: Velocity gradient components on non-uniform staggered grids.

grid the off-diagonal components of $\nabla \boldsymbol{u}$ are not placed exactly at the corners of the cells. However, for a sufficiently regular grid, the four-point average (4.3) is still a second order accurate approximation for the values of the off diagonal components of $\nabla \boldsymbol{u}$ at the pressure nodes.
This procedure was applied to (4.1) on a uniform grid and on a non-uniform grid denser in the middle. Figures 4.3a and 4.3b show the behaviour of the error in the L^∞ norm (2.60e) for various steps of the procedure, with increasing grid refinement. Table 4.1 shows the convergence rates, computed as the coefficient of the linear regression between the logarithm of the maximum grid spacing and the logarithm of the error. It can be seen that second order accuracy is essentially preserved in the non-uniform case.
If greater accuracy is of interest, or if the grid's non-uniformity is more pronounced, an alternative to the four-point average is bilinear interpolation:



(a)



(b)

Figure 4.3: (a): Rates of convergence for the velocity gradient and the characteristic rate of strain on a uniform grid. (b): Rates of convergence on a non-uniform grid

	Uniform grid	Non-uniform grid
Convergence rate for $\nabla \boldsymbol{u}_{\mathrm{SG}}$:	1.999948434	1.999646606
Convergence rate for $\nabla \boldsymbol{u}_{\mathrm{PN}}$:	1.999919937	1.982622134
Convergence rate for $\overline{S}$:	1.987875690	1.987107335

Table 4.1: Convergence rates for the velocity gradient and for the characteristic rate of strain. $\nabla \boldsymbol{u}_{\mathrm{SG}}$ is the gradient at the centre of each interval, $\nabla \boldsymbol{u}_{\mathrm{PN}}$ is the gradient with all components interpolated at the pressure nodes.

given the four values $U_{1\,1}$, $U_{1\,2}$, $U_{2\,1}$ and $U_{2\,2}$ of a function U at the points (x_1, y_1), (x_1, y_2), (x_2, y_1) and (x_2, y_2), its value at (x, y) can be approximated as:

$$U \approx \frac{(x_2 - x)\,(y_2 - y)}{(x_2 - x_1)\,(y_2 - y_1)}\,U_{1\,1} + \frac{(x_2 - x)\,(y - y_1)}{(x_2 - x_1)\,(y_2 - y_1)}\,U_{1\,2} + \\ + \frac{(x - x_1)\,(y_2 - y)}{(x_2 - x_1)\,(y_2 - y_1)}\,U_{2\,1} + \frac{(x - x_1)\,(y - y_1)}{(x_2 - x_1)\,(y_2 - y_1)}\,U_{2\,2}\,, \tag{4.4}$$

which is the equation of the three-dimensional plane passing through the four reference points. (4.4) is second order accurate in space, and can be used to approximate the values of the components of the velocity gradient at the pressure nodes from four surrounding points, however spaced.
A comparison between (4.4) and (4.3) on the same non-uniform grid used for the previous test shows that (4.4) produces more accurate results. This can be seen in figure 4.4. However, as evidenced in table 4.2, the order of accuracy is lower compared to that of (4.3). This may be due to the greater amount of floating point operations required by (4.4). Oddly enough, an

	Bilinear interpolation
Convergence rate for $\nabla \boldsymbol{u}_{\mathrm{SG}}$:	1.999646606
Convergence rate for $\nabla \boldsymbol{u}_{\mathrm{PN}}$:	1.933820679
Convergence rate for $\overline{S}$:	1.889218352

Table 4.2: Convergence rates for the velocity gradient and for the characteristic rate of strain with bilinear interpolation on a non-uniform grid.

earlier implementation of (4.4), containing IF constructs inside DO loops and calls to another subroutine for each point, proved to be slightly faster than any further attempt at optimization.

4.2 Computing the velocity gradient

The previous tests were performed writing subroutines that made abundant use of multi-dimensional arrays, however, as explained in 3.3, this approach is not compatible with the way data is represented inside the software in

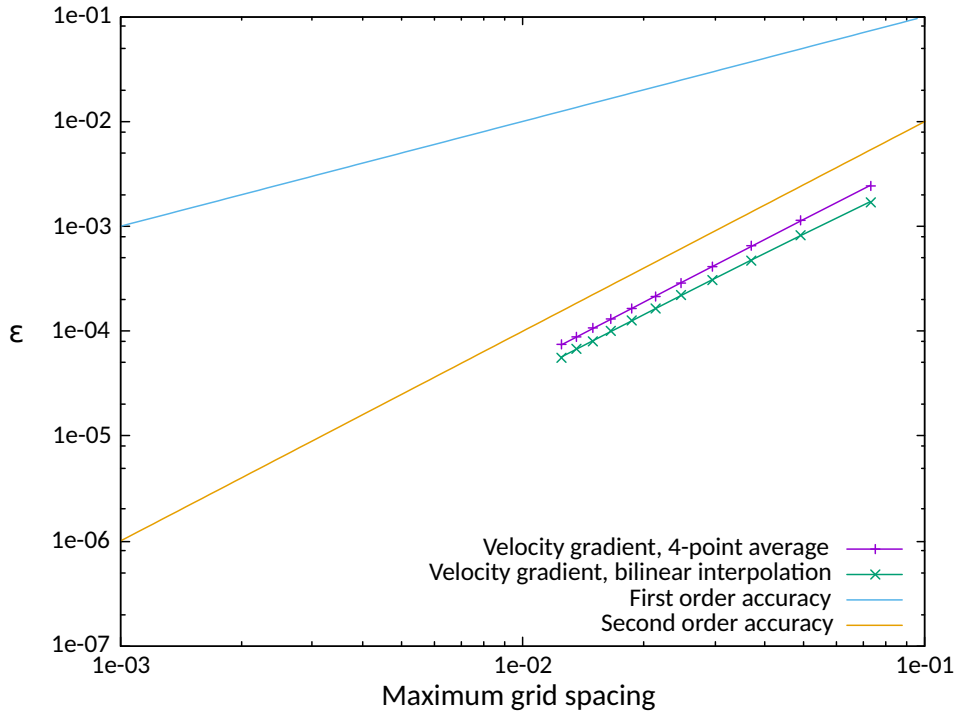


Figure 4.4: Rates of convergence for the velocity gradient at the pressure nodes, with four-point averaging and bilinear interpolation.

use. As a consequence, the subroutines for the computation of the velocity gradient inside the program had to be entirely re-written. In order to test these subroutines while they were being developed, a "reduced environment" was set up by switching off most functionalities from the program, and leaving only the subroutines strictly necessary for defining the data structures, operating on the grid and generating the MPI communicators. The module prescribing the initial conditions was also included, modified so that it would generate the velocity field (4.1). In this environment, the parallelization aspects of the problem could be tested as well.

In order to illustrate how the procedure was developed, it may be beneficial to explain how the domain is divided between processes. Each process is assigned, for each direction, a number of internal points, specified in the file *data*, however, it stores two additional points that it either shares with another process or with a boundary of the domain. The situation may appear clearer from figure 4.5. While the figure shows a two dimensional grid, the three-dimensional case is similar.

The computation of the velocity gradient was organized in three phases: the first one involving only the internal points of each process, the second one applying the boundary conditions at the external boundaries and the third one

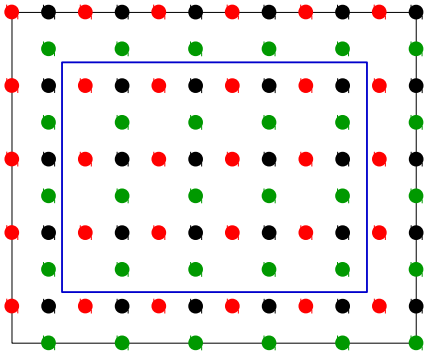


Figure 4.5: The grid assigned to one process. The red points represent the grid for u, the green ones the grid for v, and the black ones the pressure grid. The blue square surrounds the internal points, the nodes outside of it are either shared with another process or with a boundary.

dealing with communication between processes to complete the derivatives at the internal boundaries. All these three steps were collected in the subroutine *LESVelocityGradient* in the file *./SOURCES/LESEddyViscosity.f90*

4.2.1 internal points

The treatment of the diagonal components of the velocity gradient is immediate. The pressure nodes fall exactly halfway between two consecutive velocity nodes, along that particular velocity component's direction, therefore a simple centred finite difference is sufficient:

$$\left.\frac{\partial u}{\partial x}\right|_{i,\,j,\,k} \approx \frac{u_{i+\frac{1}{2},\,j,\,k} - u_{i-\frac{1}{2},\,j,\,k}}{h_i}\,. \tag{4.5a}$$

For the diagonal terms, a slightly different approach than the one used during testing was employed. Instead of using expressions like (4.5a) and then averaging (or interpolating) the results at a later time, a 6-point stencil was used to compute the derivative at the desired point in a single passage:

$$\left.\frac{\partial u}{\partial y}\right|_{i,j,k} \approx \frac{1}{2}\left(\frac{h_{j-\frac{1}{2}}}{h_{j+\frac{1}{2}}+h_{j-\frac{1}{2}}}\right)\left[\frac{u_{i-\frac{1}{2},j+1,k}-u_{i-\frac{1}{2},j,k}}{h_{j+\frac{1}{2}}}+\frac{u_{i+\frac{1}{2},j+1,k}-u_{i+\frac{1}{2},j,k}}{h_{j+\frac{1}{2}}}\right]+$$
$$+\frac{1}{2}\left(\frac{h_{j+\frac{1}{2}}}{h_{j+\frac{1}{2}}+h_{j-\frac{1}{2}}}\right)\left[\frac{u_{i-\frac{1}{2},j,k}-u_{i-\frac{1}{2},j-1,k}}{h_{j-\frac{1}{2}}}+\frac{u_{i+\frac{1}{2},j,k}-u_{i+\frac{1}{2},j-1,k}}{h_{j-\frac{1}{2}}}\right] \tag{4.5b}$$

The expression above can be obtained inserting four expressions in the form (4.5a) into (4.4). It can be seen from figure 4.5 that, at the last pressure nodes along the direction of u, both (4.5a) and (4.5b) would be missing the

upper values $u_{N_x+\frac{1}{2},j,k}$. Therefore, the derivatives at these last points are not computed in this phase. They are either passed from the process ahead, or computed from the boundary conditions. Similarly, at both the lower and upper boundaries along the derivation direction, only the upper or lower half of (4.5b) can be computed. The remaining portion will either be exchanged from the bordering process or computed using the boundary conditions.

4.2.2 boundary conditions

The assignment of boundary conditions is more complex than the treatment of the internal points. Different methods are needed depending on the direction of the velocity component, the derivation direction and the direction normal to the boundary considered. Treatments at the lower boundaries may differ from those at the upper ones, and Neumann boundary conditions have to be implemented separately from Dirichlet ones.
The simplest case is that of diagonal components of $\nabla \boldsymbol{u}$, which are already known at every boundary except the last one along their velocity component's direction. At these boundaries, (4.5a) can be applied, using the value of u at the ghost point right outside the boundary, computed from the exact solution provided in *./CODE/solutions.f90*. This strategy is used both for Dirichlet and Neumann boundary conditions, since the latter are conditions on the velocity derivative along the boundary normal, which is exactly what is being considered. For off-diagonal components, in the case of Dirichlet boundary conditions, if the boundary normal is aligned with the derivation direction, the missing half of (4.5b) is computed from the values $u_{i-\frac{1}{2},j\pm1,k}$ and $u_{i+\frac{1}{2},j\pm1,k}$ (with j equal to either 1 or N_y) at two ghost points; if the boundary normal is aligned with the velocity component, at the top boundary, the entire expression (4.5b) is computed from the last values of u before the boundary, plus the values $u_{N_x+\frac{1}{2},j-1,k}$, $u_{N_x+\frac{1}{2},j,k}$ and $u_{N_x+\frac{1}{2},j+1,k}$ at three ghost points right after the boundary. In the case of Neumann boundary conditions, the derivation stencils for off-diagonal components are moved "off-center" at the boundaries, so that they reference values inside the domain only.
At each boundary, for components of $\nabla \boldsymbol{u}$ for which neither the velocity component nor the derivation direction coincide with the boundary normal, no treatment is required, since their stencils do not include any value outside the domain.

4.2.3 communication

The communication between processes is split into two different events. The first one involves the exchange of the incomplete 6-point stencil derivatives of the off-diagonal components (4.5b), which are then summed up. After this first event, each process knows all components of $\nabla \boldsymbol{u}$ at the internal points

but, for each velocity component, it is missing all its three derivatives at the top boundary along that component's direction. The second communication event involves sending these missing values from the bottom boundaries of each process to the preceding processes.
A slightly different approach to communication is employed, compared to the rest of the program. Instead of a cascade of messages starting on one side of the domain and reaching the other, an alternating pattern is used: first odd-ranked processes send while even-ranked ones receive, then vice versa:

```
! Even sends, odd receives
IF (MOD(one_d_rank(k2), 2) == 0 .AND. mesh%BCtype(1, k2) == 0) THEN
  CALL MPI_SEND(buffer(:, 1), Ntot, MPI_DOUBLE_PRECISION, (one_d_rank(
      k2) - 1), tag, comm_one_d(k2), code)
ELSE IF (MOD(one_d_rank(k2), 2) == 1 .AND. mesh%BCtype(2, k2) == 0)
    THEN
  CALL MPI_RECV(buffer(:, 2), Ntot, MPI_DOUBLE_PRECISION, (one_d_rank(
      k2) + 1), tag, comm_one_d(k2), status, code)
END IF

! Odd sends, even receives
IF (MOD(one_d_rank(k2), 2) == 1 .AND. mesh%BCtype(1, k2) == 0) THEN
  CALL MPI_SEND(buffer(:, 1), Ntot, MPI_DOUBLE_PRECISION, (one_d_rank(
      k2) - 1), tag, comm_one_d(k2), code)
ELSE IF (MOD(one_d_rank(k2), 2) == 0 .AND. mesh%BCtype(2, k2) == 0)
    THEN
  CALL MPI_RECV(buffer(:, 2), Ntot, MPI_DOUBLE_PRECISION, (one_d_rank(
      k2) + 1), tag, comm_one_d(k2), status, code)
END IF
```

The comparison between the velocity gradient produced in this way and the exact solution shows that these subroutines have similar convergence properties to those developed for the preliminary work, as can be seen from table 4.3 or from figures 4.6a and 4.6b.

4.3 Computing the eddy viscosity

A simple subroutine for computing the characteristic rate of strain given the velocity gradient was developed, named *LES_ CSR*, employing the expression:

$$\overline{S} = \sqrt{\frac{\partial u_i}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)}\,. \tag{4.6}$$

Another subroutine combining the grid spacings to obtain the characteristic filter width at each pressure node was developed, named *LES_ cfw*. This subroutine allows for choosing between the geometric mean:

$$\Delta = \sqrt[3]{\Delta_x\,\Delta_y\,\Delta_z}\,, \tag{4.7}$$

and the arithmetic mean:

$$\Delta = \frac{\Delta_x + \Delta_y + \Delta_z}{3}\,. \tag{4.8}$$

The only missing piece for computing the turbulent viscosity (1.31) is the Smagorinsky constant C_S. A reference value of 0.1 is provided in the file *data*, however, the possibility of a varying C_S is included.
One of the arguments of the subroutine developed for combining all the previous results into the turbulent viscosity, *LES_ nu_ t*, is the scalar field Cs. An additional logical argument, Cs_var, specifies whether the subroutine should consider the local values contained in Cs as the Smagrinsky constant, or apply the reverence value provided in *data* everywhere and at all times. In the latter case, the field Cs can be passed uninitialized, since it is not used. This strategy allows for a certain flexibility in the definition of C_S.
For the tests performed in this work, either the reference value was used, or a simple C_S field was defined at the beginning, and left constant throughout the computation. However, it is possible to implement methods for recomputing C_S at each time step, either using more complex wall treatments, or a dynamic model of the type described in 1.3.5.

4.4 The sub-grid scale stresses

The sub-grid scale stresses are defined as

$$\boldsymbol{\tau}^{SGS} = \nu_t \left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T \right) ; \tag{4.9}$$

their effect on the momentum equation is computed as the divergence of (4.9). As explained in 2.3.4, it is possible to take the portion

$$\nabla \cdot \{ \nu_t \left[\nabla \boldsymbol{u} + \mathrm{diag} \left(\nabla \boldsymbol{u} \right) \right] \}$$

of the turbulent diffusion and include it, along with molecular diffusion, in the explicit ADI scheme, while the mixed term derivatives

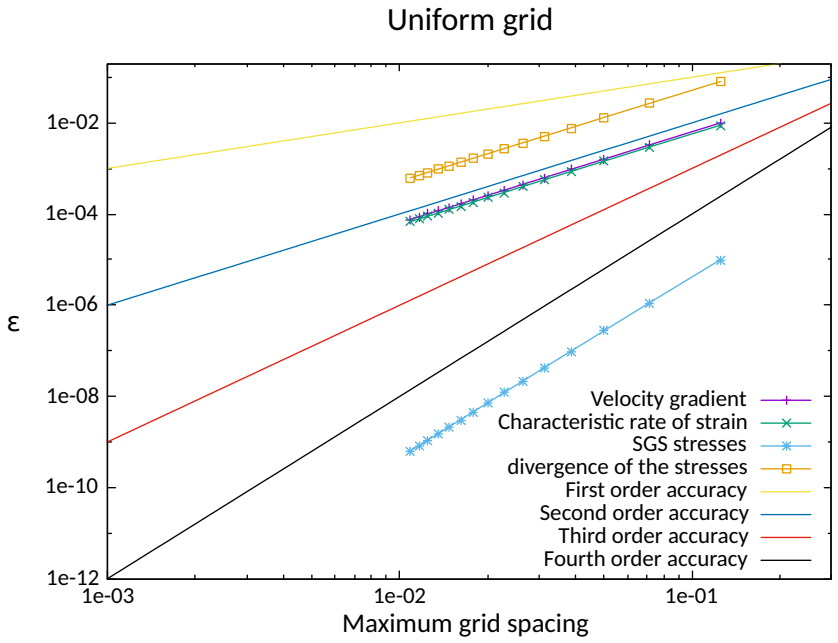
$$\nabla \cdot \left\{ \nu_t \left[\nabla \boldsymbol{u}^T - \mathrm{diag} \left(\nabla \boldsymbol{u} \right) \right] \right\}$$

have to be treated explicitly. This is the strategy employed in [26], and it was the original intention of this work. However, it became clear as the work proceeded that modifying the code for the inclusion of a variable viscosity would considerably alter the nature of the numerical scheme. First of all, the Schur component could not be computed at the beginning and left unaltered throughout the computation any more, it would have to be re-computed at each time step. Secondly, aside from having to be updated as well, the solution matrices would have to be considerably larger. In the current formulation, in fact, a single solution matrix to each one-dimensional problem is stored, being the same for all one-dimensional "segments" along the other two dimensions. With a non-uniform viscosity field, a number of solution matrices equal to the number of points along the other two dimensions would have to be computed for each dimension. All these modifications

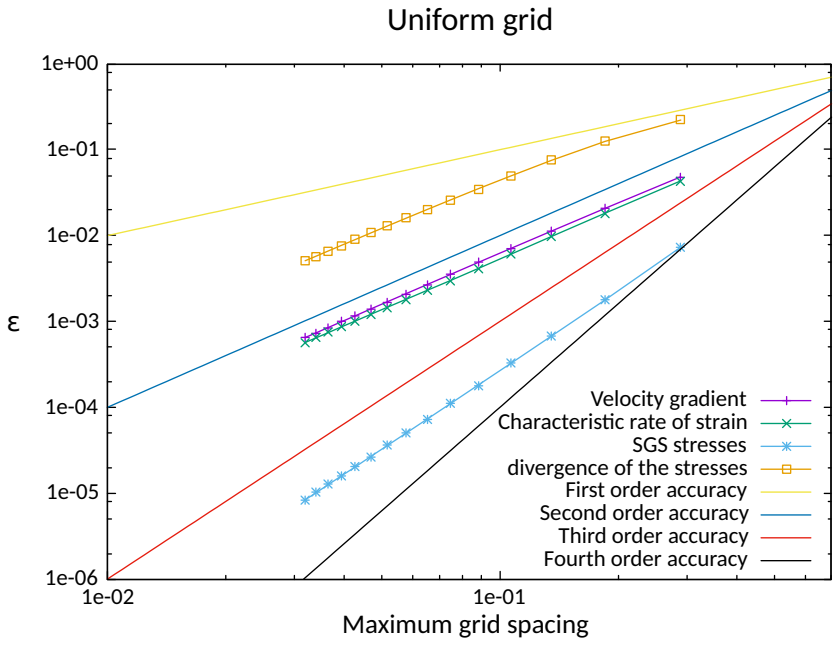
are possible and, if implemented carefully, may not penalize excessively the performance of the code. However, for the present work, it was chosen to treat the turbulent diffusion explicitly.
The subroutine *LES_SGS_Stresses* was developed to compute the sub-grid scale stresses (4.9). The option to compute only $\nu_t \left[\nabla \boldsymbol{u}^T - \mathrm{diag}\left(\nabla \boldsymbol{u}\right)\right]$ was included, so that it could be re-used if, in the future, an explicit treatment of $\nabla \cdot \{\nu_t \left[\nabla \boldsymbol{u} + \mathrm{diag}\left(\nabla \boldsymbol{u}\right)\right]\}$ is developed.
To compute the divergence of the stress tensor, the subroutine *LES_div_tensor* was developed. As all the other subroutines developed mentioned so far, it was included in the file *./SOURCES/LESEddyViscosity.f90*. This subroutine takes as input only one column of the stress tensor at a time, and adds its divergence to the right-hand side of the component of the momentum equation being considered. *LES_div_tensor* employs finite differences similar to those used in *LESVelocityGradient*, however, no treatment of the boundaries or communication between processes was included. At the external boundaries, since the expression of the turbulent diffusion is not necessarily known analytically, the stencils for the finite differences were moved off-center to include known values only, similarly to the way in which Neumann boundary conditions are applied in *LESVelocityGradient*. At internal boundaries, analogously to *LESVelocityGradient*, the derivatives are either computed on one side only, or each side computes its own half. However, these partial results are not assembled, since they are added to rhs inside the main time cycle, which is assembled at passage 8(d) of 3.4. A subroutine to assemble the partial results of *LES_div_tensor* through inter-process communication was developed separately anyway, but simply to be able to test its accuracy.
As can be seen in table 4.3, *LES_div_tensor* is less accurate and has a lower order of convergence compared to the other subroutines, especially on a non-uniform grid. This is due to the fact that while pressure nodes are placed exactly half-way between consecutive velocity nodes, the converse is not true on a non-uniform grid. *LES_div_tensor* computes, at the velocity nodes, derivatives between values placed at the pressure nodes. Some of these derivatives are computed assuming that the velocity nodes are half-way between the pressure nodes regardless of whether the grid is uniform or not. It was assumed that the loss in accuracy resulting from this would be tolerable, in order to avoid the increased complexity of implementing three-point stencil non-centred finite differences.

4.5 Convergence rates

In order to test the accuracy of the subroutines developed, exact solutions for all the relevant quantities, for the velocity field (4.1), were implemented in the file *./SOURCES/LESSolutions.f90*. Figures 4.6a and 4.6b show the



(a)



(b)

Figure 4.6: (a): Rates of convergence for the quantities computed on a uniform grid. (b): Rates of convergence on a non-uniform grid

behaviour of the errors with increasing grid refinement. The convergence rates are reported in table 4.3.
It can be noted that the turbulent viscosity and the SGS stresses are third

	Uniform grid	Non-uniform grid
Convergence rate for $\nabla \boldsymbol{u}$:	1.995688	1.962728
Convergence rate for $\overline{S}$:	1.997826	1.967738
Convergence rate for the eddy viscosity ν_t :	4.005001	3.130042
Convergence rate for the SGS stresses :	3.960816	3.062371
Convergence rate for $\nabla \left[\nu \left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right)\right]$:	1.994265	1.765501

Table 4.3: Convergence rates for the subroutines added to the program.

order accurate, or even fourth order accurate on a uniform grid. This is due to the presence of the characteristic filter width, which is of the same order of magnitude as the squate of the grid spacing. For a uniform grid, in fact:

$$\nu_t^{fd} = (C_S\,\Delta)^2\,\left[\overline{S} + \mathrm{O}\left(h^2\right)\right] = C_S^2\,\Delta^2\,\overline{S} + C_S^2\,\Delta^2\,\mathrm{O}\left(h^2\right) = \nu_t + \mathrm{O}\left(h^4\right)\,,$$

while for a non-uniform grid:

$$\begin{aligned}\nu_t^{fd} &= \left[C_S\,\left(\Delta + \mathrm{O}\left(h^2\right)\right)\right]^2\,\left[\overline{S} + \mathrm{O}\left(h^2\right)\right] = \\ &= C_S^2\,\Delta^2\,\overline{S} + 2\,C_S^2\,\Delta\,\mathrm{O}\left(h^2\right)\,\overline{S} + ... = \nu_t + \mathrm{O}\left(h^3\right)\,,\end{aligned}$$

The same does not apply to the rate of convergence of $\nabla \left[\nu \left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right)\right]$, since for its test a unitary viscosity was imposed.

4.6 Inclusion in the code

The initialization of the necessary quantities, including the possible C_S field, was placed between passages 5 and 6 of 3.4; their deallocation between passages 9 and 10. Passages 6 and 9 were modified so that the eddy viscosity would be one of the variables loaded and stored for post-processing and restart. The computation of the velocity gradient, the eddy viscosity and the sub-grid scale stresses inside the time loop was added between passages 8(b) and 8(c), before the loop over the components of the momentum equation starts. The computation of the divergence of the current column of the SGS stress tensor was positioned at the end of passage 8(c), right after the computation of the advection term. A further block for the output of intermediate results was added after passage 8(i).
Initially, the velocity gradient, and therefore all quantities descending from it would be computed from a velocity field projected at step $t + \frac{1}{2}\Delta t$ with the expression

$$\boldsymbol{u}^{n+\frac{1}{2}} = \frac{3}{2}\,\boldsymbol{u}^n - \frac{1}{2}\,\boldsymbol{u}^{n-1}\,.$$

However, since the first time-marching tests of the code showed stability problems, this approach was abandoned: the eddy viscosity would be computed at each step from the current velocity field, and its value stored for ten time steps. The SGS stresses would be computed using the velocity gradient at time t and an eddy viscosity averaged over the last ten time steps.
Later, an option was added to choose the number of previous time steps over which the averaging of the eddy viscosity is done. Choosing one, the value at time step t is used without any averaging. For the option "zero" the possibility of projecting ν_t at time step $t + \frac{1}{2}\Delta t$ was re-introuduced. However, the gradient of the velocity in use for computing the SGS stresses remains the one at time step t, so, in retrospect, this re-introduction may be redundant. When excessive spatial variations in ν_t were suspected to be the cause of further instabilities, an option for keeping ν_t outside of the divergence of the sub-grid scale stresses was added: *LES_SGS_Stresses* would be passed a unitary viscosity, *LES_div_tensor* would store the divergence of the resulting tensor in a temporary array, which would then be multiplied by a properly reordered eddy viscosity, and added to rhs. Spatial accuracy might suffer from this approach, since the eddy viscosity, placed on the pressure nodes, is unceremoniously multiplied by diffusion terms placed on the velocity nodes.

Chapter 5

Test case and results

5.1 Case set up

All tests were run on the same case: the flow around a sphere of unitary diameter, at a Reynolds number of 3700, with a time step equal to 0.001 time units. The computational domain was the one illustrated in figure 5.1. The boundary conditions were of the Dirichlet type everywhere except at the

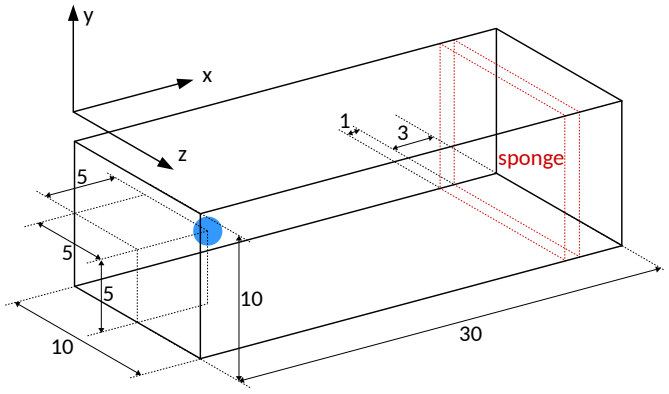


Figure 5.1: An outline of the computational domain used for the tests.

upper boundary along the x axis, on which Neumann boundary conditions were imposed. The velocity field at the boundary was unitary in the x direction, null in all other directions, and gradually increasing from 0 to 1 during the first time unit of simulation. This set up was chosen to be close to the one used in [28], so that their results could be used for comparison. There are, however, some key differences: in [28] a direct numerical simulation is performed, without any turbulence model, and an unstructured grid is used on a cylindrical domain. Most of the results found in [28] were collected between 200 and 550 time units of simulation, while the results collected in this work come from an earlier phase of the simulation. It was assumed,

however, that the wake would be developed enough after 65 time units.

5.1.1 forcing terms

If the same restart files were to be used to launch multiple simulations, they would produce exactly the same results. For the purpose of collecting flow statistics, this is undesirable. In order to be able to sightly stir up a simulation, so that in the long term it would produce unpredictable results, the source term in the file *./CODE/solutions.f90* was modified to apply a weak, random force on the fluid. The main executable *./CODE/pCNST.f90* was modified so that after one time unit it would turn off the source term.
Neumann boundary conditions at the outlet of the computational domain allow for the wake of the sphere to evolve freely up to the end. Turbulent fluctuations, however, may be reflected at the boundary and cause instabilities. To avoid this, a forcing term referred to as sponge was applied on the last three sphere diameters of the domain along the x direction (with an additional transitory sphere diameter). The sponge has the purpose of penalizing deviations from a reference velocity field, in this case, the one prescribed by the Dirichlet boundary conditions.

5.1.2 The Smagorinsky Constant

The original intention was to set the Smagorinsky constant to zero in the first portion of the domain, then have it increase asymptotically from 0 to 1 after the sphere. At $\mathrm{Re} = 3700$, the flow around the sphere is still laminar, while the wake is turbulent. The portion of the domain up to the sphere could theoretically be simulated without a turbulence model, therefore avoiding the problem of properly damping C_S at the sphere's surface. This strategy, however, was abandoned almost immediately, in favour of a constant, uniform C_S. Later, the possibility of using a simple implementation of the Van Driest damping function (1.33) was added: instead of using the local stresses on the sphere computed during the simulation, however, a constant distribution of stresses, interpolated from the ones reported in [28], was used to compute the viscous length. The C_S field computed in this way could be initialized at the beginning of the computation and kept unchanged.

5.1.3 The hardware

The initial tests were performed on the author's laptop, having a four-core (eight with multithreading) Intel Core i7-6700HQ with a reported clock speed of 2.60 Giga-Hertz and eight Gigabytes of RAM. Later tests were performed on a machine at the Dipartimento di Scienze e Tecnologie Aerospaziali, having two ten-core Intel Xeon CPU E5-4620 v4 (for a total of 40 cores with multithreading) running at 2.10 Giga-Hertz and having 252 Gigabytes of RAM.

5.2 Grid choice

The first attempts were run on a grid that was uniform around the sphere, and decreased in resolution away from it. It consisted of 201 points along the x direction, and 101 points along directions y and z. The domain was split among eight processors. The grid was set up so that at least 50 points were concentrated around the sphere. The jump in grid resolution obtained in this way proved to be excessively steep and caused the computation to destabilize after a few time units. A more smoothly varying grid with the same numbers of nodes along each direction was constructed, but it did not improve the situation. Nor did the addition of the time averaging of ν_t explained in 4.6. In the end, an attempt was done with a grid of the same size generated through a third degree polynomial. The reasoning behind this choice was that the previous grids were generated as piecewise curves, continuous up to the first order derivative but having discontinuous second order derivatives. The characteristic filter width used for computing ν_t is a function of the grid spacings, which vary like the first order derivative of the grid density. Therefore, when computing the divergence of the sub-grid scale stresses, the discontinuous derivative of the characteristic filter width was encountered. Providing a grid density function continuous up to the second order derivative would solve this problem.
A test launched on this polynomial grid ran without instabilities for 25 time units. Its results were used to seed all subsequent simulations.

Figure 5.2: u component of the velocity field at 25 time units of simulation. It can be noted that the wake is not fully developed yet. It can also be noted that the velocity is not null inside the sphere.

5.3 Multiple runs

All the simulations up to 5.5 were run on the 201x101x101 polynomial grid mentioned above, split among 8 processors, with a uniform C_S. Some attempts were made to launch simulations using the empirical Van Driest

damping function described in 5.1.2, however, they proved to be unstable. The considerable jump in velocity at the immersed boundary coupled with the absence of proper damping resulted in very high values of ν_t on the sphere, as can be seen in figure 5.3. The simulation mentioned in 5.2 was re-

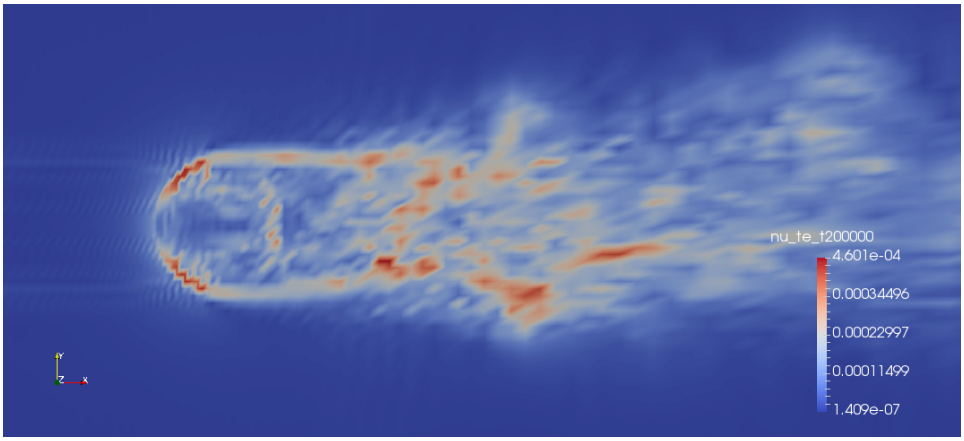
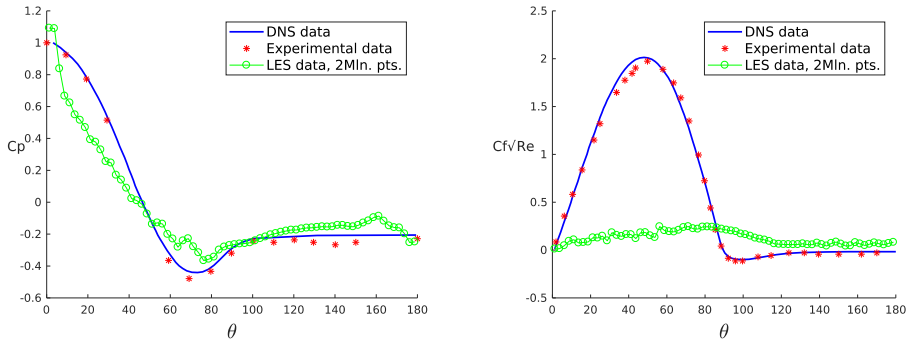


Figure 5.3: Turbulent viscosity field ν_t at 140 time units. The high values at the front of the sphere are a result of the absence of any wall treatment. Their "staircase" appearance hints that their presence is also an artefact of a not sufficiently fine grid.

launched. A second simulation, not employing any time averaging of ν_t, was launched from the intermediate results of the first one at 65 time units. The random forcing term was employed so that the resulting flow fields would differ. These two simulations seemed to be running smoothly, but diverged at 80 and 100 time units respectively. Any attempt to re-launch them from the last saved states resulted in divergence, suggesting that the cause of the instability was already present. The state at 65 time units, however, was deemed safe, since the two simulations, while both originating from it, had diverged at different times. To maximize the chances of obtaining meaningful results, five simulations were launched in parallel, all starting from the safe state at 65 time units, with the initial random forcing from 5.1.1 and averaging of ν_t over 100 time steps. For these simulations, ν_t was kept outside of the divergence of the SGS stresses. The results reported in 5.3.1 come from these simulations. Three out of five diverged at around 120 time steps, with one reaching 180 and another reaching 200 before diverging too.

5.3.1 Comparison of results

Figures 5.4a and 5.4b show the distribution of the pressure and friction coefficients around the sphere, confronted with the results reported in [28]. The results were collected at the Lagrangian points of the sphere, averaged over the azimuthal angle around the x axis, then averaged in time, then ensem-



(a) (b)

Figure 5.4: (a): Pressure coefficient on the sphere. (b): Friction coefficient normalized with $\mathrm{Re}^{-\frac{1}{2}}$ on the sphere.

ble averaged over all simulations performed. While the pressure coefficient does follow the experimental and DNS data, the friction coefficient is far off. This is probably due to the absence of wall damping for C_S: a greater viscosity is present at the sphere, resulting in a thicker boundary layer and lower stresses-per-dynamic-pressure. figure 5.5 shows a comparison between the values of the axial component of the velocity at various points in the wake reported in [28], and the ones collected during the previous simulations. Again, averaging is done along the azimuthal angle around the x axis and in time, as well as between different realizations of the simulation. The difference in behaviour decreases the further one moves from the sphere. All these results were collected between 65 and 200 time units of simulation.

5.4 Instability

In order to investigate the mechanism behind the instabilities encountered, the restart files stored by the longest running simulation, at 200 time units, were considered. If a simulation was to be launched from these restart values, it would diverge in about 2.8 additional time units. With this knowledge, a simulation was launched, with the program set up so that snapshots of the entire velocity field would be taken with increasing frequency as the instability approached. This would allow for a detailed observation of the development of the instability.
In the first few frames, no values outside the expected range could be seen, although the motor of the instability is already at play. Getting closer to the critical point, a halo of higher velocity could be seen forming around the diameter of the sphere, normal to the x axis and increasing in size. Upon closer inspection, a point with an anomalous value of the axial velocity was found. This point, as illustrated in figure 5.6, seems to be a vortex

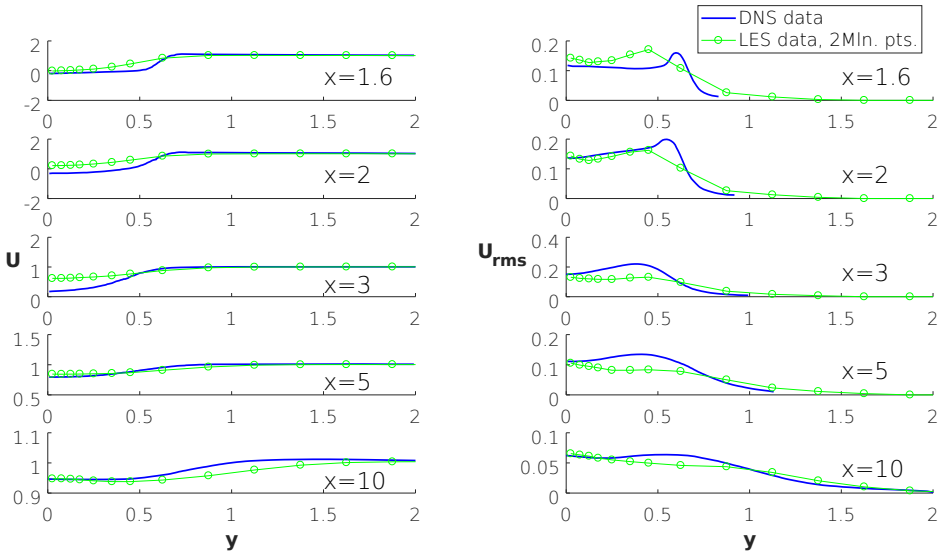


Figure 5.5: Average u component of the velocity along the transversal direction, at five points in the wake.

generated at the boundary layer which starts to move upstream and grows uncontrollably in intensity until it compromises the entire computation.

5.5 Tests at increased resolution around the sphere

It was hypothesized that the most likely cause of instabilities of the kind described in 5.4 was an insufficient grid resolution around the sphere. In the previous simulations, a quasi-stable situation was inadvertently obtained thanks to the absence of wall damping for C_S. The artificially inflated values

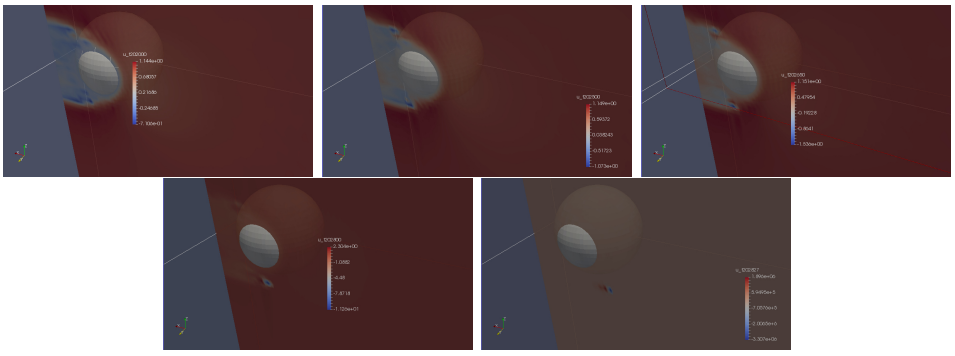
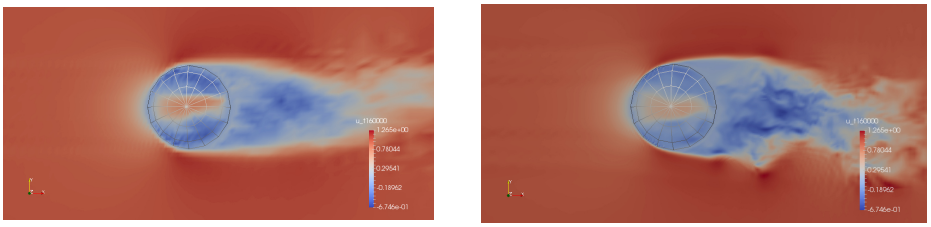


Figure 5.6: Five subsequent frames show how the instability departs from the boundary layer and grows in intensity while moving upstream.



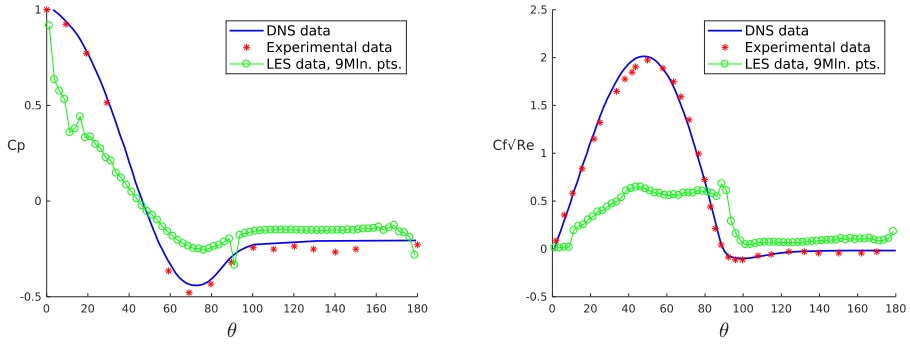
(a) (b)

Figure 5.7: (a): u velocity component around the sphere on the under-refined grid. (b): u velocity component around the sphere on the refined grid.

of ν_t on the sphere, observable in figure 5.3, would keep the instabilities resulting from the insufficient resolution at bay, but only up to a point. The simulation would eventually diverge when a sufficiently intense instability would appear. Attempts to use the Van Driest damping function on this insufficiently resolved grid resulted in the computation destabilizing almost immediately, because they removed the peak in turbulent viscosity on the sphere.
It was decided, in light of these considerations, to build a new grid. A greater number of points would be used: 333 along the x axis and 167 along the other two axes. The idea of having a uniform portion of the grid around the sphere was brought back, but this time the remaining portions of the grid were joined to it in a smoother way. The sphere was enclosed in a cube with a side 1.5 sphere diameters long, divided in 80 equally spaced intervals. This simulation would be split among 16 processors instead of 8.
To save time, data from restart files saved at 160 time units during one of the previous computations, and deemed stable enough, was linearly interpolated to produce restart files compatible with the new grid. A first simulation was launched, with the Van Driest damping function active, ν_t kept outside of the divergence of the SGS stresses and averaged over 100 time steps. Once it became clear that this simulation was more stable than the previous ones, a second one was launched, with ν_t inside the divergence and not averaged in time. Both simulations showed no sign of the previously encountered instabilities, and could be kept running seemingly indefinitely.

5.5.1 Comparison of results

During the two simulations performed on the refined grid, flow statistics were collected in the same way as during the previous simulations. The results, confronted with the ones from [28], are reported in figures 5.8a, 5.8b, 5.9 and 5.10. It can be seen that the situation of the friction coefficient has not improved significantly. These results were collected between 170 and 230 time units.



(a) (b)

Figure 5.8: (a): Pressure coefficient on the sphere, on the refined grid. (b): Friction coefficient normalized with $\mathrm{Re}^{-\frac{1}{2}}$ on the sphere, on the refined grid.

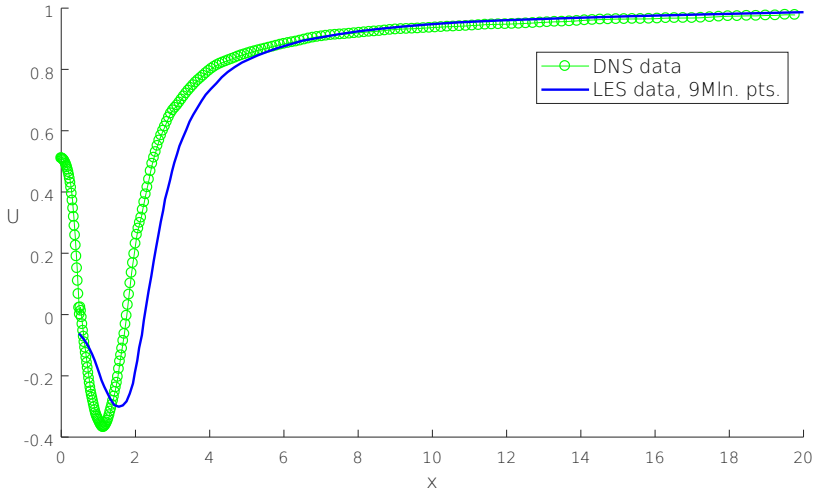


Figure 5.9: Average u component of the velocity along the x axis.

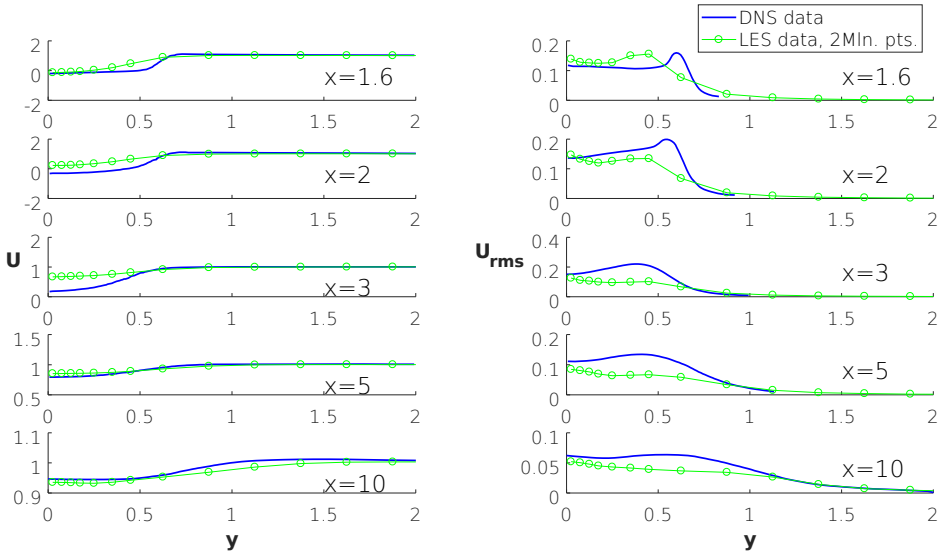
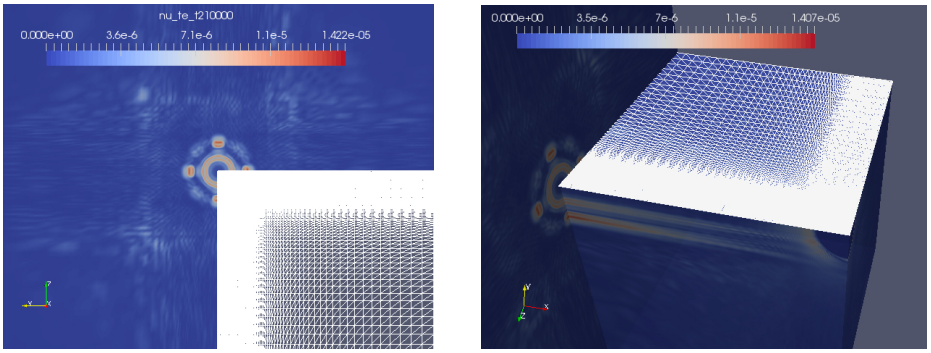


Figure 5.10: Average u component of the velocity along the transversal direction, at five points in the wake, computed on the refined grid.

5.6 Upstream disturbances

One distinct feature of all the flow visualizations presented so far is the appearance of straight, wave-like disturbances upstream of the sphere. While it is true that disturbances can travel upstream, the ones considered here have an artificial appearance. They seem to concentrate around areas in which the grid spacings vary the most, as the visualizations of the ν_t field in figures 5.11a and 5.11b show. They may be an artefact of subroutine *LES_div_tensor*'s lack of accuracy on non-uniform grids, and be visible only in the mildly disturbed portion of the flow ahead of the sphere. This hypothesis may be tested by running simulations on a uniform grid, or without the LES closure model turned on, to see if they disappear. If this is the case, attempts to improve the accuracy of subroutine *LES_div_tensor* by implementing more precise finite difference expressions could be made, to try to eliminate, or at least to reduce, these disturbances.



(a) (b)

Figure 5.11: (a): Disturbances in ν_t, front view. (b): Disturbances in ν_t, side view. A slice of the sphere can be seen on the right.

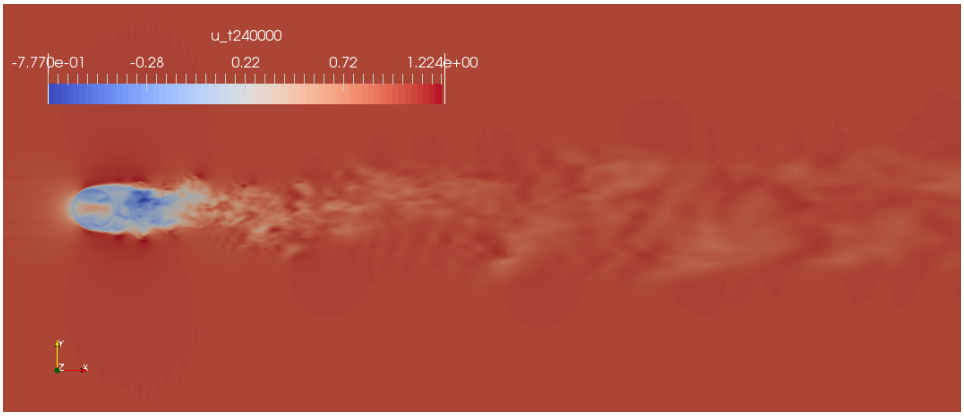


Figure 5.12: u component the velocity at 240 time units, computed on the refined grid.

Chapter 6

Final remarks

The software in use had already been thoroughly validated at Reynolds numbers ranging from 50 to 100. Most of its applications, at lest to the author's knowledge, were at Reynolds numbers in the order of a few hundreds.
Despite the preliminary nature of this work, it showed that the extension of the software's range of applicability to higher values of Re by means of a turbulence closure model is feasible. The Smagorinsky-Lilly closure model developed here ended up being simpler than originally intended. The critical aspect proved to be the performance of the model close to the sphere, which is to be expected for a large eddy simulation with no wall treatment. However, the end results of the simulations performed were not too far removed from the ones found in the literature.
There is a lot of potential for improvement in the approach developed here: a proper formulation of the Van Driest damping function could be added, for example, taking into account the actual stresses on the immersed surface, which are available inside the program. If an efficient strategy for determining the distance of a given grid point from an immersed surface is developed, such wall treatment could be employed on bodies of any shape, moving or otherwise. Alternatively, a hybrid method could be developed, with the interface between different closure models handled as an immersed boundary. A proper implicit treatment of turbulent diffusion is another possible development, and so is the addition of a dynamic model for determining C_S.
If the turbulence closure model introduced here is developed further, along with the ongoing improvement of the immersed boundary routines, the advantages of Guermond and Minev's direction splitting approach could be made available to whole new classes of problems.

Bibliography

[1] Luca Baldassarre. Parallelization of an immersed-boundary method with a direction splitting solver for time-dependent incompressible navier-stokes equation for fluid-structure interaction. Master's thesis, Politecnico di Milano, 2017.

[2] Luigi Carlo Berselli, Traian Iliescu, and William J. Layton. *Mathematics of large eddy simulation of turbulent flows*. Springer, 2005.

[3] Haecheon Choi and Parviz Moin. Grid-point requirements for large eddy simulation: Chapman's estimates revisited. *Physics of fluids*, 24(1):011702, 2012.

[4] Peter A. Davidson. *Turbulence: an introduction for scientists and engineers*. Oxford University Press, 2004.

[5] Marco D. de Tullio and Giuseppe Pascazio. A moving-least-squares immersed boundary method for simulating the fluid-structure interaction of elastic bodies with arbitrary thickness. *Journal of Computational Physics*, 325:201–225, 2016.

[6] James W. Deardorff. A numerical study of three-dimensional turbulent channel flow at large reynolds numbers. *Journal of Fluid Mechanics*, 41(2):453–480, 1970.

[7] Jim Douglas Jr. Alternating direction methods for three space variables. *Numerische Mathematik*, 4(1):41–63, 1962.

[8] Massimo Germano, Ugo Piomelli, Parviz Moin, and William H. Cabot. A dynamic subgrid-scale eddy viscosity model. *Physics of Fluids A: Fluid Dynamics*, 3(7):1760–1765, 1991.

[9] Jean-Luc Guermond and P. D. Minev. A new class of massively parallel direction splitting for the incompressible navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 200(23-24):2083–2093, 2011.

[10] Jean-Luc Guermond, Peter Minev, and Jie Shen. An overview of projection methods for incompressible flows. *Computer methods in applied mechanics and engineering*, 195(44-47):6011–6045, 2006.

[11] Brian Hahn. *Fortran 90 for scientists and engineers*. Elsevier, 1994.

[12] Francis H. Harlow and J. Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.

[13] John William Keating. *Direction-splitting schemes for particulate flows*. PhD thesis, University of Alberta (Canada), 2013.

[14] John Kim and Parviz Moin. Application of a fractional-step method to incompressible navier-stokes equations. *Journal of computational physics*, 59(2):308–323, 1985.

[15] Pijush K. Kundu, Ira M. Cohen, and D. W. Dowling. *Fluid Mechanics, 5th edition*. Elsevier, 2012.

[16] Randall J. LeVeque. Intermediate boundary conditions for lod, adi and approximate factorization methods. 1985.

[17] Randall J. LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. Siam, 2007.

[18] Rajat Mittal, Haibo Dong, Meliha Bozkurttas, F. M. Najjar, Abel Vargas, and Alfred Von Loebbecke. A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. *Journal of computational physics*, 227(10):4825–4852, 2008.

[19] Paolo Orlandi. *Fluid flow phenomena: a numerical toolkit*. Kluwer, 2000.

[20] Suhas Patankar. *Numerical heat transfer and fluid flow*. Hemisphere publishing, 1980.

[21] Stephen B. Pope. *Turbulent flows*. Cambridge university press, 2000.

[22] L. Quartapelle. *Numerical Solution of the Incompressible Navier-Stokes Equations*. Springer Basel AG, 1993.

[23] Alfio Quarteroni. Modellistica numerica per problemi differenziali, 5a edizione. 2012.

[24] Alfio Quarteroni, Riccardo Sacco, Fausto Saleri, and Paola Gervasio. *Matematica Numerica*. Springer, 2014.

[25] Rolf Rannacher. On chorin's projection method for the incompressible navier-stokes equations. In *The Navier-Stokes equations II — theory and numerical methods*, pages 167–183. Springer, 1992.

[26] Narsimha R. Rapaka and Sutanu Sarkar. An immersed boundary method for direct and large eddy simulation of stratified flows in complex geometry. *Journal of Computational Physics*, 322:511–534, 2016.

[27] Patrick J. Roache. *Fundamentals of Computational Fluid Dynamics.* Hermosa publishers, 1998.

[28] Ivette Rodriguez, Ricard Borell, Oriol Lehmkuhl, Carlos D. Perez Segarra, and Assensi Oliva. Direct numerical simulation of the flow over a sphere at re = 3700. *Journal of Fluid Mechanics*, 679:263–287, 2011.

[29] Pierre Sagaut. *Large eddy simulation for incompressible flows: an introduction.* Springer, 1998.

[30] Ulrich Schumann. Subgrid scale model for finite difference simulations of turbulent flows in plane channels and annuli. *Journal of computational physics*, 18(4):376–404, 1975.

[31] Jie Shen. On pressure stabilization method and projection method for unsteady navier-stokes equations. In *Advances in Computer Methods for Partial Differential Equations.* Citeseer, 1992.