TLab Documentation

The T group

April 15, 2024

Contents

Preface

This document has been derived from the DNS/CHEM document, originally created at the Computational Fluid Dynamics Laboratory at UC San Diego between 1999 and 2004 (`http://www.cfdlab.ucsd.edu/`). The work has been resumed at the Max Planck Institute for Meteorology since 2010 within the research group Turbulent Mixing Processes in the Earth System (`http://www.mpimet.mpg.de/`).

This manual is simply an introduction to the methodology and code, just a small part of the documentation of the set of tools TLab. The major part of the documentation is in the code itself in terms of `README` files, git version control system and comments within the source files. A set of examples has also been included for the user to get acquainted with the different tools (pre-processing, simulation and post-processing). Please note that, by default, all the different tools are continuously under development, so this document is continuously incomplete – the comments within the source files have always priority.

TLab—an acronym for Turbulence Laboratory—is a set of tools whose aim is **to efficiently solve and analyze a particular set of governing equations with a controlled accuracy**. The accuracy can be controlled in different ways: comparing with analytical solutions, including linear stability analysis; grid convergence studies; balance of transport equations, like integral turbulent kinetic energy or local values at specific relevant locations (e.g., at the wall). Resolution can be measured by the ratio between the grid spacing Δx and the relevant small scales, like the Kolmogorov scale η or the thickness of the diffusion sub-layers next to the wall. For the compact schemes used here, typical values are $\Delta x/\eta \simeq 1-2$; larger values can lead to numerical instability because of the aliasing generated by the non-linear terms. Note that these schemes are non-monotone, but typical out-of-bounds deviations of conserved scalars are below $10^{-6} - 10^{-8}$ relative to the mean variations, and this error is therefore negligibly small compared to the typical error associated with the statistical convergence, of the order of $1 - 5\%$. The statistical convergence can be estimated by varying the sample size of the data set, e.g. varying the domain size along the statistically homogeneous directions. The efficiency can be measured in different ways but, ultimate, it should be related with the computational time needed to understand a particular problem with a given accuracy, and so the importance of the controlled accuracy. Making the code user-friendly comes after the previous two main priorities: controlled accuracy and efficiency.

Regarding the content of this document, the first chapter describes the mathematical formulation, in particular, the governing equations. The boundary and initial conditions, discussed in chapter 2, are relatively simple for the geometries that we use; the major complexity is its actual implementation. Chapter 3 covers the code structure itself and the input data, the grid being discussed separately in chapter 4, and the postprocessing tools in chapter 5. As already mentioned, this part should be complemented with the examples included in the directory. Chapter 6 cover some major aspects of the numerical algorithms; details thereof, however, are to be found in the papers that are referred to in that part. The parallelization is discussed in chapter 8. So far, this includes only the domain decomposition. Last, scaling studies are included in chapter 9.

Contributors

Cedrick Ansorge

Alberto de Lózar

Jonathan Kostelecky

Juan Pedro Mellado

Lukas Müßle

Chiel van Heerwaarden

Part I

User Guide

1 Governing equations

1.1 Compressible formulation

Let us consider first an ideal gas with one single species. The evolution equation for the energy is written in terms of the specific internal energy (sensible plus formation). Viscosity, thermal conductivity, diffusivity and the specific heat ratio can depend on the temperature. The governing equations are written as follows:

$$\partial_t \rho = -\partial_k(\rho u_k) \tag{1.1a}$$

$$\partial_t(\rho u_i) = -\partial_k(\rho u_i u_k) + \mathbf{Re}^{-1}\partial_k \tau_{ik} - \partial_i p + \mathbf{Fr}^{-1}\rho g_i\, b + \mathbf{Ro}^{-1}\rho\epsilon_{ijk} f_k u_j \tag{1.1b}$$

$$\partial_t(\rho e) = -\partial_k(\rho e u_k) + \mathbf{Re}^{-1}\mathbf{Pr}^{-1}\partial_k\left(\lambda^*\partial_k T\right) - (\gamma_0 - 1)\mathbf{Ma}^2\, p\,\partial_k u_k + (\gamma_0 - 1)\mathbf{Ma}^2\mathbf{Re}^{-1}\phi \tag{1.1c}$$

$$\partial_t(\rho\zeta_i) = -\partial_k(\rho\zeta_i u_k) - \mathbf{Re}^{-1}\mathbf{Sc}_i^{-1}\partial_k j_{ik} \tag{1.1d}$$

with

$$\tau_{ij} \equiv \mu^*\left[\partial_j u_i + \partial_i u_j - (2/3)\,\partial_k u_k\,\delta_{ij}\right]\ , \qquad \phi \equiv \tau_{ij}\partial_j u_i\ , \qquad j_{ik} \equiv -(\rho D)_i^*\,\partial_k\zeta_i \tag{1.2}$$

and

$$\mu^* = T^{n_\mu}\ , \qquad \lambda^* = T^{n_\kappa}\ , \qquad (\rho D)_i^* = T^{n_{D,i}} \tag{1.3}$$

and

$$p = (\gamma_0\mathbf{Ma}^2)^{-1}\rho T\ . \tag{1.4}$$

The variables in these equations are normalized by the reference scales L_0, U_0, ρ_0 and T_0, which represent a length, a velocity, a density, and a temperature, respectively. The pressure is normalized by $\rho_0 U_0^2$. Thermal energy variables are normalized with $C_{p0}T_0$, where C_{p0} is a reference specific heat capacity at constant pressure. R_0 is the specific gas constant of the gas under consideration. The dimensionless numbers are defined by

$$\mathbf{Re} \equiv \frac{\rho_0 \mathrm{U}_0 \mathrm{L}_0}{\mu_0}\ , \qquad \mathbf{Pr} \equiv \frac{\mu_0}{(\lambda_0/C_{p0})}\ , \qquad \mathbf{Sc}_i \equiv \frac{\mu_0}{\rho_0 D_{i0}}\ , \tag{1.5}$$

and

$$\mathbf{Ma} \equiv \frac{\mathrm{U}_0}{\sqrt{\gamma_0 R_0 \mathrm{T}_0}}\ , \qquad \gamma_0 \equiv \frac{C_{p0}}{C_{p0} - R_0}\ , \tag{1.6}$$

and

$$\mathbf{Fr} \equiv \frac{\mathrm{U}_0^2}{g\mathrm{L}_0} \qquad \mathbf{Ro} \equiv \frac{\mathrm{U}_0}{\mathrm{L}_0 f}\ . \tag{1.7}$$

Although γ_0 and $\mathbf{Ma}$ are the input parameters, the governing equations depend on $\gamma_0\mathbf{Ma}^2$ and $(\gamma_0 - 1)\mathbf{Ma}^2$, which suggests to introduce the derived set of parameters

$$\gamma_0\mathbf{Ma}^2 = \frac{U_0^2}{R_0 T_0}\ , \qquad (\gamma_0\mathbf{Ma}^2)^{-1} = \frac{R_0}{U_0^2/T_0}\ , \qquad (\gamma_0 - 1)\mathbf{Ma}^2 = \frac{U_0^2/T_0}{C_{p0}}\ , \tag{1.8}$$

in the code implementation (names `MRATIO`, `RRATIO` and `CRATIO_INV`, respectively). They can be interpreted as a normalized gas constant, and a normalized heat capacity. The values of these parameters are defined in the procedure `termo_initialize`.

The vectors g_i and f_i need to be provided and should be unitary, so that the magnitude of the term is completely determined by the corresponding non-dimensional number. In this compressible case, the centrifugal term should probably be included; not yet studied.

Multispecies

Let us consider a mixture of N species or constituents with mass fractions Y_i:

$$\partial_t \rho = -\partial_k(\rho u_k) \tag{1.9a}$$

$$\begin{aligned}\partial_t(\rho u_i) = &-\partial_k(\rho u_i u_k) + \mathbf{Re}^{-1}\partial_k \tau_{ik} \\ &-\partial_i p + \mathbf{Fr}^{-1}\rho g_i\, b + \mathbf{Ro}^{-1}\rho \epsilon_{ijk} f_k u_j\end{aligned} \tag{1.9b}$$

$$\begin{aligned}\partial_t(\rho e) = &-\partial_k(\rho e u_k) + \mathbf{Re}^{-1}\mathbf{Pr}^{-1}\partial_k[(\lambda/C_p)^*\partial_k h] \\ &+\mathbf{Re}^{-1}\mathbf{Pr}^{-1}\partial_k\left[\sum\left(\mathbf{Le}_i^{-1}(\rho D)_i^* - (\lambda/C_p)^*\right) h_i \partial_k Y_i\right] \\ &-(\gamma_0-1)\mathbf{Ma}^2\, p\,\partial_k u_k + (\gamma_0-1)\mathbf{Ma}^2\mathbf{Re}^{-1}\phi\end{aligned} \tag{1.9c}$$

$$\partial_t(\rho\zeta_i) = -\partial_k(\rho\zeta_i u_k) - \mathbf{Re}^{-1}\mathbf{Sc}_i^{-1}\partial_k j_{ik} \tag{1.9d}$$

and

$$\mu^* = T^{n_\mu}\,, \qquad (\lambda/C_p)^* = T^{n_\kappa}\,, \qquad (\rho D)_i^* = T^{n_{D,i}} \tag{1.10}$$

and

$$\mathbf{Sc}_i = \mathbf{Le}_i\mathbf{Pr} \tag{1.11}$$

and

$$Y_i = Y_i^e(\zeta_j)\,, \qquad \sum_1^N Y_i = 1 \tag{1.12}$$

$$h = \sum_1^N h_i Y_i\,, \qquad h_i = \Delta h_i^0 + \int_{T_0}^T C_{pi}(T)dT\,, \qquad e = h - (\gamma_0-1)\mathbf{Ma}^2\frac{p}{\rho} \tag{1.13}$$

$$C_p = \sum_1^N C_{pi}(T)Y_i\,, \qquad \gamma = \frac{C_p}{C_p - \frac{\gamma_0-1}{\gamma_0}R} \tag{1.14}$$

and

$$p = (\gamma_0\mathbf{Ma}^2)^{-1}\rho T R\,, \qquad R = \sum_1^N R_i Y_i\,. \tag{1.15}$$

Each species has a specific heat capacity C_{pi}, a molar mass W_i, and a specific gas constant $R_i = \mathcal{R}/W_i$, where $\mathcal{R}$ is the universal gas constant. They are defined in `termo_initialize`. They are nondimensionalized by the reference values C_{p0}, W_0 and $R_0 = \mathcal{R}/W_0$, which are the values of one of the species. R is the specific gas constant of the gas mixture nondimensionalized by R_0 and $W = 1/R$ is the mean molecular weight nondimensionalized by W_0. As in the single species case, it proves convenient to introduce the derived parameters

$$\gamma_0\mathbf{Ma}^2 = \frac{U_0^2}{R_0 T_0}\,, \qquad (\gamma_0\mathbf{Ma}^2)^{-1} = \frac{R_0}{U_0^2/T_0}\,, \qquad (\gamma_0-1)\mathbf{Ma}^2 = \frac{U_0^2/T_0}{C_{p0}}\,, \tag{1.16}$$

in the code implementation (names `MRATIO`, `RRATIO` and `CRATIO_INV`, respectively) and use

$$\tilde{R} \equiv (\gamma_0 \mathbf{Ma}^2)^{-1} R = \sum_1^N Y_i \, \tilde{R}_i \ , \qquad \tilde{R}_i \equiv (\gamma_0 \mathbf{Ma}^2)^{-1} R_i \ , \tag{1.17}$$

to save computation time and facilitate the consideration of a dimensional formulation in addition to the default non-dimensional formulation here described (see below). Note that specific heat ratio

$$\gamma = \frac{C_p}{C_p - \frac{\gamma_0 - 1}{\gamma_0} \gamma_0 \mathbf{Ma}^2 \tilde{R}} \tag{1.18}$$

depends on the composition of the mixture, and γ_0 is no longer an independent input parameter but is determined by the mixture in `termo_initialize` as $\gamma_0 = C_{p0}/(C_{p0} - R_0)$. The parameter

$$\frac{\gamma_0 - 1}{\gamma_0} \gamma_0 \mathbf{Ma}^2 \tag{1.19}$$

is defined as `GRATIO` in the code and it is defined as $[(\gamma_0 - 1)/\gamma_0]$`MRATIO`. This definition allows us to switch between dimensional and nondimensional formulations easily. (Why do we need this additional parameter?)

Dimensional Formulation

In this case of a multi-species, a dimensional formulation can be considered by setting the input parameter `nondimensional` equal to `.false.` in the block `[Thermodynamics]` of the input file (by default, `tlab.ini`). The boundary and initial conditions defined in the input file should then be given in dimensional form. The code sets the parameters `RRATIO`, `MRATIO` and `CRATIO_INV` equal to 1, and the equations above correspond to those of the dimensional formulation.

The input parameters `Reynolds`, `Froude` and `Rossby` are then substituted by `Viscosity`, `Gravity` and `Coriolis`.

On the difference between the species and the prognostic scalars

The functions $Y_i^e(\rho, e, \zeta_j)$ need to be provided. We assume that $\sum_1^N Y_i = 1$ and use this relation to use $Y_N = 1 - \sum_1^N$ in the expressions above. For instance,

$$\tilde{R} = \sum_1^{N-1} Y_i (\tilde{R}_i - \tilde{R}_N) \ . \tag{1.20}$$

It is then clear that we do not need Y_N as prognostic variable. More generally, the total number of species is N, need not be equal to the total number of scalars with an evolution equation. The former can be considered as part of the diagnostic variables and the latter as part of the prognostic variables. This includes the simplest possible case of $Y_i^e = \zeta_i$ for $i = 1, \ldots, N-1$.

Another case is equilibrium, e.g. Burke-Schumann approximation, where mass fraction of all species is related to the mixture fraction variable, ζ. In this case, the functions $Y_i^e(Z)$ are smoothed around Z_s to reduce the strength of the discontinuity [Higuera and Moser, 1994]. The restriction of unity Lewis number the extra conditions $n_\mu = n_\kappa = n_D$ and $Sc = Pr$. These relationships are obtained by assuming equal diffusivity of all species and a single-step infinitely fast chemical reaction. For further discussion on the formulation see Williams [1985].

Another possible case is phase equilibrium (saturation adjustment), as considered for an misture of air and water, where only the total water content is a prognostic variables and the partition into liquid and vapor is done assuming phase equilibrium.

To reserve memory space for some of these diagnostic variables, the code implementation considers two global variables, namely, `inb_scal` and `inb_scal_array`. The first one is the number of prognostic variables, and the latter is the number of prognostic and diagnostic variables. In principle, this should be specified through the variable `imixture` in the input file. The average statistical data is calculated for all of them, prognostic and diagnostic variables.

Reacting flows

Add reaction terms to the scalar equations

$$\partial_t(\rho\zeta_i) = -\partial_k(\rho\zeta_i u_k) - \mathbf{Re}^{-1}\mathbf{Sc}_i^{-1}\partial_k j_{ik} + \mathbf{Da}_i\, w_i \tag{1.21}$$

and $\mathbf{Da}_i$ are the Damköhler numbers. A reaction mechanism needs to be given to obtain $w_i(\rho, e, \zeta_j)$.

1.2 Incompressible formulation

We need to solve a Poisson equation for the pressure.

Anelastic formulation

Dynamics and thermodynamics are still coupled by the background density and the buoyancy field. The evolution equations are:

$$0 = -\partial_k(\rho_{\rm bg}u_k) \tag{1.22a}$$

$$\partial_t u_i = -\partial_k(u_i u_k) - \rho_{\rm bg}^{-1}\partial_i p' + \mathbf{Re}^{-1}\rho_{\rm bg}^{-1}\partial_k\left(\mu^*\partial_k u_i\right) + \mathbf{Fr}^{-1}\, g_i\, b + \mathbf{Ro}^{-1}\,\epsilon_{ijk} f_k\, u_j \tag{1.22b}$$

$$\partial_t \zeta_i = -\partial_k(\zeta_i u_k) + \mathbf{Re}^{-1}\mathbf{Sc}_i^{-1}\,\rho_{\rm bg}^{-1}\partial_k\left(\mu^*\partial_k\zeta_i\right) + \mathbf{Da}_i\,\rho_{\rm bg}^{-1} w_i \tag{1.22c}$$

where the buoyancy is

$$b = \rho^{-1}(\rho_{\rm bg} - \rho) \approx \rho_{\rm bg}^{-1}(\rho_{\rm bg} - \rho)\ . \tag{1.23}$$

The dynamic pressure p' satisfies the following Poisson equation:

$$\partial_i\partial_i p' = \partial_i\left[\rho_{\rm bg}\left(-\partial_k(u_i u_k) + \mathbf{Re}^{-1}\rho_{\rm bg}^{-1}\partial_k\left(\mu^*\partial_k u_i\right) + \mathbf{Fr}^{-1}\, g_i\, b + \mathbf{Ro}^{-1}\,\epsilon_{ijk} f_k\, u_j\right)\right]\ . \tag{1.24}$$

So far, only the case $\mu^* = 1$ in the equations above has been implemented. The variables in these equations are assumed to be nondimensionalized by ρ_0, U_0 and L_0.

The thermodynamics is defined by the background profiles $\{\rho_{\rm bg}, p_{\rm bg}, T_{\rm bg}, Y_{\rm bg}\}$, which are steady and satisfy the hydrostatic balance equation,

$$\partial_2 p_{\rm bg} = -\mathbf{H}^{-1}\, g_2\, \rho_{\rm bg}\ , \qquad p_{\rm bg}|_{x_2=x_{2,0}} = p_{{\rm bg},0}\ , \tag{1.25}$$

and the thermal equation of state,

$$p_{\rm bg} = \rho_{\rm bg} R_{\rm bg} T_{\rm bg}\ . \tag{1.26}$$

$\mathbf{g}$ is defined opposite to the gravitational acceleration (the problem is formulated in terms of the buoyancy). The background profile information in the input file `tlab.ini`. The values $x_{2,0}$ and $p_{{\rm bg},0}$ are provided through the background profile information for the pressure.

The two equations above relate 4 thermodynamic variables, so we need two additional constraints. Typically, we impose the background profile of static energy (enthalpy plus potential energy) and the composition. In this case, the first scalar is the static energy

$$\zeta_1 = h + \frac{\gamma_0 - 1}{\gamma_0}\mathbf{H}^{-1}(x_2 - x_{2,0}) \; . \tag{1.27}$$

The remaining scalars are the composition (e.g., total water specific humidity and liquid water specific humidity in the case of the airwater mixture). If there is only one species, then $R_{\rm bg} = 1$ and we only need one additional constraint.

The thermodynamics is nondimensionalized by the reference scales p_0, T_0 and R_0, such that $\rho_0 = p_0(R_0T_0)^{-1}$ and

$$\mathbf{H} = \frac{R_0T_0}{gL_0} \tag{1.28}$$

is a nondimensional scale height. This is a new thermodynamic parameter that appears in the anelastic formulation and needs to be provided in `tlab.ini`. The default reference values are $p_0 = 10^5$ Pa and $T_0 = 298\ K$, and are set in `thermo_initilize`. A dimensional formulation can be considered by setting the parameter `nondimensional` equal to `.false.`.

The code sets the parameters `RRATIO`, `MRATIO` and `CRATIO_INV` equal to 1, and we can use the same thermodynamic routines that in the compressible formulation.

Incompressible formulation

Dynamics and thermodynamics are at most coupled by the buoyancy field, and thermodynamics is not necessary (but can still be used). With an appropriate form of the buoyancy function (see below), this formulation includes the Boussinesq limit. The evolution equations are :

$$0 = -\partial_k u_k \tag{1.29a}$$

$$\partial_t u_i = -\partial_k(u_i u_k) + \mathbf{Re}^{-1}\ \partial_k\left(\mu^*\partial_k u_i\right) - \partial_i p\ + \mathbf{Fr}^{-1}\, g_i\, b + \mathbf{Ro}^{-1}\,\epsilon_{ijk} f_k\, u_j \tag{1.29b}$$

$$\partial_t \zeta_i = -\partial_k(\zeta_i u_k) + \mathbf{Re}^{-1}\mathbf{Sc}_i^{-1}\,\partial_k\left(\mu^*\partial_k \zeta_i\right)\ + \mathbf{Da}_i\, w_i \tag{1.29c}$$

So far, only the case $\mu^* = 1$ in the equations above has been implemented.

Because of the decoupling between the momentum and the internal energy evolution equations, one of the scalar equations can correspond to the internal energy equation. Reaction or phase change processes, as well as radiation processes, can then be formulated as the appropriate source terms $\mathbf{Da}_{\rm L} w_{\rm L}$ and $\mathbf{Da}_{\rm R} w_{\rm R}$, respectively, in the corresponding scalar equation. Note that the physical meaning of the corresponding (generalized) Damköhler numbers $\mathbf{Da}_{\rm L}$ and $\mathbf{Da}_{\rm R}$ is a non-dimensional heat parameter, and not a timescale ratio – which is the correct meaning of the Damköhler numbers appearing in the evolution equations for the components in a compressible mixture. We maintain this inconsistency, instead of introducing new symbols and variables, for code simplicity.

Another consequence of the decoupling between the momentum and the internal energy evolution equations is that the compressibility terms in the code are not needed. We set `MRATIO` and `CRATIO_INV` equal to 1. The thermodynamic modules can still be used, and the formulation can also be dimensional or non-dimensional. In non-dimensional, the thermodynamic pressure is nondimensionalized by $p_0 = \rho_0 R_0 T_0$ instead of $\rho_0 U_0^2$. The parameter `GRATIO` remains in the governing equations to relate gas constant and heat capacity (thermal equation of state and caloric equation of state).

Body force

In the incompressible case, $b(\mathbf{x},t) = b^e(\zeta_i(\mathbf{x},t),\mathbf{x},t)$ and the buoyancy function $b^e(\zeta_i,\mathbf{x},t)$ needs to be provided. The buoyancy function is assumed to be normalized by a reference buoyancy (acceleration) b_0 so that $\mathbf{Fr} = U_0^2/(b_0 L_0)$. (Note that non-dimensional numbers represent the relative magnitude of physical processes and thus they are positive semi-definite; the sign or direction associated with the process, if any, in indicated in the corresponding parameters.)

For the buoyancy function (see routine `mappings/fi_buoyancy`), the common expressions are a linear or bilinear relation to the scalar fields as

$$b^e = \alpha_1\zeta_1 + \alpha_2\zeta_2 + \ldots + \alpha_{\texttt{inb\_scal\_array+1}} - b_{\rm ref}(x_2)\ , \tag{1.30}$$

where the coefficients α_i need to be provided. A quadratic form

$$b^e = -\frac{4\alpha_0}{\alpha_1^2}\zeta_1(\zeta_1-\alpha_1) - b_{\rm ref}(x_2) \tag{1.31}$$

is also available, so that the maximum buoyancy α_0 is achieved at $\zeta_1 = \alpha_1/2$.

The reference profile $b_{\rm ref}(x_2)$ is used to remove the hydrostatic balance to leading order and reduce the numerical error when calculating the pressure from the Poisson equation. (The exact form is not important because the Poisson equation provides the corresponding part of the hydrostatic balance.) To consider the background variation of density, there is also the class

$$b(\mathbf{x},t) = (\zeta_1(\mathbf{x},t) - \langle\zeta_1\rangle)/\langle\zeta_1\rangle\ , \tag{1.32}$$

where angle brackets indicate an average over the horizontal directions.

Note that the buoyancy field is *always* treated as a diagnostic variable and the average statistical data – actually from the momentum source term $\mathbf{Fr}^{-1}b$ – is calculated as an additional scalar field on top of `inb_scal_array` scalars (prognostic plus diagnostic). The only exception occurs when the buoyancy function is merely a linear relation because, in that case, the corresponding statistical information can be easily obtained from the corresponding scalar field.

Coriolis force

The current formulation corresponds to that of an Ekman layer along an Ox_1x_3 plane. The direction Ox_2 is defined along the angular velocity, so that $f_1 = f_3 = 0$. The momentum equation reads

$$\partial_t u_1 = -\partial_k(u_1u_k) - \partial_1 p \quad +\mathbf{Re}^{-1}\,\partial_k\left(\mu^*\partial_k u_1\right) + \mathbf{Fr}^{-1}\,g_1\,b + \mathbf{Ro}^{-1}\,f_2\,(u_{3,g}-u_3) \tag{1.33a}$$

$$\partial_t u_2 = -\partial_k(u_2u_k) - \partial_2 p \quad +\mathbf{Re}^{-1}\,\partial_k\left(\mu^*\partial_k u_2\right) + \mathbf{Fr}^{-1}\,g_2\,b \tag{1.33b}$$

$$\partial_t u_3 = -\partial_k(u_3u_k) - \partial_3 p \quad +\mathbf{Re}^{-1}\,\partial_k\left(\mu^*\partial_k u_3\right) + \mathbf{Fr}^{-1}\,g_3\,b + \mathbf{Ro}^{-1}\,f_2\,(u_1-u_{1,g}) \tag{1.33c}$$

The geostrophic velocity vector $(u_{1,g},\,u_{2,g},\,u_{3,g}) = (\cos\alpha,\,0,\,-\sin\alpha)$ is defined in terms of the input parameter α (rotation angle around Ox_2), to be provided.

Radiation

Formally, radiation heating or cooling can be considered in the equations above as an additional source term $\mathbf{Da}_{\rm R}\,\omega_{\rm R}\,\delta_{i\beta}$ in the right-hand side of one of the evolution equations, where $\mathbf{Da}_{\rm R}$ is the corresponding non-dimensional heat parameter and the radiation function $\omega_{\rm R}(\zeta_j)$ is normalized by the corresponding (dimensional) heat parameter Q.

Practically, the implementation is completely embedded in the routine `operators/opr_radiation` and uses data provided in the block `[Radiation]` (i.e., assumes $\mathbf{Da}_\mathrm{R} = 1$ and all coefficients enter in the definition of ω_R). One formulation is a one-dimensional bulk model, where

$$\omega_\mathrm{R} = \zeta_\gamma \left(\alpha_0 \exp\left[-\alpha_1^{-1} \int_z^{z_\infty} \zeta_\gamma(z')\,\mathrm{d}z' \right] + \alpha_2 \exp\left[-\alpha_1^{-1} \int_0^{z} \zeta_\gamma(z')\,\mathrm{d}z' \right] \right) . \tag{1.34}$$

The scalar indeces $\{\beta\,,\gamma\}$ and the parameters $\{\alpha_0\,,\alpha_1\,,\alpha_2\}$ need to be provided. Default values are $\beta = 1$, $\gamma =$`inb_scal_array` and $\alpha_2 = 0$. The scalar field ζ_γ can be an average profile, instead of instantaneous values, and non-linear mapping functions can be specified.

For clarity, this corresponds to

$$\omega_\mathrm{R} = -\partial_z F\ , \qquad F = F_{c,0} \exp\left[-\kappa \int_z^{z_\infty} \rho q_\mathrm{l}\,\mathrm{d}z' \right] + F_{c,1} \exp\left[-\kappa \int_0^{z} \rho q_\mathrm{l}\,\mathrm{d}z' \right] \tag{1.35}$$

with

$$\alpha_0 = -F_{c,0}\kappa\ , \qquad \alpha_1 = \kappa^{-1}\ , \qquad \alpha_2 = F_{c,1}\kappa\ , \qquad \zeta_\gamma = \rho q_\mathrm{l}\ , \tag{1.36}$$

where ρq_l is the liquid water content. The flux is the sum $F = F_0 + F_1$ of the solutions to the linear problems

$$\partial_z F_0 = \ \ \kappa\rho q_\mathrm{l} F_0\ , \qquad F_0|_{z=z_\infty} = F_{c,0}\ , \tag{1.37a}$$

$$\partial_z F_1 = -\kappa\rho q_\mathrm{l} F_1\ , \qquad F_1|_{z=0} = F_{c,1}\ . \tag{1.37b}$$

Transport

Transport phenomena different from molecular transport, e.g., that due to particle sedimentation, can be considered as an additional source term $\mathbf{Da}_\mathrm{T}\,\omega_\mathrm{T}\,\delta_{i\beta}$ in the right-hand side of one of the evolution equations, where $\mathbf{Da}_\mathrm{T}$ is the corresponding non-dimensional transport parameter and the transport function $\omega_\mathrm{T}^e(\zeta_j)$, to be provided, is normalized by the corresponding (dimensional) transport parameter.

For the transport function, see routine `mappings/fi_sources`.

1.3 Particle formulation

Lagrange formulation to describe the evolution of a set of particles.

To be done.

2 Boundary and Initial Conditions

2.1 Background profiles

The general form is given as a function of the coordinate x_2 according to

$$f(x_2) = f_{\mathsf{ref}} + \Delta f \, g(\xi) \, , \qquad \xi = \frac{x_2 - x_{2,\mathsf{ref}}}{\delta} \, . \tag{2.1}$$

where the set of parameters $\{f_{\mathsf{ref}},\, \Delta f,\, x_{2,\mathsf{ref}},\, \delta\}$ need to be provided together with the normalized profile $g(\xi)$.

Possible forms are given in table 2.1 and figure 2.1. There are shear-like and jet-like profiles. In the former case, the normalized profiles vary between $-1/2$ and $+1/2$, so that equation (2.1) represents a variation of order Δf around the reference value f_{ref} across a distance of order δ centered around the position $x_{2,\mathsf{ref}}$. The sign of δ can be used to impose the symmetric form, if needed. The gradient thickness is defined by

$$\delta_g = \frac{\Delta f}{|df/dx_2|_{\mathsf{max}}} = \delta \, \frac{1}{|dg/d\xi|_{\mathsf{max}}} \, . \tag{2.2}$$

In the case in which the profile is used to define the mean velocity, this thickness is known as vorticity thickness. It is very often more convenient to define the problem in terms of the gradient thickness instead of the thickness parameter δ. The reason to keep it in terms of δ in the code is simply for compatibility with the previous versions.

The second group of profiles deliver a jet-like shape. In that case, Δf provides the maximum difference with respect to the reference level f_{ref}. The integral thickness is defined by

$$\delta_i = \frac{1}{\Delta f} \int (f - f_{\mathsf{ref}}) \, \mathrm{d}x_2 = \delta \int g(\xi) \mathrm{d}\, \xi \, . \tag{2.3}$$

According the implementation currently used, typical values are $\delta_i \sim 2 - 3\delta$ (table 2.1).

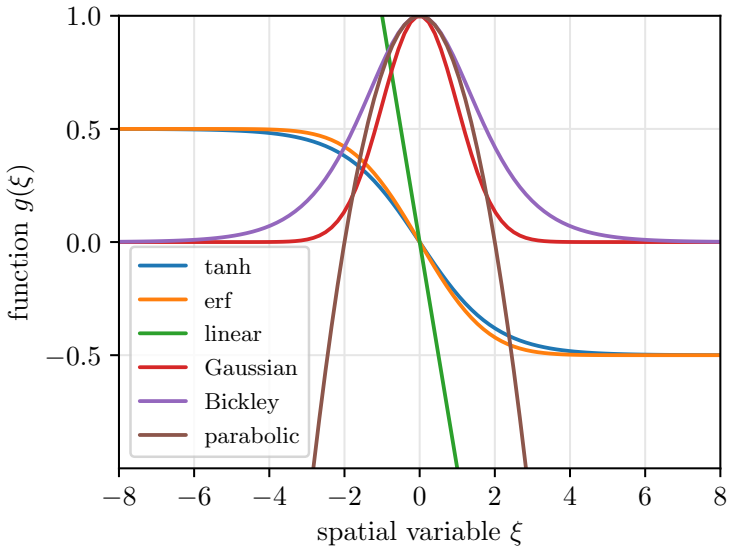

Figure 2.1: Different normalized profiles used in equation (2.1). The black profiles provide shear-like backgrounds, the green lines provide jet-like backgrounds. These background profiles are used consistently for the boundary and initial conditions and aims at the study of different canonical flows, free and wall-bounded, shear- and buoyancy-driven.

Type	$\mathbf{g}(\boldsymbol{\xi})$	$\boldsymbol{\delta_g/\delta}$	$\boldsymbol{\delta_i/\delta}$	Notes
Hyperbolic tangent	$(1/2)\tanh(-\xi/2)$	4		Used in shear layer because it is the reference profile commonly used in linear stability analysis. The parameter δ is equal to the momentum thickness. Used because of available linear stability analysis.
Error function	$(1/2)\mathrm{erf}(-\xi/2)$	$2\sqrt{\pi}$		Used in diffusion dominated problems because it is a solution of the diffusion equation.
Linear	$-\xi$	1		Varying Δf along δ.
Ekman	$1-\exp(-\xi)\cos(\xi)$ $-\exp(-\xi)\sin(\xi)$	1		Velocity component along geostrophic wind. Normal component.
Gaussian	$\exp(-\xi^2/2)$	1.65	$\sqrt{2\pi}$	Gaussian bell with standard deviation equal to δ.
Bickley	$1/\cosh^2(\xi/2)$		4	Bell shape used in the linear stability of jets. Used because of available linear stability analysis.
Parabolic	$(1+\xi/2)(1-\xi/2)$		$8/3$	Parabola crossing the reference value at $x_{2,ref}\pm 2\delta$. Used for Poiseuille and channel flows. The thickness δ_i is calculated using only the positive part of the profile in the integral.

Table 2.1: Different normalized profiles used in equation (2.1). The third column contains the gradient thickness δ_g, defined by equation (2.2), written explicitly as a function of the thickness parameter δ. The fourth column contains the integral thickness δ_i, defined by equation (2.3)

The are additionally two linear contributions below and above the reference position $x_{2,\mathrm{ref}}$:

$$f(x_2) = f(x_2) + \alpha_l(x_2 - x_{2,\mathrm{ref}})\mathcal{H}(x_{2,\mathrm{ref}} - x_2) + \alpha_u(x_2 - x_{2,\mathrm{ref}})\mathcal{H}(x_2 - x_{2,\mathrm{ref}}) \ , \qquad (2.4)$$

where $\mathcal{H}$ is the Heaviside function. The slopes α_l and α_u need to be provided. The default values of the slopes are zero, so that no linear variations are added unless explicitly indicated.

2.2 Initial conditions

See files in `tools/initialize/{flow,scal}`. The information is in the block `[IniFields]` of the input file. White noise is often added to mean profiles to accelerate the transition to turbulence. There are reasons, however, to construct initial conditions that are more elaborated than white noise, both in the flow fields and in the scalar fields:

- To be able to compare with analytical solutions and thereby validate the code and algorithms.
- To ascertain the duration of the initial transient before the flow enters into the fully developed turbulent regime. We can do that by varying the initial conditions in a controlled way.
- To control certain aspects of that transient by using results from stability analysis and exciting or not certain modes. White noise excites all of them equally, and the higher frequency content is dissipated much faster, which might render the energy amount that we use in the initialization misleading.

The first step is to defined a mean background profile for velocity, thermodynamic and scalar fields according to the previous section. For instance, a hyperbolic tangent profile for the mean streamwise velocity, $\bar{u}_1(x_2)$, while all other mean velocity components are set to zero. The upper stream has a velocity $u_{1,ref} - \Delta u/2$ and the lower stream has a velocity $u_{1,ref} + \Delta u/2$ (see figure 2.1). The

mean density (or the mean temperature) can be similarly initialized, and a mean pressure is set to a uniform value p_0. The mean scalar can be similarly initialized. For this step, we use the information in the block `[Flow]` and the block `[Scalar]`.

These mean values define mean fields. In addition, we can add perturbation fields to these mean fields. These perturbation fields can be discrete or broadband.

2.2.1 Discrete Perturbations

For the general case of a scalar field s, we consider the linear superposition of m modes of the form

$$s_m = A_m f(y) \cos(w_{xm} x + \phi_{xm}) \cos(w_{zm} z + \phi_{zm}) \ , \tag{2.5}$$

where $f(y)$ is one of the shape functions discussed before.

For the particular case of the velocity field, we consider the linear superposition of m modes of the form

$$u_m = A_m g(y)\, c_{xm} \sin(\omega_{xm} x + \phi_{xm}) \cos(\omega_{zm} z + \phi_{zm}) \ , \tag{2.6}$$
$$v_m = A_m f(y) \quad \cos(\omega_{xm} x + \phi_{xm}) \cos(\omega_{zm} z + \phi_{zm}) \ , \tag{2.7}$$
$$w_m = A_m g(y)\, c_{zm} \cos(\omega_{xm} x + \phi_{xm}) \sin(\omega_{zm} z + \phi_{zm}) \ , \tag{2.8}$$

where

$$\begin{array}{lll} c_{xm} = 1/\omega_{xm} & c_{zm} = 0 & \text{if } \omega_{zm} = 0 \ , \\ c_{xm} = 0 & c_{zm} = 1/\omega_{zm} & \text{if } \omega_{xm} = 0 \ , \\ c_{xm} = 1/(2\omega_{xm}) & c_{zm} = 1/(2\omega_{zm}) & \text{otherwise} \ , \end{array}$$

and $g(y) = -f'(y)$. Each mode satisfies the solenoidal constraint.

For this step, we use the information in the block `[Discrete]`.

2.2.2 Random Fields

The first step to make a broadband perturbation is to generate a random field. We generate a random field on which an isotropic turbulence spectrum is imposed according to the spectral functions in Table 2.2. These three-dimensional fields are saved to disk in the same format as the usual flow and scalar variables. We can use them as initial conditions for the simulations. However, we typically use them to create perturbation fields that are added to mean fields, and this sum of mean plus perturbation is what makes the flow and scalar fields that are used as initial condition.

Type	$E(f)$	Notes
uniform	1	
quadratic	$(f/f_0)^2 \exp[-2(f/f_0)]$	
quartic	$(f/f_0)^4 \exp[-2(f/f_0)^2]$	
Gaussian	$\exp[-(1/2)(f/f_0 - 1)^2/(\sigma/f_0)^2]$	

Table 2.2: Different spectral functions that can be used to create a random field. The variable f is the spatial frequency and the parameter f_0 is the peak spatial frequency.

See files in `tools/initialize/rand` and the information in the block `[Broadband]` in the input file.

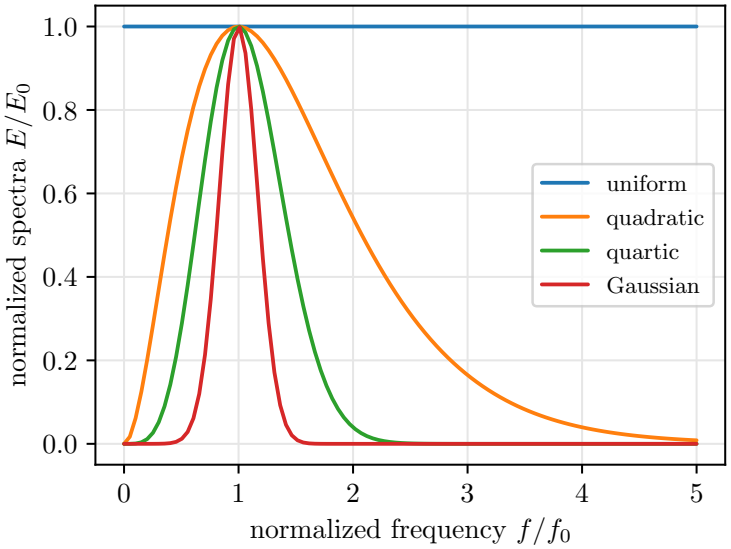

Figure 2.2: Different power spectral densities available as initial conditions, Table (2.2). Normalized such that all of them have equal integral. The Gaussian profile is plotted for the case $\sigma/f_0 = 1/6$ typically used in the simulations.

2.2.3 Broadband Perturbations

To construct perturbation fields from these random fields, we first multiply the random field with a shape function $f(y)$ to control the spatial location of this random fluctuation. This shape function follows the same structure as the profiles that we explained in the first section about background profiles. It is the variable `IniK` in `tools/initialize/flow` and `IniS` in `tools/initialize/scal`. For instance, we can multiply them by a Gaussian function of a given thickness to put the perturbation in certain layer inside the computational domain.

There are 3 options to generate a velocity field from these random fields. We can use the random field directly to create a perturbation in the velocity field $f\mathbf{a}$. Instead, we can consider the random fields as a vector potential $f\mathbf{a}$ of a velocity field $\mathbf{u} = \nabla \times (f\mathbf{a})$. The third option is to consider a Helmholtz decomposition of $\mathbf{u}$ into a divergence free part and an irrotational part, set the irrotational part equal to zero, and consider $\nabla \times (f\mathbf{a})$ as the divergence free part.

In case of the velocity field, the last step typically consists in imposing the solenoidal constraint. For the incompressible and anelastic formulations, we need it. For the compressible formulation, such quasi-incompressible fluctuations minimize compressibility transients Erlebacher et al. [1990].

In the compressible formulation, the pressure fluctuations are obtained from the Poisson equation that would correspond to incompressible flows.

See files in `tools/initialize/{flow,scal}` and the information in the block `[IniFields]` in the input file.

2.3 Boundary conditions

2.3.1 Compressible formulation

For non-periodic directions, the treatment of the boundary conditions in the periodic formulation is done in characteristic form [Thompson, 1987, 1990, Lodato et al., 2008].

2.3.2 Incompressible formulation

To be developed.

2.3.3 Buffer zone

Buffer or sponge zones can be considered at the beginning and at the end of the directions Ox and Oy. These are simply defined by specifying the number of points over which they extend. We can consider a filter or a relaxation form.

Regarding the relaxation form, we simply add

$$\left(\frac{\delta \mathbf{q}}{\delta t}\right)_b = -\tau_q^{-1}(\mathbf{q} - \mathbf{q}_0) \ , \qquad \left(\frac{\delta \mathbf{s}}{\delta t}\right)_b = -\tau_s^{-1}(\mathbf{s} - \mathbf{s}_0) \tag{2.9}$$

to the right-hand sides of the transport equations [Hu, 1996]. The relaxation times $\tau(\mathbf{x})$ are defined in terms of a power function as

$$\tau_q^{-1} = \alpha_1 (n - n_0)^{\alpha_2} \ , \qquad \tau_s^{-1} = \beta_1 (n - n_0)^{\beta_2} \tag{2.10}$$

where n is the coordinate normal to the corresponding boundary and n_0 the coordinate where the buffer region begins. The coefficients α_i and β_i need to be provided, the exponents being preferably larger or equal than 2 so than the pressure equation has a continuous right-hand side. The reference fields $\mathbf{q}_0(\mathbf{x})$ and $\mathbf{s}_0(\mathbf{x})$ also need to be provided and it can be any general field. Normally, a reference field is created at some moment in the simulation (generally the initial time) as an average of the corresponding region in the flow and scalar fields.

3 Code

The root directory contains the sources for the common libraries, and the directory `tools` contains the sources for the specific binaries: the main code in `tools/dns`, and the preprocessing and postprocessing tools.

Files `README` and `TODO` contain additional information. To compile, read `INSTALL`.

Directory `examples` contains a few examples to get acquainted with using the code.

3.1 Executables

Simulation	
`dns.x`	runs the simulation. Reads input from the file `tlab.ini` and needs initial flow and scalar fields, and a grid file. Examples are in `examples`. Standard output is written to `tlab.log` and `dns.out`, errors to `tlab.err` and warnings to `tlab.war`. Sources in `tools/dns`.

Preprocessing	
`inigrid.x`	Generates the grid from the parameters of the tlab.ini file, section [IniGridOx], [IniGridOy] and [IniGridOz]. Sources in `tools/initialize/grid`.
`inirand.x`	Generates the scal.rand and flow.rand files that contain pseudo-random, isotropic fields that are used to generate flow and scalar initial fields. The parameters are described in `tlab.ini`, section [Broadband]. Sources in `tools/initialize/rand`
`iniscal.x`	Generates the scal.ics file from the parameters of the tlab.ini file, section [IniFields]. Sources in `tools/initialize/scal.`
`iniflow.x`	Generates the flow.ics file from the parameters of the tlab.ini file, section [IniFields]. Sources in `tools/initialize/flow`.
`inipart.x`	Generates the part.ics file from the parameters of the tlab.ini file, section [Particles]. Files *.id are equivalent to grid file; they contain the number of processors, the number of particles per processor, and the ordered tags of the particles. Sources in `tools/initialize/part`.

Postprocessing	
`averages.x`	Calculates main average profiles and conditional averages (outer intermittency). Sources in `tools/statistics/averages`.
`spectra.x`	Calculates 1D, 2D and 3D spectra and co-spectra of main variables. Sources in `tools/statistics/spectra`.
`pdfs.x`	Calculates PDFs, joints PDFs and conditional PDFs. Sources in `tools/statistics/pdfs`.
`visuals.x`	Calculates different fields and exports data for visualization. Sources in `tools/plot/visuals`.

3.2 Scripts

Sources in `scripts/python`.

Python	
`xdmf.py`	Create XDMF file with fields data for visualization tools, such as ParaView. Argument is the list of binary files to be included in the XDMF file. Edit the file to set grid size.
`xdmfplanes.py`	Create XDMF file with planes data for visualization tools, such as ParaView. Argument is the list of binary files to be included in the XDMF file. Edit the file to set grid size and plane parameters.
`stats2nc.py`	Construct NetCDF files from a list of ASCII statistics files.

3.3 Input file tlab.ini

The following tables describe the different blocks appearing in the input file `tlab.ini`. The first column contains the tag. The second column contains the possible values, the first one being the default one and the word *value* indicating that a numerical value needs to be provided. The third column describes the field. This data is read in the file `*_READ_GLOBAL` and in the files `*_READ_LOCAL` of each of the tools.

Data is case insensitive.

[Version]		
`Major`	*value*	Error generated if different from value in `DNS_READ_GLOBAL`.
`Minor`	*value*	Warning generated if different from value in `DNS_READ_GLOBAL`.

[Main]		
`Type`	`temporal, spatial`	Temporally evolving or spatially evolving simulation.
`CalculateFlow`	`yes, no`	Execute code segments affecting flow variables.
`CalculateScalar`	`yes, no`	Execute code segments affecting scalar variables.
`Equations`	`internal, total, incompressible`	Define system of equations to be solved.
`Mixture`	`None, AirVapor, AirWater, AirWaterLinear`	Defines the mixture to be used for the thermodynamics.
`TermAdvection`	`divergence, convective, skewsymmetric`	Formulation of advection terms.
`TermViscous`	`divergence, explicit`	Formulation of viscous terms.
`TermDiffusion`	`divergence, explicit`	Formulation of diffusion terms.
`TermBodyForce`	`None, Explicit, Homogeneous, Linear, Bilinear, Quadratic`	Formulation of body force terms (see `mappings/fi_buoyancy`).
`TermCoriolis`	`None, Explicit, Normalized`	Formulation of Coriolis terms.
`TermRadiation`	`None, Bulk1dGlobal, Bulk1dLocal`	Formulation of radiation terms (see `operators/opr_radiation`).
`SpaceOrder`	`CompactJacobian4, CompactJacobian6, CompactDirect6`	Finite difference method used for spatial derivatives.
`TimeOrder`	`RungeKuttaExplicit3, RungeKuttaExplicit4, RungeKuttaDiffusion3`	Runge-Kutta method used for time advancement.
`TimeCFL`	*value*	Courant number for the advection part.
`TimeStep`	*value*	If `TimeCFL` is negative, constant time step in time marching scheme.

[Iteration]		
`Start`	*value*	Initial iteration. Files `flow.*` and `scal.*` are read from disk.
`End`	*value*	Final iteration at which to stop.
`Restart`	*value*	Iteration-step frequency to write the restart files to disk.

`Statistics`	*value*	Iteration-step frequency to calculate statistics.
`IteraLog`	*value*	Iteration-step frequency to write the log-file `dns.out`.
`SavePlanes`	*value*	Iteration-step frequency to write plane data to disk.
Spatially evolving simulations		
`RunAvera`	`no, yes`	Save running averages to disk.
`RunLines`	`no, yes`	Save line information to disk.
`RunPlane`	`no, yes`	Save plane information to disk.
`StatSave`	*value*	Iteration-step frequency to accumulate statistics.
`StatStep`	*value*	Iteration-step frequency to save statistics to disk.

[Parameters]		
`Reynolds`	*value*	Reynolds number $\mathbf{Re}$ (see section 1).
`Prandtl`	*value*	Prandtl number $\mathbf{Pr}$.
`Froude`	*value*	Froude number $\mathbf{Fr}$.
`Rossby`	*value*	Rossby number $\mathbf{Ro}$.
`Mach`	*value*	Mach number $\mathbf{Ma}$.
`Gama`	*value*	Ratio of specific heats γ.
`Schmidt`	*value1, value2, ...*	List of Schmidt numbers $\mathbf{Sc}_i$, as many as scalar fields.
`Damkohler`	*value1, value2, ...*	List of Damkohler numbers $\mathbf{Da}_i$.
`Settling`	*value1, value2, ...*	List of settling numbers $\mathbf{Sv}_i$.

[Control]		
`MinPressure`	*value*	Lower bound for the pressure interval.
`MaxPressure`	*value*	Upper bound for the pressure interval.
`MinDensity`	*value*	Lower bound for the density interval.
`MaxDensity`	*value*	Upper bound for the density interval.
`FlowLimit`	`yes, no`	Force the thermodynamic fields within the prescribed interval.
`MinScalar`	*value*	Lower bound for the scalar interval.
`MaxScalar`	*value*	Upper bound for the scalar interval.
`ScalLimit`	`yes, no`	Force the scalar fields within the prescribed interval.

[Grid]		
`Imax`	*value*	Number of points along Ox (first array index).
`Jmax`	*value*	Number of points along Oy (second array index).
`Kmax`	*value*	Number of points along Oz (third array index). Set to 1 in 2D.
`XUniform`	`yes, no`	If yes, no Jacobian information is needed along Ox.
`YUniform`	`yes, no`	If yes, no Jacobian information is needed along Oy.
`ZUniform`	`yes, no`	If yes, no Jacobian information is needed along Oz.
`XPeriodic`	`yes, no`	Periodicity along Ox.
`YPeriodic`	`yes, no`	Periodicity along Oy.
`ZPeriodic`	`yes, no`	Periodicity along Oz.
MPI parallel mode		
`Imax(*)`	*value*	Number of points per processor (MPI task) along Ox.
`Jmax(*)`	*value*	Number of points per processor (MPI task) along Oy. Set equal to the total size in the current 2D decomposition.
`Kmax(*)`	*value*	Number of points per processor (MPI task) along Oz.

[BoundaryConditions]		
`VelocityImin`	`none, noslip, freeslip`	Velocity boundary condition at x_{min}.
`VelocityImax`	`none, noslip, freeslip`	Velocity boundary condition at x_{max}.
`Scalar#Imin`	`none, dirichlet, neumman`	Boundary condition of scalar number # at x_{min}.
`Scalar#Imax`	`none, dirichlet, neumman`	Boundary condition of scalar number # at x_{max}.

Similarly in the other directions Oy and Oz. Compressible case to be done.

[BufferZone]		
`Type`	`none, relaxation, filter, both`	Type of buffer layer (sponge layer) to use.

`LoadBuffer`	`no, yes`	If `no`, construct buffer fields and save them to disk. If `yes`, read them from disk: `{flow,scal}.bcs.{imin,imax,jmin,jmax}.#`.
`PointsImin`	*value*	Number of points in Ox at x_{min}.
`PointsImax`	*value*	Number of points in Ox at x_{max}.
`PointsUJmin`	*value*	Number of points in Oy at y_{min} for the velocity fields.
`PointsUJmax`	*value*	Number of points in Oy at y_{max} for the velocity fields.
`PointsEJmin`	*value*	Number of points in Oy at y_{min} for the thermodynamic fields.
`PointsEJmax`	*value*	Number of points in Oy at y_{max} for the thermodynamic fields.
`PointsSJmin`	*value*	Number of points in Oy at y_{min} for the scalar fields.
`PointsSJmax`	*value*	Number of points in Oy at y_{max} for the scalar fields.
`ParametersU`	*value1, value2, ...*	Parameters that define strength and exponent of the relaxation term in the flow fields, section 2.3.3.
`ParametersS`	*value1, value2, ...*	Parameters that define strength and exponent of the relaxation term in the scalar fields, section 2.3.3.

[Flow]		
`Profile*`	`None, Tanh, Erf, Ekman,...`	Functional form of the mean profile g, equation (2.1).
`Mean*`	*value*	Reference mean value.
`YMean*`	*value*	y-coordinate of profile reference point.
`YMeanRelative*`	*value*	y-coordinate of profile reference point, relative to the total scale.
`Thick*`	*value*	Reference profile thickness.
`Delta*`	*value*	Reference profile difference.
`LowerSlope*`	*value*	Slope of the linear variation below the reference point.
`UpperSlope*`	*value*	Slope of the linear variation above the reference point.
`Diam*`	*value*	Reference profile diameter (jet mode).
** is VelocityX, VelocityY, VelocityZ, density, pressure or temperature*		

[Scalar]		
`idem`	`idem`	idem
Same as `[Flow]` *but * is* `Scalar#`.		

[Thermodynamics]		
`Pressure`	*value*	Reference mean pressure.
`Parameters`	*value1, value2, ...*	Parameters that define the buoyancy function $b^e(s_i)$.

[BodyForce]		
`Vector`	*value1, value2, value3*	Components of buoyancy unitary vector (g_1, g_2, g_3), section 1.
`Parameters`	*value1, value2, ...*	Parameters that define the buoyancy function $b^e(s_i)$.

[Rotation]		
`Vector`	*value1, value2, value3*	Components of angular velocity vector (f_1, f_2, f_3), section 1.
`Parameters`	*value1, value2, ...*	Parameters that define the Coriolis force term.

[Radiation]		
`Scalar`	*value*	Index of scalar field on which radiation heating or cooling acts.
`Parameters`	*value1, value2, ...*	Parameters that define the radiation function $r^e(s_i)$.

[IniFields]		
`Velocity`	`None, VelocityDiscrete, VelocityBroadband, VorticityBroadband, PotentialBroadband`	Type of initial velocity field.
`Temperature`	`None, PlaneBroadband, PlaneDiscrete`	Type of initial temperature field.

`Scalar`	`None, LayerDiscrete, LayerBroadband, PlaneDiscrete, PlaneBroadband, DeltaDiscrete, DeltaBroadband, FluxDiscrete, FluxBroadband`	Type of initial scalar field.
`ForceDilatatio`	`yes, no`	Force the velocity field to satisfy the solenoidal constraint.
`ProfileIni{K,S}`	`Gaussian, Parabolic, ...`	Shape profile of the fluctuation. Use `K` or `S` to differentiate between flow and scalar fields.
`ThickIni{K,S}`	*value*[*1 , value2, ...*]	Thickness of fluctuation shape profile. It defaults to the corresponding values in `[Flow]` and `[Scalar]`. For scalars, provide as many values as scalars.
`YMeanIni{K,S}`	*value*[*1 , value2, ...*]	Coordinate along Oy of the reference point of the fluctuation shape profile, relative to the total scale.
`NormalizeK`	*value*	Maximum value of the profile of the turbulent kinetic energy.
`NormalizeP`	*value*	Maximum value of the profile of the pressure root-mean-square.
`NormalizeS`	*value1, value2, ...*	Maximum value of the profile of the scalar root-mean-square.

[Filter]		
`Type`	`None, TopHat, Helmholtz, Compact, Adm, Explicit6, Explicit4, SpectralBand, SpectralErf`	Type of filter used in simulation and postprocessing.
`Parameters`	*value1, value2, ...*	Values defining the filter, e.g., α in compact, or Δ in tophat.
`ActiveY`	`yes,no`	Active or not in direction Oy.
`BcsJmin`	`Free, Solid`	Type of boundary condition for tophat filter at y_{min}. Other filters might have other options, section 6.1.3.
`BcsJmax`	`Free, Solid`	Type of boundary condition for tophat filter at y_{max}. Other filters might have other options, section 6.1.3.
Similarly in the other directions Ox and Oz.		

[Broadband]		
`Distribution`	`none, uniform, gaussian`	Type of PDF. If none, randomize phases in frequency space.
`Covariance`	*value1, value2, ...*	Covariance matrix.
`Spectrum`	`quartic, gaussian, ...`	Functional form of the power spectral density, figure (2.2).
`f0`	*value1, value2, ...*	Parameters that define the functional form.
`Seed`	*value*	Seed for the random generator.

[Discrete]		
`Amplitude`	*value1, value2, ...*	Amplitude of the modes.
`ModeX`	*value1, value2, ...*	Mode numbers as multiples of fundamental frequency along Ox.
`ModeZ`	*value1, value2, ...*	Mode numbers as multiples of fundamental frequency along Oz.
`PhaseX`	*value1, value2, ...*	Corresponding phases.
`PhaseZ`	*value1, value2, ...*	Corresponding phases.
`Type`	`Varicose, Sinuous, Gaussian`	Modulation of the sinusoidal perturbation.
`Parameters`	*value*	Parameters that define the modulation.

[Inflow]		
`Type`	`None, BroadbandSequential, BroadbandPeriodic, Discrete`	Type of inflow forcing to use in a spatially evolving simulation.
`Adapt`	*value*	Interval in global time units for starting the inflow forcing.

[PostProcessing]		
`Files`	*value1, value2, ...*	Iterations to be postprocessed.
`Subdomain`	$i_1, i_2, j_1, j_2, k_1, k_2$	Grid block to be postprocessed.

`Partition`	α_1[, α_2], β_1, ..., β_{n-1}	Type of partition defined by values $\{\alpha_1[, \alpha_2]\}$. The first parameter defines the conditioning field: 1. external field, 2. scalar field, 3. enstrophy, 4. magnitude of scalar gradient. The second parameter chooses between a relative or an absolute threshold values. Set of thresholds $\{\beta_1, ...,\beta_{n-1}\}$ to define the partition of the conditioning field into n zones.
`ParamAverages`	α_1[, α_2, α_3, α_4]	Main option α_1 (see `tools/statistics/averages.f90`); block size α_2; gate level α_3; maximum order of the moments α_4.
`ParamPdfs`	α_1[, α_2, α_3, α_4]	Main option α_1 (see `tools/statistics/pdfs.f90`); block size α_2; gate level α_3; number of bins α_4.
`ParamSpectra`	α_1[, α_2, α_3, α_4]	Main option α_1 (see `tools/statistics/spectra.f90`); block size α_2; save full spectra α_3; average over iterations α_4.
`ParamVisuals`	α_1, α_2, ...	List of fields to be calculated (see `tools/plot/visuals.f90`).

[Particles]		
`Type`	`Tracer, ...`	Type of particles equations to solve.
`Number`	*value*	Number of particles.
`ProfileIniP`	`TanhSymmetric, Scalar, ...`	Type of initialization.
`YMeanIniP`	*value*	Similar to [Flow] and [Scalar].
`ThickIniP`	*value*	Similar to [Flow] and [Scalar].
`TrajNumber`	*value*	Number of particle trajectories to save.

4 Grid

The equations are solved using Cartesian co-ordinates and the grid is structured. The grid is constructed by building up the three directions separately (in a 2D case, Oz has simply one node). Each direction is broken into segments, and each of those segments is built with specified generation algorithms. The first node in each direction is set at zero.

The executable is `inigrid.x` and the data in `tlab.ini` is specified in the blocks [IniGridOx], [IniGridOy] and [IniGridOz]. Once created, basic information about the grid is saved into the file `grid.sts`. Grid transformations can be done with `transgrid.x`; the corresponding sources are in the `tools/transform` sub-directory. They allow to print out an ASCII file with the grid positions, add an offset, drop or introduce planes and make a scaling.

[IniGridOx]		
`Segments`	*value*	Number of segments in that direction.
`Periodic`	`yes, no`	If yes, a uniform mesh is done with a number of points equal to the sum of the points of all segments, and over a length equal to the sum of the length of all segments, regardless the input options for each segment. The last plane in that direction is dropped, so that the last point does not match the scale (the latter is a bit larger).
`Mirrored`	`yes, no`	If yes, this direction is created with the corresponding options and then it is mirrored with respect to the origin. The final scale is the double of that set in the input file `tlab.ini` and the first node is moved to zero. In the case of mirroring the number of points is always even.
`Scales_#`	*value*	Physical end of the segment number #.
`Points_#`	*value*	Number of points in the segment number #. This number includes the first and the last points of the present segment, which are common with the adjacent ones. (E.g., if one direction has three segments with 11, 16 and 6 points each, the total number of points in that direction will be 31, 30 steps.)
`Opts_#`	`uniform, tanh, exp...`	Type of grid generation algorithm (see text below).
`Vals_#`	*value1, value2, ...*	List of constants for the different generation algorithms.

4.1 Generation algorithms

4.1.1 Explicit mappings

The reference is a grid with a uniform spacing:

$$\{s_j = h_0(j-1) : j = 1, \ldots, n\} \, . \tag{4.1}$$

Then, one considers different mappings from this reference grid to the final grid $\{x_j = x(s_j)\}$.

For `opts=tanh`, the mapping $x = x(s)$ is obtained by solving the equation

$$\frac{\mathrm{d}x}{\mathrm{d}s} = 1 + \frac{h_1/h_0 - 1}{2} \left[1 + \tanh\left(\frac{s - s_{t1}}{2\delta_1} \right) \right] \, , \tag{4.2}$$

that is, the grid step $\Delta x \equiv \mathrm{d}x/\mathrm{d}j$ varies according to a hyperbolic tangent function between the uniform values h_0 and h_1, the transition occurring at $s = s_{t1}$ and over a length δ_1. The grid step h_0 corresponds to that of the initial uniform grid. The parameters s_{t1}, h_1/h_0 and δ_1 are provided in the corresponding `vals` record of the `tlab.ini` file.

The mapping can be written explicitly as

$$x = s + (h_1/h_0 - 1)\delta_1 \left\{\ln(1 + \exp[(s - s_{t1})/2]) + C\right\} \tag{4.3}$$

where C is the integration constant, defined such that $x = 0$ for $s = 0$. The final extension obeys the relationship

$$x_{\max} = s_{\max} + (h_1/h_0 - 1)\delta_1 \ln\left[\frac{\exp(s_{\max}/\delta_1) + 1}{1 + \exp(-s_{t1}/\delta_1)}\right]. \tag{4.4}$$

where $s_{\max} = (n-1)h_0$.

Two transitions are possible if the `vals` record contains 6 elements instead of 3:

$$\frac{dx}{ds} = 1 + \frac{h_1/h_0 - 1}{2}\left[1 + \tanh\left(\frac{s - s_{t1}}{2\delta_1}\right)\right] + \frac{h_2/h_0 - 1}{2}\left[1 + \tanh\left(\frac{s - s_{t2}}{2\delta_2}\right)\right] . \tag{4.5}$$

For `opts=exp`, the mapping $x = x(s)$ is obtained by solving the equation

$$\frac{\mathrm{d}^2x}{\mathrm{d}s^2} - (f/h_0)\frac{\mathrm{d}x}{\mathrm{d}s} = 0 , \tag{4.6}$$

where the stretching factor is imposed according to a hyperbolic tangent function

$$f = \frac{\Delta f_1}{2}\left[1 + \tanh\left(\frac{s - s_{t1}}{2\delta_1}\right)\right] . \tag{4.7}$$

The stretching factor is defined as $f \equiv \mathrm{d}/\mathrm{d}x(\Delta x)$, so that $f + 1$ an approximation to the ratio $(\Delta x)_{j+1}/(\Delta x)_j$. The non-dimensional parameter f then varies from 0 to Δf_1; values smaller that 0.1 are recommended (less that 10% stretching). The parameters s_{t1}, Δf_1 and δ_1 are provided in the corresponding `vals` record of the `tlab.ini` file.

A first integral of this problem leads to

$$\frac{\mathrm{d}x}{\mathrm{d}s} = \left[1 + \exp\left(\frac{s - s_{t1}}{\delta_1}\right)\right]^{\delta_1(\Delta f_1/h_0)} . \tag{4.8}$$

This equation for $x(s)$ needs to be solved numerically, but already shows that this mapping leads to an exponential growth of the grid spacing Δx and the grid $x(s)$. Note that δ_1 admits negative values.

Two transitions are possible if the `vals` record contains 6 elements instead of 3:

$$f = \frac{\Delta f_1}{2}\left[1 + \tanh\left(\frac{s - s_{t1}}{2\delta_1}\right)\right] + \frac{\Delta f_2}{2}\left[1 + \tanh\left(\frac{s - s_{t2}}{2\delta_2}\right)\right] , \tag{4.9}$$

which implies

$$\frac{\mathrm{d}x}{\mathrm{d}s} = \left[1 + \exp\left(\frac{s - s_{t1}}{\delta_1}\right)\right]^{\delta_1(\Delta f_1/h_0)}\left[1 + \exp\left(\frac{s - s_{t2}}{\delta_2}\right)\right]^{\delta_2(\Delta f_2/h_0)} . \tag{4.10}$$

4.1.2 Geometric progression

These algorithms correspond to `opts=1-4`. (It has not been used in a long time.)

For each segment i the geometric ratio r_i is given (input variable $val1$). The rest of the constrains are the number of points in each segment, n_i and the total length in that direction, L. The first step of the first segment is initialized to 1 (this number does not matter because of the final scaling), and then the first segment is generated. The last step size is taken as the first one for the second segment, and the sequence continue until the last segment. A final rescaling adjusts the physical length of the grid in that direction.

The length of each segment in the grid is saved into `grid.sts`. For calculating them, if we denote the length of each segment by l_i, we have

$$l_i = h_i^1(1 + r_i + r_i^2 + \ldots + r_i^{n_i-2}) = h_i^1 \frac{1 - r_i^{n_i-1}}{1 - r_i} = h_i^1 C_i \ . \tag{4.11}$$

The first step h_i^1 of each segment is related to the previous one by

$$h_{i+1}^1 = h_i^1 r_i^{n_i-1} \tag{4.12}$$

and the equation to close the problem is

$$L = \sum_{seg} l_i = h_1^1 \left(C_1 + r_1^{n_1-1} C_2 + r_1^{n_1-1} r_2^{n_2-1} C_3 + \ldots \right) \ . \tag{4.13}$$

5 Post-Processing Tools

5.1 Averages

See file `dns/tools/statistics/averages`.

Allows for conditional analysis.

5.2 Probability density functions

See file `dns/tools/statistics/pdfs`.

Allows for conditional analysis.

5.3 Conditional analysis

To be developed.

5.4 Spatially coarse grained data at full time resolution

Data can be saved at full temporal resolution with a user-defined spatial coarse graining. In that mode, on top of the standard output, a single file is generated for each horizontal data point per restart. This allows to maintain full flexibility and at the same time to avoid parallel I/O at this stage. For small strides (see table), this approach may produce a large number of files which can cause problems if the data is not converted to netCDF immediately after a simulation is run. The files can be merged into a netCDF file with the script `scripts/python/tower2nc.py`.

[SaveTowers]		
`Stride`	*value1, value2, value3*	Strides along the directions Ox, Oy and Oz.

For example, `Stride=0,0,0` would not save any data; `Stride=1,1,1` would save all points, but it more efficient to use `Iteration.Restart=1`; `Stride=16, 1,16` would save vertical profiles every 16x16 horizontal point. A Stride of 0 has no effect whatsoever (no output will be generated). *Note: vertical corse graining is not tested, but should work.*

File name

The file name contains crucial information which is not stored elsewhere. Hence, files should not be renamed. Output files are named by the following scheme

tower	.	iloc	×	kloc	.	t_{start}	-	t_{end}	.	ivar
tower	.	000015	×	000015	.	000001	-	000010	.	1

Only one scalar is supported, and the variables (ivar) are

Inner direction of write $\rightarrow$

it_{start}	$t(it_{\text{start}})$	$v_{ivar}(y_1)$	$v_{ivar}(y_2)$	$\dots$	$v_{ivar}(ny)$
$it_{\text{start}}+1$	$t(it_{\text{start}}+1)$	$v_{ivar}(y_1)$	$v_{ivar}(y_2)$	$\dots$	$v_{ivar}(ny)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$it_{\text{end}}-1$	$t(it_{\text{end}}-1)$	$v_{ivar}(y_1)$	$v_{ivar}(y_2)$	$\dots$	$v_{ivar}(ny)$
it_{end}	$t(it_{\text{end}})$	$v_{ivar}(y_1)$	$v_{ivar}(y_2)$	$\dots$	$v_{ivar}(ny)$

Figure 5.1: Internal organization of tower files.

1–3 velocities (u, v, w)
4 pressure (p)
5 scalar1 (s_1)

In case of multiple scalars, only the first one will be output.

File structure

The file consists of $nt = it_{\text{end}} - it_{\text{start}} + 1$ records. Irrespective of the iteration being an integer, all data is saved in double precision real format, i.e. one record has $(ny+2) * 8\,Bytes$. The whole file has $8 * (it_{\text{end}} - it_{\text{start}} + 1) * (ny+2)\,Bytes$.

netCDF output

The python script

`tlab/scripts/python/tower2nc.py`

is available to bundle tower files from one restart into a single netCDF file which is then self-descriptive through its Meta-data. The script handles tower files of multiple restarts, but it will generate one netCDF file per restart. The python script is executed in the directory where the tower file resides. It relies on

- availability of **all** tower files belonging to one restart,
- the `grid` file used for the simulation,
- the `tlab.ini` used for the simulation (restart etc. does not matter, but the value of `Stride` in the section `[SaveTowers]` must not change),
- a properly installed `netCDF4-python` library. (On ZMAW computers, it may be necessary to load a particular python module).

The script is not thread-safe, i.e. it may only be run once at the same time one the same system. This is ascertained by a lock in the form of an empty file which is touched in the working directory.

The script moves tower files to a directory named `towerdump_`<`TIME_STAMP`>. Once the script exited successfully and integrity of the netCDF file was checked, this directory may be tarred and archived or removed. Should, for any reason, the script exit before successful completion, the files that were already moved to the `towerdump` **must be copied back** before the script is run again.

5.5 Two-point statistics

See file `dns/tools/statistics/spectra`. Based on package FFTW Frigo and Johnson [2005].

Given two scalar fields $\{a_{nm} : n = 1, \ldots, N, m = 1, \ldots, M\}$ and similarly b_{nm}, we calculate the one-dimensional co-spectra $\{E_0^x, E_1^x, \ldots, E_{N/2}^x\}$ and $\{E_0^z, E_1^z, \ldots, E_{M/2}^z\}$ normalized such that

$$\langle ab \rangle = E_0^x + 2 \sum_1^{N/2-1} E_n^x + E_{N/2}^x = E_1^z + 2 \sum_0^{M/2-1} E_m^z + E_{M/2}^z \tag{5.1}$$

The mean value is removed, such that the left-hand side is $\langle a'b' \rangle$. The Nyquist frequency energy content $E_{N/2}^x$ and $E_{M/2}^z$ is not written to disk, only the $N/2$ values $\{E_0^x, E_1^x, \ldots, E_{N/2-1}^x\}$ and the $M/2$ values $\{E_0^z, E_1^z, \ldots, E_{M/2-1}^z\}$. When $a \equiv b$, then we obtain the power spectral density.

The sum above can be interpreted as the trapezoidal-rule approximation to the integral $(L/2\pi) \int_0^{\kappa_c} 2E(\kappa) \mathrm{d}\kappa$, where $\kappa_c = \pi/h$ is the Nyquist frequency, $\Delta\kappa = \kappa_c/(N/2) = 2\pi/L$ is the uniform wavenumber spacing, $h = L/N$ is the uniform grid spacing and L is the domain size. Hence, the physical spectral function at wavenumber $\kappa_n = n\Delta\kappa$ (equivalently, wavelength L/n) is $2E_n/\Delta\kappa$.

Due to the relatively large size of the files, we split the calculations is the auto-spectra and the cross-spectra. The corresponding files containing the one-dimensional spectra along the direction Ox are `xsp` and `xCsp`, respectively, and similarly along the direction Oz. The two-dimensional co-spectra E_{nm} can also be written to disk, though the additional memory requirement can be a difficulty.

The one-dimensional cross-correlations $\{C_0^x, C_1^x, \ldots, C_{N-1}^x\}$ and $\{C_0^z, C_1^z, \ldots, C_{M-1}^z\}$ are normalized by $a_{\mathrm{rms}} b_{\mathrm{rms}}$, so that $C_0^x = C_0^z = 1$ when $a \equiv b$ and we calculate the auto-correlations. The auto-correlations are even functions and therefore only $\{C_0^x, C_1^x, \ldots, C_{N/2-1}^x\}$ and $\{C_0^z, C_1^z, \ldots, C_{M/2-1}^z\}$ are written to disk (note that we also dropped the last term $C_{N/2}^x$ and $C_{M/2}^z$.)

The corresponding files containing the one-dimensional cross-correlations along the direction Ox are `xcr` and `xCcr`, and similarly along the direction Oz. The two-dimensional cross-correlation C_{nm} can also be written to disk, though the additional memory requirement can be a difficulty.

Both form a Fourier pair according to

$$E_k = \frac{1}{N} \sum_0^{N-1} (a_{\mathrm{rms}} b_{\mathrm{rms}} C_n) \exp(-i\omega_k n) \ , \qquad a_{\mathrm{rms}} b_{\mathrm{rms}} C_n = \sum_0^{N-1} E_k \exp(i\omega_k n) \ ,$$

where $\{\omega_k = (2\pi/N)k : k = 0, \ldots, N-1\}$ is the scaled wavenumber and $i = \sqrt{-1}$ is the imaginary unit. Therefore,

$$\frac{1}{N} \sum_0^{N-1} C_n = \frac{E_0}{a_{\mathrm{rms}} b_{\mathrm{rms}}} \ , \tag{5.2}$$

relation that can be used to relate integral scales ℓ to the Fourier mode E_0, as follows. First, for the auto-correlation function, we can re-write

$$\frac{1}{N/2} \sum_0^{N/2-1} C_n = \frac{E_0}{a_{\mathrm{rms}}^2} + \frac{1 - C_{N/2}}{N} \tag{5.3}$$

because

$$\frac{1}{N} \sum_0^{N-1} C_n = \frac{1}{N} \left(\sum_0^{N/2-1} C_n + C_{N/2} + \sum_{N/2+1}^{N-1} C_n \right) = \frac{1}{N} \left(2 \sum_0^{N/2-1} C_n + C_{N/2} - 1 \right) \ ,$$

since, from periodicity, $C_N = C_0 = 1$ and, from the symmetry of the auto-correlation sequence, $\sum_{N/2+1}^{N} C_n = \sum_0^{N/2-1} C_n$. Therefore, if we use a trapezoidal rule to define the integral length scale

as

$$\ell = h\left(\frac{C_0 + C_{N/2-1}}{2} + \sum_{1}^{N/2-2} C_n\right) , \tag{5.4}$$

where $h = L/N$ is the grid spacing and L is the domain size, we obtain

$$\ell = \frac{L}{2}\left(\frac{E_0}{a_{\rm rms}^2} - \frac{2C_{N/2}}{N}\right) \simeq \frac{L}{2a_{\rm rms}^2} E_0 . \tag{5.5}$$

This result applies to both directions Ox and Oz, providing relations between ℓ^x and E^x, and ℓ^z and E^z. Each case needs to use the corresponding domain size, L^x and L^z.

These relations show that the integral length scales can be obtained directly from the spectral information without the need to calculate the correlation functions. However, the statistical convergence of those integral scales might be too poor and alternative definitions might be more useful. Also, correlation functions provide information about the degree of de-correlation achieved with a particular domain size, and about the structural organization of the flow in terms of different properties.

5.6 Summary of budget equations for second-order moments

Let f and g be fluid properties per unit mass. We use the Reynolds decomposition

$$f = \overline{f} + f' , \tag{5.6}$$

and the Favre decomposition

$$f = \widetilde{f} + f'' , \tag{5.7}$$

where

$$\widetilde{f} = \overline{\rho f}/\overline{\rho} . \tag{5.8}$$

Favre decomposition proves convenient in variable-density flows. The overbar indicates ensemble average, which is approximated by spatial average, or temporal average, or both, depending on the configuration. We will refer to it as Reynolds average, and use the term Favre average to refer to the density-weighted averages (property per unit volume). Note that $\widetilde{f''} = 0$ and $\overline{f'} = 0$, but

$$\overline{f''} = \overline{f} - \widetilde{f} , \tag{5.9}$$

which only zero in constant-density flows, when $\overline{f}$ and $\widetilde{f}$ coincide. The covariances satisfy

$$\overline{f'g'} = \overline{fg'} = \overline{f'g} = \overline{f'g''} = \overline{f''g'} = \overline{fg} - \overline{f}\,\overline{g} , \tag{5.10}$$

$$\widetilde{f''g''} = \widetilde{fg''} = \widetilde{f''g} = \widetilde{f'g''} = \widetilde{f''g'} = \widetilde{fg} - \widetilde{f}\widetilde{g} , \tag{5.11}$$

$$\overline{fg} - \overline{f}\widetilde{g} = \overline{f''g'} - \overline{f}\,\overline{g''} . \tag{5.12}$$

We consider evolution equations of the following form:

$$\partial_t(\rho f) + \nabla\cdot(\rho f\mathbf{v}) = -\nabla\cdot\mathbf{F} + \rho S_f , \tag{5.13a}$$

$$\partial_t(\rho g) + \nabla\cdot(\rho g\mathbf{v}) = -\nabla\cdot\mathbf{G} + \rho S_g . \tag{5.13b}$$

Multiplying the first equation by g and the second by f and adding them, one finds the evolution equation for the product ρfg:

$$\begin{aligned}\partial_t(\rho fg) + \nabla\cdot(\rho fg\mathbf{v}) = &- g\nabla\cdot\mathbf{F} - f\nabla\cdot\mathbf{G} \\ &+ \rho g S_f + \rho f S_g ,\end{aligned} \tag{5.14}$$

having used the equation of conservation of mass

$$\partial_t \rho + \nabla\cdot(\rho\mathbf{v}) = 0 \tag{5.15}$$

in the left hand side.

The evolution equation for the mean properties can be written as

$$\partial_t\left(\overline{\rho}\widetilde{f}\right) + \nabla\cdot\left(\overline{\rho}\widetilde{f}\widetilde{\mathbf{v}}\right) = -\nabla\cdot\left(\overline{\mathbf{F}} + \overline{\rho}\widetilde{f''\mathbf{v}''}\right) + \overline{\rho}\widetilde{S_f}\ , \tag{5.16a}$$

$$\partial_t\left(\overline{\rho}\widetilde{g}\right) + \nabla\cdot\left(\overline{\rho}\widetilde{g}\widetilde{\mathbf{v}}\right) = -\nabla\cdot\left(\overline{\mathbf{G}} + \overline{\rho}\widetilde{g''\mathbf{v}''}\right) + \overline{\rho}\widetilde{S_g}\ . \tag{5.16b}$$

Multiplying the first equation by $\widetilde{g}$ and the second by $\widetilde{f}$ and adding them, one finds the evolution equation for the product $\overline{\rho}\widetilde{f}\widetilde{g}$:

$$\begin{aligned}\partial_t\left(\overline{\rho}\widetilde{f}\widetilde{g}\right) + \nabla\cdot\left(\overline{\rho}\widetilde{f}\widetilde{g}\widetilde{\mathbf{v}}\right) =& -\widetilde{g}\nabla\cdot\overline{\mathbf{F}} - \widetilde{f}\nabla\cdot\overline{\mathbf{G}} \\ &- \nabla\cdot\left(\overline{\rho}\widetilde{f''\mathbf{v}''}\widetilde{g} + \overline{\rho}\widetilde{g''\mathbf{v}''}\widetilde{f}\right) \\ &+ \overline{\rho}\widetilde{\mathbf{v}''f''}\cdot\nabla\widetilde{g} + \overline{\rho}\widetilde{\mathbf{v}''g''}\cdot\nabla\widetilde{f} \\ &+ \widetilde{g}\overline{S_f} + \widetilde{f}\overline{S_g}\ ,\end{aligned} \tag{5.17}$$

having used the equation of conservation of mass

$$\partial_t\overline{\rho} + \nabla\cdot\left(\overline{\rho}\widetilde{\mathbf{v}}\right) = 0 \tag{5.18}$$

in the left hand side.

Averaging Eq. (5.14) and substracting Eq. (5.17) from it, one finds

$$\begin{aligned}\partial_t\left(\overline{\rho}\widetilde{f''g''}\right) + \nabla\cdot\left(\overline{\rho}\widetilde{f''g''}\widetilde{\mathbf{v}}\right) =& -\nabla\cdot\left(\overline{\rho}\widetilde{\mathbf{v}''f''g''} + \overline{\mathbf{F}'g''} + \overline{\mathbf{G}'f''}\right) \\ &+ \overline{\mathbf{F}'\cdot\nabla g''} + \overline{\mathbf{G}'\cdot\nabla f''} \\ &- \overline{\rho}\widetilde{\mathbf{v}''f''}\cdot\nabla\widetilde{g} - \overline{\rho}\widetilde{\mathbf{v}''g''}\cdot\nabla\widetilde{f} \\ &+ \overline{\rho}\widetilde{S''_f g''} + \overline{\rho}\widetilde{S''_g f''} \\ &- \overline{g''}\nabla\cdot\overline{\mathbf{F}} - \overline{f''}\nabla\cdot\overline{\mathbf{G}}\ ,\end{aligned} \tag{5.19}$$

having used the relationship

$$\overline{\rho f g\mathbf{v}} = \overline{\rho}\widetilde{f}\widetilde{g}\widetilde{\mathbf{v}} + \overline{\rho}\widetilde{\mathbf{v}''f''g''} + \widetilde{f''g''}\widetilde{\mathbf{v}} + \widetilde{f''\mathbf{v}''}\widetilde{g} + \widetilde{g''\mathbf{v}''}\widetilde{f}\ , \tag{5.20}$$

together with

$$\begin{aligned}\overline{g\nabla\cdot\mathbf{F}} - \widetilde{g}\nabla\cdot\overline{\mathbf{F}} &= \overline{g''\nabla\cdot\mathbf{F}'} + \overline{g''}\nabla\cdot\overline{\mathbf{F}} \\ &= \nabla\cdot\left(\overline{\mathbf{F}'g''}\right) - \overline{\mathbf{F}'\cdot\nabla g''} + \overline{g''}\nabla\cdot\overline{\mathbf{F}}\end{aligned} \tag{5.21}$$

and similarly for the other second-order term $f\nabla\cdot\mathbf{G}$.

5.6.1 Reynolds Stresses

Consider the momentum equations:

$$f = u_i\ , \qquad F_k = p\,\delta_{ik} - \tau_{ik}\ , \qquad S_f = b\,g_i - \epsilon_{imk}f_m u_k\ , \tag{5.22}$$

$$g = u_j\ , \qquad G_k = p\,\delta_{jk} - \tau_{jk}\ , \qquad S_g = b\,g_j - \epsilon_{jmk}f_m u_k\ . \tag{5.23}$$

Substituting into Eq. (5.19), we find

$$\partial_t R_{ij} = C_{ij} + P_{ij} - E_{ij} + \overline{\rho}^{-1}(\Pi_{ij} - \partial_k T_{ijk} + M_{ij}) + B_{ij} - F_{ij} \tag{5.24}$$

where

$$\begin{aligned}
R_{ij} &= \widetilde{u_i'' u_j''} && \text{(Reynolds-stress component)}\\
C_{ij} &= -\widetilde{u}_k\, \partial_k R_{ij} && \text{(mean advection)}\\
P_{ij} &= -R_{ik}\partial_k \widetilde{u}_j - R_{jk}\partial_k \widetilde{u}_i && \text{(mean-gradient (shear) production)}\\
E_{ij} &= \overline{\rho}^{-1}(\overline{\tau'_{jk}\partial_k u_i''} + \overline{\tau'_{ik}\partial_k u_j''}) && \text{(viscous dissipation)}\\
T_{ijk} &= \overline{\rho u_i'' u_j'' u_k''} + \overline{p' u_i'}\delta_{jk} + \overline{p' u_j'}\delta_{ik} - (\overline{\tau'_{jk} u_i''} + \overline{\tau'_{ik} u_j''}) && \text{(turbulent transport)}\\
\Pi_{ij} &= \overline{p'(\partial_j u_i'' + \partial_i u_j'')} && \text{(pressure strain)}\\
M_{ij} &= \overline{u_i''}(\partial_k \bar{\tau}_{jk} - \partial_j \bar{p}) + \overline{u_j''}(\partial_k \bar{\tau}_{ik} - \partial_i \bar{p}) && \text{(mean flux)}\\
B_{ij} &= \widetilde{b'' u_j''}\, g_i + \widetilde{b'' u_i''}\, g_j && \text{(buoyancy production-destruction)}\\
F_{ij} &= \epsilon_{imk} f_m R_{jk} + \epsilon_{jmk} f_m R_{ik} && \text{(Coriolis redistribution)}
\end{aligned}$$

Depending on symmetries, several terms can be zero (within statistical convergence). Note that if $b \equiv 1$, then $B_{ij} = 0$. The mean flux term is sometimes written as $M_{ij} = D_{ij} - G_{ij}$, the first term grouping the mean viscous stress contributions and the last term the mean pressure contributions. In cases of constant density, then $\overline{u_j''} = 0$ and $M_{ij} = D_{ij} = G_{ij} = 0$.

Contracting indices, the budget equation for the turbulent kinetic energy $K = R_{ii}/2$ reads

$$\partial_t K = C + P + B - E + \overline{\rho}^{-1}\left(\Pi - \partial_k T_k + M\right) \ . \tag{5.25}$$

Note that $F_{ii} = 0$. If the flow is solenoidal, then $\Pi = 0$.

5.6.2 Scalar Fluxes

Consider the momentum and scalar equations:

$$f = u_i \ , \qquad F_k = p\,\delta_{ik} - \tau_{ik} \ , \qquad S_f = b\, g_i - \epsilon_{imk} f_m u_k \ , \tag{5.26}$$

$$g = \zeta \ , \qquad G_k = -q_k \ , \qquad S_g = w \ . \tag{5.27}$$

Substituting into Eq. (5.19), we find

$$\partial_t R_{i\zeta} = C_{i\zeta} + P_{i\zeta} - E_{i\zeta} + \overline{\rho}^{-1}(\Pi_{i\zeta} - \partial_k T_{i\zeta k} + M_{i\zeta}) + B_{i\zeta} - F_{i\zeta} + Q_{i\zeta} \tag{5.28}$$

where

$$\begin{aligned}
R_{i\zeta} &= \widetilde{u_i''\zeta''} && \text{(scalar-flux component)}\\
C_{i\zeta} &= -\widetilde{u}_k\, \partial_k R_{i\zeta} && \text{(mean advection)}\\
P_{i\zeta} &= -R_{ik}\, \partial_k \widetilde{\zeta} - R_{\zeta k}\, \partial_k \widetilde{u}_i && \text{(mean-gradient and tilting production)}\\
E_{i\zeta} &= \overline{\rho}^{-1}(\overline{q'_k \partial_k u_i''} + \overline{\tau'_{ik}\partial_k \zeta''}) && \text{(molecular destruction)}\\
\Pi_{i\zeta} &= \overline{p'\partial_i \zeta''} && \text{(pressure-flux interaction)}\\
T_{i\zeta k} &= \overline{\rho \zeta'' u_k'' u_i''} + \overline{p' s''}\delta_{ik} - \overline{\tau'_{ik} s''} - \overline{q'_k u_i''} && \text{(turbulent transport)}\\
M_{i\zeta} &= \overline{u_i''}\partial_k \overline{q}_k + \overline{\zeta''}(\partial_k \bar{\tau}_{ik} - \partial_i \bar{p}) && \text{(mean flux)}\\
B_{i\zeta} &= \widetilde{b''\zeta''} g_i && \text{(buoyancy iteraction)}\\
F_{i\zeta} &= \epsilon_{imk} f_m R_{k\zeta} && \text{(Coriolis interation)}\\
Q_{i\zeta} &= \widetilde{w'' u_i''} && \text{(source)}
\end{aligned}$$

The mean flux term is sometimes written as $M_{i\zeta} = D_{i\zeta} - G_{i\zeta}$, the first term grouping the mean molecular flux contributions and the last term the mean pressure contributions.

5.6.3 Scalar Variance

Consider the scalar equation twice:

$$f = \zeta \,, \qquad F_k = -q_k \,, \qquad S_f = w \,, \tag{5.29}$$

$$g = \zeta \,, \qquad G_k = -q_k \,, \qquad S_g = w \,. \tag{5.30}$$

Substituting into Eq. (5.19), we find

$$\partial_t R_{\zeta\zeta} = C_{\zeta\zeta} + P_{\zeta\zeta} - E_{\zeta\zeta} + \overline{\rho}^{-1}(-\partial_k T_{\zeta\zeta k} + M_{\zeta\zeta}) + Q_{\zeta\zeta} \tag{5.31}$$

where

$$\begin{aligned}
R_{\zeta\zeta} &= \widetilde{\zeta''\zeta''} && \text{(scalar variance)} \\
C_{\zeta\zeta} &= -\widetilde{u}_k\, \partial_k R_{\zeta\zeta} && \text{(mean advection)} \\
P_{\zeta\zeta} &= -2R_{k\zeta}\, \partial_k \widetilde{\zeta} && \text{(mean-gradient production)} \\
E_{\zeta\zeta} &= 2\overline{\rho}^{-1}\overline{q'_k \partial_k \zeta''} && \text{(molecular destruction)} \\
T_{\zeta\zeta k} &= \overline{\rho \zeta''^2 u''_k} - 2\overline{q'_k \zeta''} && \text{(turbulent transport)} \\
M_{\zeta\zeta} &= 2\overline{\zeta''}\, \partial_k \overline{q}_k && \text{(mean flux)} \\
Q_{\zeta\zeta} &= 2\widetilde{w''\zeta''} && \text{(source)}
\end{aligned}$$

5.6.4 Energy Equation

To be developed.

Part II

Technical Guide

6 Numerical Algorithms

The system of equations is written as

$$\frac{\partial \mathbf{q}}{\partial t} = \mathbf{f}_q(\mathbf{q}, \boldsymbol{s}, t) \ , \qquad \frac{\partial \mathbf{s}}{\partial t} = \mathbf{f}_s(\mathbf{q}, \boldsymbol{s}, t) \ , \tag{6.1}$$

where $\boldsymbol{q}$ and $\boldsymbol{s}$ are the flow and scalar vectors. For the compressible formulation, we have

$$\mathbf{q} = (\rho, \rho u_1, \rho u_2, \rho u_3, \rho e)^T \ , \qquad \mathbf{s} = (\rho s_1, \rho s_2, \ldots)^T \ . \tag{6.2}$$

The energy equation can be also solved in terms of the total energy per unit volume $\rho(e + v^2/2)$ instead of the internal energy per unit volume ρe. For the incompressible equations, we have

$$\mathbf{q} = (u_1, u_2, u_3)^T \ , \qquad \mathbf{s} = (s_1, s_2, \ldots)^T \ . \tag{6.3}$$

The basic formulation is the method of lines, so that the algorithm is a combination of different spatial operators that calculate the right-hand side of the equations (typically, derivatives) and a time marching scheme. An implicit treatment of the diffusive terms in the incompressible case has also been implemented.

6.1 Spatial operators

Spatial operators are based on finite difference methods (FDM). There are two levels of routines. The low-level libraries contain the basic algorithms and are explained in this section. It consists of the FDM kernel library `fdm` and three-dimensional operators library `operators`. The high-level library `fields` is composed of routines that are just a combination of the low-level routines.

6.1.1 Derivatives

See file `operators/opr_partial`. Spatial derivatives are calculated using fourth- or sixth-order compact Padé schemes as described by Lele [1992] and Lamballais et al. [2011] for uniform grids and extended by Shukla and Zhong [2005] for non-uniform grids. The kernels of the specific algorithms are in the library `fdm`.

We consider FD approximations

$$\sum_{k=-1}^{k=1} a_k s'_{j+k} = h^{-1} \sum_{l=-2}^{l=2} b_k s_{j+l} \ , \qquad \sum_{k=-1}^{k=1} a_k s''_{j+k} = h^{-2} \sum_{l=-3}^{l=3} b_k s_{j+l} \ . \tag{6.4}$$

$\{s_j : j = 1, \ldots, n\}$ are the values of the function $s(x)$ evaluated at $x_j \in [x_1 \, , x_n]$. The sets $\{s'_j\}$ and $\{s''_j\}$ denote approximations to the first- and second-order derivatives evaluated at $\{x_j\}$. For the first-order derivatives, we have a 5-point stencil. For the second-order derivatives, we have a 7-point stencil. The coefficients for the first-order derivative are given in table 6.1. The coefficients for the first-order derivative are given in table 6.2.

	a_{-2}	a_{-1}	a_0	a_{+1}	a_{+2}	b_{-3}	b_{-2}	b_{-1}	b_0	b_{+1}	b_{+2}	b_{+3}	t
C2	0	0	1	0	0	0	0	$-1/2$	0	$1/2$	0	0	$1/3!\, h^2 s^{(3)}$
C4	0	$1/4$	1	$1/4$	0	0	0	$-3/4$	0	$3/4$	0	0	$-1/5!\, h^4 s^{(5)}$
C6	0	$1/3$	1	$1/3$	0	0	$-1/36$	$-7/9$	0	$7/9$	$1/36$	0	$4/7!\, h^6 s^{(7)}$
C6m	$\frac{34}{375}$	$\frac{56}{100}$	1	$\frac{56}{100}$	$\frac{34}{375}$	$\frac{51}{1250}$	$\frac{5453}{5625}$	$\frac{581}{450}$	0	$\frac{581}{450}$	$\frac{5453}{5625}$	$\frac{51}{1250}$	$4/7!\, h^6 s^{(7)}$
B1	0	0	1	0	0	0	0	0	-1	1	0	0	$1/2!\, h\ s^{(2)}$
B3	0	0	1	2	0	0	0	0	$-5/2$	2	$1/2$	0	$-2/4!\, h^3 s^{(4)}$
B5	0	$1/6$	1	$1/2$	0	0	0	$-10/18$	$-1/2$	1	$1/18$	0	$-2/6!\, h^5 s^{(6)}$

Table 6.1: Coefficients of the finite-difference formulae (6.4) for the first-order derivative. The first three rows are centered differences, the last three are biased differences. The matrix A_1 in (6.6) is constructed in terms of the coefficients $\{a_i\}$, the matrix B_1 in terms of $\{b_i\}$. The last column contains the leading order term of the local truncation error defined by (6.7).

	a_{-1}	a_0	a_{+1}	b_{-3}	b_{-2}	b_{-1}	b_0	b_{+1}	b_{+2}	b_{+3}	t
C2	0	1	0	0	0	1	-2	1	0	0	$2/4!\, h^2 s^{(4)}$
C4	$1/10$	1	$1/10$	0	0	$6/5$	$-12/5$	$6/5$	0	0	$3/5\ 1/5!\, h^4 s^{(6)}$
C6	$2/11$	1	$2/11$	0	$3/44$	$12/11$	$-51/22$	$12/11$	$3/44$	0	$-23/11\ 1/7!\, h^6 s^{(8)}$
C6b	-	1	-	-	-	-	-	-	-	-	-
B1	0	1	0	0	0	0	1	-2	1	0	$h\, s^{(3)}$
B3	0	1	11	0	0	0	13	-27	15	-1	$-2/4!\, h^3 s^{(5)}$
B5	$1/10$	1	$-7/20$	0	0	$99/80$	-3	$186/80$	$-3/5$	$3/80$	$3/5\ 1/5!\, h^5 s^{(7)}$

Table 6.2: Coefficients of the finite-difference formulae (6.4) for the second-order derivative. The first three rows are centered differences, the last three are biased differences. The matrix A_2 in (6.6) is constructed in terms of the coefficients $\{a_i\}$, the matrix B_2 in terms of $\{b_i\}$. The last column contains the leading order term of the local truncation error defined by (6.7).

Global schemes are constructed as a combination of n of these formulae, using biased finite differences at the boundaries in case of non-periodicity. For the first-order derivative, we define the global algorithm (35653) by using the centered scheme C6 at the $(n-4)$ interior points, and the biased schemes B5 at $j=2$ and B3 at $j=1$ with the corresponding symmetric counterpart at $j=n-1$ and $j=n$, respectively [Carpenter et al., 1993]. For the second-order derivative, we define the global algorithm (3466b43) by using the centered scheme C6b at the $(n-6)$ interior points, C6 at $j=3$ and $j=n-2$, C4 at $j=2$ and $j=n-1$, and the biased scheme B3 at $j=1$ with the corresponding symmetric counterpart at $j=n$. We tried the biased scheme B5 instead of the centered scheme C4 for $j=2$ and $j=n-1$; the eigenvalues were closer to the spectral case, but the truncation errors e_j were larger despite t_j being smaller (see below).

We define the n-dimensional vectors

$$\mathbf{s}^T = [s_1\ , \dots s_n]\ , \qquad (\mathbf{s}')^T = (\delta_x \mathbf{s})^T = [s'_1\ \dots s'_n]\ , \qquad (\mathbf{s}'')^T = (\delta_{xx}\mathbf{s})^T = [s''_1\ \dots s''_n]\ . \tag{6.5}$$

The global schemes can be represented as

$$A_1\, \delta_x \mathbf{s} = (1/h) B_1\, \mathbf{s}\ , \qquad A_2\, \delta_{xx}\mathbf{s} = (1/h^2) B_2\, \mathbf{s} \tag{6.6}$$

where $h = (x_n - x_1)/(n-1)$ is a reference space step and the square matrices $A = (a_{ij})$ and $B = (b_{ij})$ are narrow banded. If periodic boundary conditions are used, these are imposed at x_{n+1} and not at x_n, i.e. $s(x_{n+1}) = s(x_1)$. The size of the domain is then $L = n(x_n - x_1)/(n-1) = nh$ instead of $x_n - x_1 = (n-1)h$, and we do not save the information at x_{n+1}. The matrices A and B are then circulant instead of banded. The Thomas' algorithm is used to solve those linear systems. An LU decomposition is performed during the initialization and equations are normalized such that the right-hand side contains at least one diagonal of just ones, so as to save memory and computational time (see routine `fdm_initialize`). For analysis, one can consider equation (6.6)

as the definition of linear finite-difference operators $\delta_x : \mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{s}' = \delta_x \mathbf{s} = (1/h)(A_1^{-1}B_1)\mathbf{s}$, and $\delta_{xx} : \mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{s}'' = \delta_{xx}\mathbf{s} = (1/h^2)(A_2^{-1}B_2)\mathbf{s}$ [Mellado and Ansorge, 2012].

The local truncation errors of the FD approximations (6.6) to the first-order derivative, $\{e_{1,j} = ds/dx(x_j) - s'_j \,:\, j = 1, \ldots, n\}$, are given by

$$\mathbf{e_1} = -A_1^{-1}\mathbf{t_1} \,, \qquad \mathbf{t_1} = \frac{1}{h} B_1 \begin{pmatrix} s(x_1) \\ \vdots \\ s(x_n) \end{pmatrix} - A_1 \begin{pmatrix} \frac{ds}{dx}(x_1) \\ \vdots \\ \frac{ds}{dx}(x_n) \end{pmatrix} . \tag{6.7}$$

A similar expression can be derived for the second-order derivatives. The components of $\mathbf{t_1}$ and $\mathbf{t_2}$ can be found in table 6.1 and table 6.2.

For notational convenience in the following discussion on Fourier analysis, we change the index so that it varies between $j = 0$ and $j = n - 1$. We can define a new sequence of numbers $\{\hat{s}_k\}_0^{n-1}$ from $\{s_j\}_0^{n-1}$ by

$$\hat{s}_k = \frac{1}{n} \sum_0^{n-1} s_j \exp(-i\omega_k j) \,, \qquad s_j = \sum_0^{n-1} \hat{s}_k \exp(i\omega_k j) \,, \tag{6.8}$$

where $\{\omega_k = (2\pi/n)k \,:\, k = 0, \ldots, n-1\}$ is the scaled wavenumber and $i = \sqrt{-1}$ is the imaginary unit. We use the library FFTW in the code for this transformation [1]. Note that $\hat{s}_{k+n} = \hat{s}_k$ because $\exp(-i2\pi j) = 1$ for any integer number j, so that we can write

$$s_j = \sum_{-n/2+1}^{n/2} \hat{s}_k \exp[i\omega_k j] \,. \tag{6.9}$$

The expression above coincides with the Fourier series $\sum_{-n/2}^{n/2} \hat{s}_k \exp[i\kappa_k x]$ of the function $s(x)$ over the interval $[0, L]$ particularized at the grid points $\{x_n\}$ provided that the spectral content of the function $s(x)$ beyond the Nyquist frequency $\kappa_{n/2} = (2\pi/L)(n/2) = 2\pi/(2h) = \pi/h$ is zero. Then we have the relation $\kappa_k h = \omega_k$ between the wavenumber $\kappa_k = (2\pi/L)k$ and the scaled wavenumber ω_k. In principle, both $\{\hat{s}_k\}$ from $\{s_j\}$ are complex numbers; however, the sequence $\{\hat{s}_k\}$ is typically real and we only need to know the Fourier modes between $k = 0$, the mean value, and $k = n/2$, the Nyquist frequency, because of the symmetry. Then, ω_k varies between 0 and π and κ_k varies between 0 and π/h. A third quantity sometimes used in the discussion is the number of points per wavelength $\mathrm{PPW}_k = (L/k)/h = 2\pi/\omega_k$; for a given wavelength L/k, or wavenumber κ_k, reducing the grid step h and thus increasing resolution – increasing PPW_k – means reducing the scaled wavenumber ω_k towards zero.

With previous framework we can use von Neumann analysis to understand the properties of the FDM. We know that the exact values of $\{s'_j\}$ and $\{s''_j\}$ under the conditions stated above are

$$s'(x_j) = \sum_0^{n-1} (i\omega_k/h)\hat{s}_k \exp(i\omega_k j) \,, \qquad s''(x_j) = \sum_0^{n-1} (-\omega_k^2/h^2)\hat{s}_k \exp(i\omega_k j) \,, \tag{6.10}$$

having used $\kappa_k = \omega_k/h$. The FDM approximations can be written as

$$s'_j = \sum_0^{n-1} (\lambda_1/h)\hat{s}_k \exp(i\omega_k j) \,, \qquad s''_j = \sum_0^{n-1} (\lambda_2/h^2)\hat{s}_k \exp(i\omega_k j) \,. \tag{6.11}$$

The deviation of the complex functions $\lambda_1(\omega)$ and $\lambda_2(\omega)$ from the exact values $i\omega$ and $-\omega^2$ measures the FDM discretization error (see figure 6.1). One possible way to quantify this error is by means of

[1] visit http://www.fftw.org/

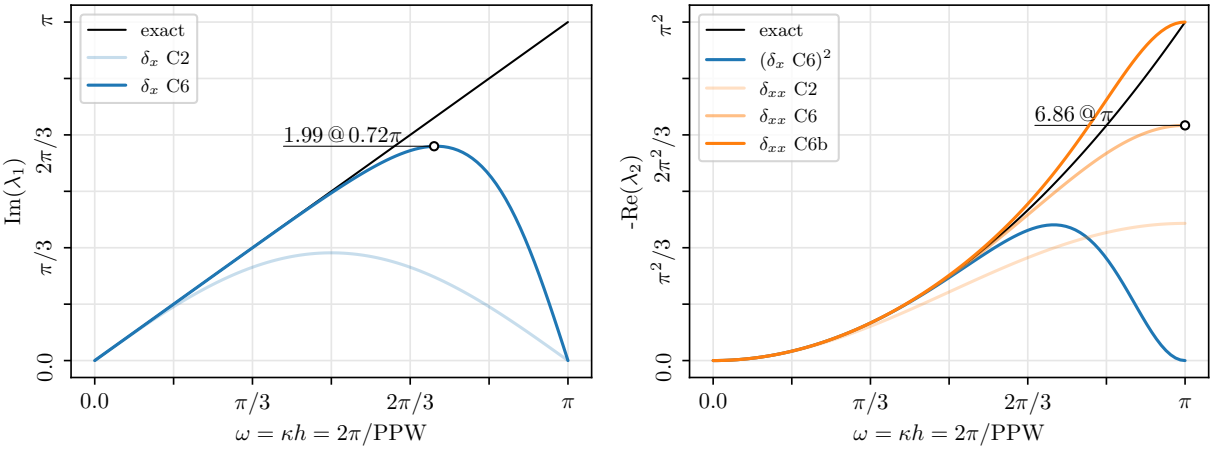

Figure 6.1: Modified wavenumbers of the FDM approximations to the (left) first-order derivative and (right) second-order derivatives. The blue line in right panel corresponds to the operator $\delta_x\delta_x = (A_1^{-1}B_1)^2$. The interval $[-\pi, 0$ is simply the anti-symmetric extension in the left panel, and the symmetric extension in the right panel. The scheme compact a corresponds to the 5-point stencil in Lele [1992] and compact b corresponds to the 7-point stencil in Lamballais et al. [2011]. The latter is the default scheme in the code.

the resolving efficiency, defined as the number of points per wavelength PPW required to maintain errors in the corresponding transfer function below a specified level (or, equivalently, a specified error in the dispersion velocity of the linear advection problem). In these terms, for the first-order derivative, an error of 1% requires 4 PPW in the case of the implicit compact scheme C6, whereas the second-order explicit scheme C2 requires 25 PPW [Lele, 1992, Lomax et al., 1998]. This difference is even higher if the common reference error of 0.1% is retained, for which the previous finite difference schemes require 6 PPW and 100 PPW, respectively. The corresponding resolution requirements for the second-order derivative using a compact scheme are similar to those of the first-order derivative; the second-order central explicit FDM improves slightly and only needs 18 PPW for 1% error and 67 PPW for 0.1% error. These errors in the transfer function can be understood as errors in the exponential rate of decrease of a given wave caused by the diffusion operator. The scheme compact b corresponds to the 7-point stencil described in Lamballais et al. [2011]. Last, we also note that, as we increase resolution, h decreases and ω_k moves towards the origin in figure 6.1 for a fixed wavenumber κ_k; the departure at the origin of the approximation from the exact value gives then the order of the FDM.

It is also useful to use the notation

$$\hat{\mathbf{s}} = W\,\mathbf{s} \qquad \mathbf{s} = W^{-1}\hat{\mathbf{s}}\,. \tag{6.12}$$

for the discrete Fourier transform (DFT) defined by equation (6.8), where W is the DFT matrix. Let us consider the first-order derivative in equation (6.6). Then, we can write

$$\hat{\mathbf{s}'} = W\,\mathbf{s}' = [(1/h)W(A_1^{-1}B_1)W^{-1})W\mathbf{s} = (1/h)\Lambda_1\hat{\mathbf{s}}\,. \tag{6.13}$$

From this relation between the vectors $\hat{\mathbf{s}'}$ and $\hat{\mathbf{s}}$ and equation (6.11), we deduce that the array Λ_1 is diagonal with $\{\lambda_k\}_0^{n-1}$ as diagonal elements. Moreover, since W is invertible, W represents just a similarity transformation – a change of base – and therefore we know that the matrices $A_1^{-1}B_1$ and Λ_1 have the same eigenvalues. Hence, $\{\lambda_k\}_0^{n-1}$ is simply the set of eigenvalues of $A_1^{-1}B_1$; figure 6.1 shows the curve through the imaginary part of half of them. In the complex plane, the numbers $\{\lambda_{1,k}\}_0^{n-1}$ corresponding to the first-order derivative move in the imaginary axis, and the numbers $\{\lambda_{2,k}\}_0^{n-1}$ corresponding to the second-order derivative move in the negative part of the real axis.

We can calculate the FD approximation to the second-order derivative as $\delta_x\delta_x\mathbf{s} = (A_1^{-1}B_1)^2$, that is, to consecutively apply the FD approximation to the first-order derivative twice. However, the

spectral transfer function of the FD operator $\delta_x\delta_x$ falls to zero at the Nyquist frequency, as shown in figure 6.1, which results in a very poor representation of the diffusion terms at the high wavenumbers. This property can become very important because the errors derived from the aliasing in calculating the non-linear terms accumulate and the use of filter might become unavoidable to have stable simulations. Hence, we use a direct discretization of the second-order derivative operator, compact a or compact b.

Last, a uniform grid $\{x_j = x_1+(j-1)h : j = 1,\ldots,n\}$ has been considered so far. If a non-uniform grid $\{x_j : j = 1,\ldots,n\}$ is employed instead, we can define $\mathbf{x}' = (1/h)A_1^{-1}B_1\mathbf{x}$ from the mapping between the computational and the physical domains and the calculation of the approximation $\delta_x\mathbf{s}$ to the first-order derivative is given by

$$(A_1D_1)\,\delta_x\mathbf{s} = (1/h)B_1\mathbf{s}\;, \tag{6.14}$$

that is, A_1 should be replaced by A_1D_1, where $D_1 = \mathrm{diag}(\mathbf{x}')$ is a diagonal matrix with $\{x'_j\}$ as diagonal elements. Similarly, for the FD approximation to the second-order derivative, we obtain

$$(A_2D_1^2)\,\delta_{xx}\mathbf{s} = (1/h^2)B_2\mathbf{s} - (A_2D_2)\,\delta_x\mathbf{s}\;, \tag{6.15}$$

where $D_2 = \mathrm{diag}(\mathbf{x}'')$ is again a diagonal matrix with $\{x''_j\}$ as diagonal elements; these elements are the components of the vector $\mathbf{x}'' = (1/h^2)A_2^{-1}B_2\mathbf{x}$. As a result of using this Jacobian formulation for non-uniform grids, we need to calculate the approximation to the first-order derivative in order to calculate $\delta_{xx}\mathbf{s}$. In general, that is not a problem because we need both, the first- and the second-order derivatives, in all the transport equations.

We could reformulate the calculation of the second-order derivative as

$$(A_2D_1^2)\,\mathbf{s}^* = (1/h^2)B_2\mathbf{s}\;, \tag{6.16a}$$

$$\delta_{xx}\mathbf{s} = \mathbf{s}^* - (D_1^{-2}D_2)\,\delta_x\mathbf{s}\;, \tag{6.16b}$$

where $D_1^{-2}D_2$ is a diagonal matrix and could be precomputed. This formulation is clearer and useful for the stability analysis, but not faster.

A direct formulation of compact FD approximation to the second-order derivative for non-uniform grids is needed, however, when using the implicit temporal scheme for the diffusion terms in order to solve exactly the corresponding Helmholtz equations, without any approximation. Such a direct formulation leads to the system form (6.6). We followed Shukla and Zhong [2005].

6.1.2 Advection and Diffusion

The non-linear advection terms can be formulated in conservative, convective and skew-symmetric forms [Blaisdell et al., 1996, Kravchenko and Moin, 1997]. The molecular transport terms can be formulated in the conservative and non-conservative forms (this latter if transport coefficients are constant).

In the convective formulation, the routines `OPR_BURGERS_*` combine the first and second-order derivative operators as

$$f = \epsilon s'' - us' \tag{6.17}$$

where s is a scalar field, u a velocity field, and ϵ is the diffusivity. The combination reduces transpositions, either locally or across processors. The reason is that, in general, we need 2 transpositions for s'', forward and backward, and similarly for s', which amounts to 4 transpositions. In the the combined form, we need 1 forward transposition for s and 1 for u, and then 1 backward transposition

for the result f. In total, 3 transpositions. The addition and multiplication operations are done in transposed space. If $u = s$, then it is only 2 transpositions that we need. When $u \neq s$, then the transposed velocity needs to be passed through the arguments in case it is needed.

In the anelastic mode, the operator calculates

$$f = \rho_{\rm bg}^{-1} \epsilon s'' - u s' \,, \tag{6.18}$$

where $\rho_{\rm bg}$ is the background density profile.

We could reformulate the calculation of $\mathbf{f}$ by combining the linear operation with the calculation of the second derivative, which yields

$$\epsilon^{-1}(A_2 D_1^2)\,\mathbf{f} = (1/h) B_2 \mathbf{s} - A_2 (D_2 + \epsilon^{-1} D_1^2 \mathbf{u})\,\delta_x \mathbf{s} \,, \tag{6.19}$$

or

$$\epsilon^{-1}(A_2 D_1^2)\,\mathbf{f}^* = (1/h) B_2 \mathbf{s} \,, \tag{6.20a}$$

$$\mathbf{f} = \mathbf{f}^* - (\epsilon D_1^{-2} D_2 + \mathbf{u})\,\delta_x \mathbf{s} \,, \tag{6.20b}$$

where $\epsilon D_1^{-2} D_2$ is a diagonal matrix and could be precomputed. This formulation is clearer, but not sure if faster, and we need more memory because $\epsilon D_1^{-2} D_2$ varies with ϵ.

6.1.3 Filters

See file `operators/opr_filter`. The kernels of the specific algorithms are in the library `filters`. Used in previous versions for long-term stability, filters are now mainly used for post-processing.

There are global filters and directional filters. Global filters are spectral filters in the horizontal plane, and Helmholtz-based filters. Band spectral filters remove the energy content outside a band $[\alpha_1, \alpha_2]$ in frequency space. Erf spectral filters are low- and high-pass filters defined at α_1 with a thickness α_2 in the logarithm of the frequency ($\alpha_1 > 0$ for high-pass filter, and $\alpha_1 < 0$ for low-pass filter). Helmholtz-based filters are defined by

$$(1 - \alpha^2 \nabla^2) f = s \,, \tag{6.21}$$

where s is the original field, f is the filtered field, and $\alpha = \lambda/(2\pi)$ where λ is the filter size. This equation is transformed into the Helmholtz equation and solved as explained in section 6.1.6. This filter is motivated by the Navier-Stokes-alpha model [Foias et al., 2001]. Along the vertical non-periodic direction, we can impose Dirichlet or Neumann boundary conditions, where the former option maintains the value of the field at the boundary and the latter option imposes a zero gradient.

Directional filters are a sequence of one-dimensional filters in each direction. There are compact filters, which are implemented following Lele [1992], and top-hat filters (or box filter), which are implemented using the trapezoidal rule to discretize the integral.

As an example of top-hat filter, let us consider a top-hat filter of a function $s(x)$ where the filter size is four times the grid size. Then, the value at the grid node x_i of the filtered function $f(x)$ is

$$f_i = \frac{1}{\Delta_{f,i}} \left\{ \frac{s_{i-2} + s_{i-1}}{2} \Delta_{i-2} + \frac{s_{i-1} + s_i}{2} \Delta_{i-1} + \frac{s_i + s_{i+1}}{2} \Delta_i + \frac{s_{i+1} + s_{i+2}}{2} \Delta_{i+1} \right\} \tag{6.22}$$

where

$$\Delta_i = x_{i+1} - x_i \,, \tag{6.23a}$$

$$\Delta_{f,i} = x_{i+2} - x_{i-2} = \Delta_{i-2} + \Delta_{i-1} + \Delta_i + \Delta_{i+1} \,. \tag{6.23b}$$

The linear operation can be written as

$$f_i = c_{1,i}s_{i-2} + c_{2,i}s_{i-1} + c_{3,i}s_i + c_{4,i}s_{i+1} + c_{5,i}s_{i+2} \ , \tag{6.24}$$

where the coefficients are

$$c_{1,i} = (2\Delta_{f,i})^{-1}\Delta_{i-2} \ , \tag{6.25a}$$
$$c_{2,i} = (2\Delta_{f,i})^{-1}(\Delta_{i-1} + \Delta_{i-2}) \ , \tag{6.25b}$$
$$c_{3,i} = (2\Delta_{f,i})^{-1}(\Delta_i + \Delta_{i-1}) \ , \tag{6.25c}$$
$$c_{4,i} = (2\Delta_{f,i})^{-1}(\Delta_{i+1} + \Delta_i) \ , \tag{6.25d}$$
$$c_{5,i} = (2\Delta_{f,i})^{-1}\Delta_{i+1} \ . \tag{6.25e}$$

For the tophat filter, we can imposed free or solid boundary conditions along the non-periodic directions. The former option uses ghost cells where the function is the linear extrapolation of $s(x)$ based on the values at the two nodes next to the boundary. The latter option uses ghost cells where the function is constant and equal to the boundary value of $s(x)$. As an example, consider the solid boundary near $i = 1$ for the example above:

$$f_1 = \tilde{c}_{3,1}s_1 + c_{4,1}s_2 + c_{5,1}s_3 \ , \tag{6.26a}$$
$$f_2 = \tilde{c}_{2,2}s_1 + c_{3,2}s_2 + c_{4,2}s_3 + c_{5,2}s_4 \ , \tag{6.26b}$$

where the coefficients are

$$i = 1 \ : \quad \tilde{c}_{3,1} = c_{1,1} + c_{2,1} + c_{3,1} = (2\Delta_{f,1})^{-1}5\Delta_1 = (2\Delta_{f,i})^{-1}[2(n-i)+3]\Delta_1 \ , \tag{6.27a}$$
$$i = 2 \ : \quad \tilde{c}_{2,2} = c_{1,2} + c_{2,2} \qquad = (2\Delta_{f,2})^{-1}3\Delta_1 = (2\Delta_{f,i})^{-1}[2(n-i)+3]\Delta_1 \ , \tag{6.27b}$$

together with

$$i = 1 \ : \quad \Delta_{f,1} = 3\Delta_1 + \Delta_2 \qquad = (n-i+2)\Delta_1 + \ldots + \Delta_{n+i-1} \ , \tag{6.28a}$$
$$i = 2 \ : \quad \Delta_{f,2} = 2\Delta_1 + \Delta_2 + \Delta_3 = (n-i+2)\Delta_1 + \ldots + \Delta_{n+i-1} \ , \tag{6.28b}$$

Generalizing to a filter halfsize n, we obtain:

$$i = 1, \ldots, n \ : \quad \Delta_{f,i} = (n-i+1)\Delta_1 + \sum_{k=1}^{n+i-1} \Delta_k \ , \tag{6.29a}$$
$$i_{\rm org} = n - i + 1 \ , \tag{6.29b}$$
$$\tilde{c}_{i_{\rm org}+1,i} = (2\Delta_{f,i})^{-1}[2(n-i+1)+1]\Delta_1 \ , \tag{6.29c}$$
$$f_i = \tilde{c}_{i_{\rm org}+1,i}\, s_1 + \sum_{k=2}^{n+i} c_{i_{\rm org}+k,i}\, s_k \ . \tag{6.29d}$$

Near the boundary $i = i_{\max}$, we obtain:

$$i = i_{\max} - n + 1, \ldots, i_{\max} \ : \quad \Delta_{f,i} = \sum_{k=i-n}^{i_{\max}-1} \Delta_k + (n + i - i_{\max})\Delta_{i_{\max}-1} \ , \tag{6.30a}$$
$$i_{\rm org} = n - i + 1 \ , \tag{6.30b}$$
$$\tilde{c}_{i_{\rm org}+i_{\max},i} = (2\Delta_{f,i})^{-1}[2(n + i - i_{\max}) + 1]\Delta_{i_{\max}-1} \ , \tag{6.30c}$$
$$f_i = \sum_{k=i-n}^{i_{\max}-1} c_{i_{\rm org}+k,i}\, s_k + \tilde{c}_{i_{\rm org}+i_{\max},i}\, s_{i_{\max}} \ . \tag{6.30d}$$

In any case, the algorithm can be expressed formally as a matrix multiplication, the diagonals being calculated once in the initialization step and being stored in the filter-structure data.

6.1.4 Fourier transform

See file `operators/opr_fourier`. It is based on the FFTW library and it has been already discussed in the previous section (see text around equation (6.8)).It is used in the pre-processing (generation of the initial random field), in the post-processing (spectral analysis), and also during the simulation (Poisson and Helmholtz solvers).

The Fourier transform is applied by default to an array `imax_total`$\times$`(jmax_total+2)`$\times$`kmax_total`. The reason to add two additional planes {`jmax_total+1,jmax_total+2`} is that we need them for the boundary conditions of the Poisson equations, and we make that the standard procedure. If not needed, then these two planes contain simply zeros.

The sequence of transformations is $Ox \rightarrow Oz \rightarrow Oy$. The transformed field contains the Nyquist frequency, so it needs an array `(imax_total/2+1)`$\times$`(jmax_total+2)`$\times$`kmax_total` of complex numbers.

Given the scalar field s, the power spectral density $\{E_0,\, E_1,\, \ldots,\, E_{N/2}\}$ is normalized such that

$$\langle s^2 \rangle = E_0 + 2 \sum_{0}^{N/2-1} E_n + E_{N/2} \, . \tag{6.31}$$

The mean value is typically removed, such that the left-hand side is s^2_{rms}. The Nyquist frequency energy content $E_{N/2}$ is not written to disk, only the $N/2$ values $\{E_0,\, E_1,\, \ldots,\, E_{N/2-1}\}$.

6.1.5 Poisson equation

See file `operators/opr_poisson`. Given the scalar field s, obtain the scalar field f such that

$$\nabla^2 f = s \, , \tag{6.32}$$

complemented with appropriate boundary conditions. The current version only handles cases with periodic boundary conditions along Ox and Oz. It performs a Fourier decomposition along these two directions, to obtain the a set of finite difference equations along Oy of the form

$$\delta_x \delta_x \mathbf{f}|_j - (\lambda_1/h)^2 \mathbf{f}|_j = \mathbf{s}|_j \, , \qquad j = 2, \ldots, n-1 \, , \tag{6.33}$$

$\lambda_1 \in \mathbb{R}$, where boundary conditions need to be provided at $j = 1$ and $j = n$. The algorithm is described in Mellado and Ansorge [2012]. These routines are in the source file `dns/opr_fde_pool`.

6.1.6 Helmholtz equation

See file `operators/opr_helmholtz`. Given the scalar field s, obtain the scalar field f such that

$$\nabla^2 f + \alpha f = s \, , \tag{6.34}$$

complemented with appropriate boundary conditions. The current version only handles cases with periodic boundary conditions along Ox and Oz. The algorithm is similar to that used for the Poisson equation. It performs a Fourier decomposition along these two directions, to obtain the a set of finite difference equations along Oy of the form

$$\delta_x \delta_x \mathbf{f}|_j - (\lambda_2/h^2 - \alpha) \mathbf{f}|_j = \mathbf{s}|_j \, , \qquad j = 2, \ldots, n-1 \, , \tag{6.35}$$

$\lambda_2 \in \mathbb{R}$, where boundary conditions need to be provided at $j = 1$ and $j = n$. The difference is that, for the Helmholtz equation, we also include the case in which the second-order derivative is implemented in terms of the δ_{xx} FDM operator, not only the $\delta_x \delta_x$ FDM operator.

6.2 Time marching schemes

See file `tools/dns/time_rungekutta`. The time advancement is based on Runge-Kutta methods (RKM).

6.2.1 Explicit schemes

We can use three- or five-stages, low-storage RKM that gives third- or fourth-order accurate temporal integration, respectively [Williamson, 1980, Carpenter and Kennedy, 1994]. The essential feature is that only two levels are needed at a time, reducing thereby the number of three-dimensional arrays compared to the convectional Runge-Kutta schemes. In particular, the implementation is

$$\begin{aligned} &\mathbf{h} = 0 \\ &\left.\begin{array}{rrl} \mathbf{h} \leftarrow & \mathbf{h} & +\mathbf{f}(\mathbf{s}, t + C_M\tau) \\ \mathbf{s} \leftarrow & \mathbf{s} & +B_M\tau\ \mathbf{h} \\ \mathbf{h} \leftarrow & A_M\ \mathbf{h} & \end{array}\right] M \text{ times,} \end{aligned}$$

where $M = 3$ or $M = 5$, $C_1 = 0$ and we do not need the last step for the last stage. The stability properties for the biased finite difference schemes are considered in Carpenter et al. [1993]. The incompressible formulation follows Wilson et al. [1998].

The analysis of time marching schemes is based on the linearization of of the right-hand size (6.1), and its diagonalization to obtain a set of ODEs of the form

$$\frac{ds_j}{dt} = \lambda_j s_j \ , \tag{6.36}$$

where s_j can be a complex function of the real variable t, and λ_j is the corresponding eigenvalue, a complex number. Given the initial condition s^n at t_n, the RKM provides an approximation s^{n+1} to $s(t_{n+1})$, where the time step is $\tau = t_{n+1} - t_n$. The ratio provides the amplification factor $r = s^{n+1}/s^n$. The exact amplification factor is $\exp(\lambda\tau)$, whereas that from the discrete method is

$$r = 1 + \sum_1^p c_k(\lambda\tau)^k \ , \tag{6.37}$$

the coefficients depending on the RKM method and p being the number of stages. The region of absolute stability is the region of the complex plane $\lambda\tau$ for which $|r| < 1$. In addition, we can compare the approximation with the exact value

$$\rho\exp(i\theta) = \frac{r}{\exp(\lambda\tau)} \ , \tag{6.38}$$

such that $\rho - 1$ and θ represents the amplitude (or dissipation) error and the phase (or dispersion) error, respectively [Hu et al., 1996].

For the fourth-order five-step Runge-Kutta method that we use in the code, one finds (see figure 6.2)

$$r = 1 + \sum_1^4 \frac{1}{k!}(\lambda\tau)^k + \frac{1}{200}(\lambda\tau)^5 \ . \tag{6.39}$$

The equation above shows the forth-order accuracy, since the first term that deviates from the Taylor series of the exponential function is proportional to $(\lambda\tau)^5$. The crossing points of the boundary of

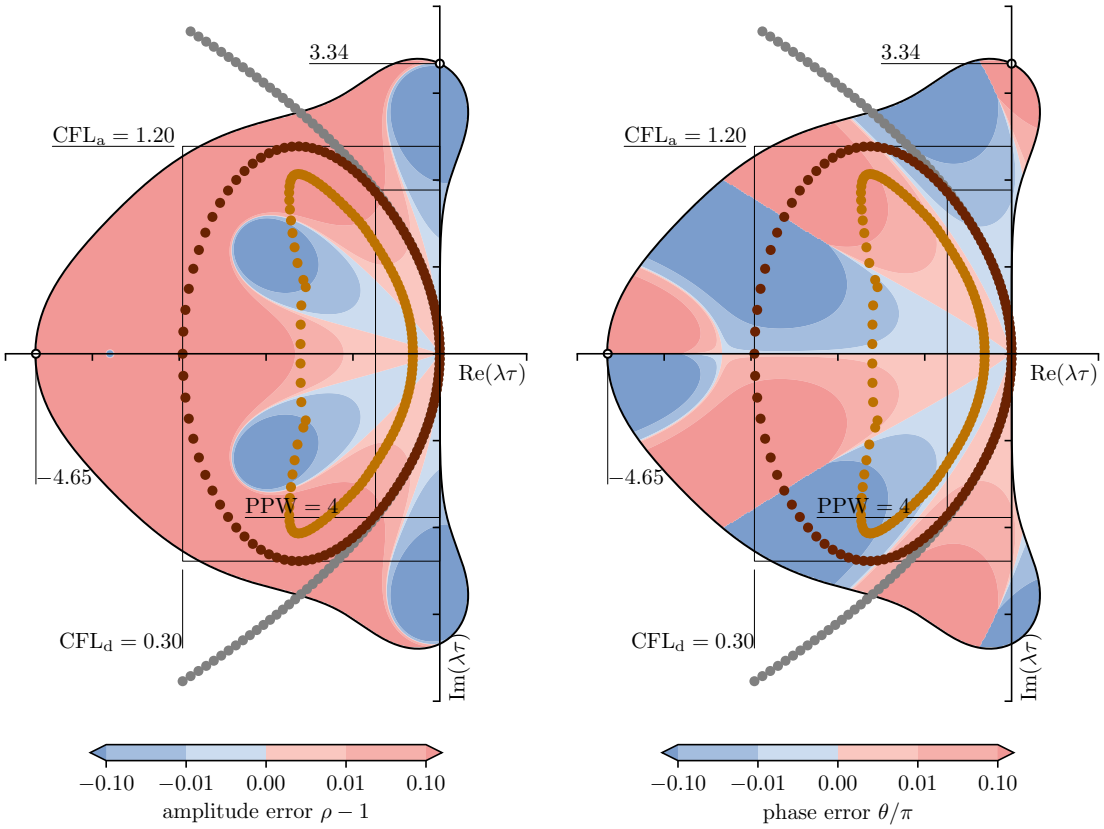

Figure 6.2: Fourth-order explicit Runge-Kutta scheme. Colored area indicates the stability region. The five zeros of the polynomial $r = r(\lambda\tau)$ are enclosed by the blue closed regions in the left panel. Markers indicate the eigenvalues for the advection-diffusion equation and $n = 128$: gray, periodic case with spectral FDMs; dark brown, periodic case with compact FDMs; light brown, Dirichlet case with compact FDMs.

the stability region with the real and imaginary axis are $(\lambda\tau)_r \simeq -4.65$ and $(\lambda\tau)_i \simeq \pm 3.34$. These numbers determine the maximum CFL numbers associated with the advection-diffusion equation, which is the basis for many non-reacting flows (a source term might add an additional stability constraint for). Assuming periodic boundary conditions for simplicity, we can diagonalize the original system to the set of equations

$$\frac{d\hat{s}_j}{dt} = (ic\lambda_{1,j}/h - \nu\lambda_{2,j}/h^2)\hat{s}_j \;, \qquad j = 0, \ldots, n-1 \;, \tag{6.40}$$

according to the eigenvalue analysis discussed in section 6.1.1. In the expression above, c is a constant representing an advection velocity and ν is the viscosity. The expression within parenthesis is the eigenvalue λ_j and it needs to belong to the stability region shown for the algorithm to be stable. This condition implies

$$\frac{\nu\tau}{h^2} < \frac{|(\lambda\tau)_r|}{\max_j\{\lambda_{2,j}\}} \;, \qquad \frac{c\tau}{h} < \frac{|(\lambda\tau)_i|}{\max_j\{\lambda_{1,j}\}} \;. \tag{6.41}$$

The left-hand sides in the expressions above are the CFL numbers $\mathrm{CFL_d}$ and $\mathrm{CFL_a}$ for the diffusion and the advection operators, respectively. The right-hand sides provide the upper bounds $\mathrm{CFL_{d,max}} = 4.65/6.86 = 0.68$ ($4.65/\pi^2 = 0.47$ if we use Lamballais et al. [2011]) and $\mathrm{CFL_{a,max}} = 3.34/1.99 = 1.68$ having used for $\max_j\{\lambda_{2,j}\}$ and $\max_j\{\lambda_{1,j}\}$ the values shown in figure 6.1.

Besides stability, we also want a small error. Figure 6.2 shows the amplitude and phase errors as defined in (6.38). That figure explains the reason to use CFL numbers that are smaller than the maximum for stability. The value $\mathrm{CFL_a} = 0.7\,\mathrm{CFL_{a,max}} \simeq 1.2$ is used in the code by default. In

the real axis, the code uses by default $1/4$ of the limit in the imaginary axis, that is, $\mathrm{CFL_d} = 0.3$. Wavenumbers corresponding to more than 4 PPW fall approximately within 10% error.

For the third-order Runge-Kutta method that we use in the code, one finds (see figure 6.3)

$$r = 1 + \sum_{1}^{3} \frac{1}{k!}(\lambda\tau)^k \,. \tag{6.42}$$

The maximum CFL numbers to guarantee stability for the advection and diffusion operators are $\mathrm{CFL_{a,max}} = 1.73/1.989 = 0.871$ and $\mathrm{CFL_{d,max}} = 2.57/6.857 = 0.375$ ($2.57/\pi^2 = 0.26$ if we use Lamballais et al. [2011]), respectively. The value $\mathrm{CFL_a} = 0.7\,\mathrm{CFL_{a,max}} \simeq 0.6$ is used in the code by default. In the real axis, the code uses by default $1/4$ of the limit in the imaginary axis, that is, $\mathrm{CFL_d} = 0.15$.

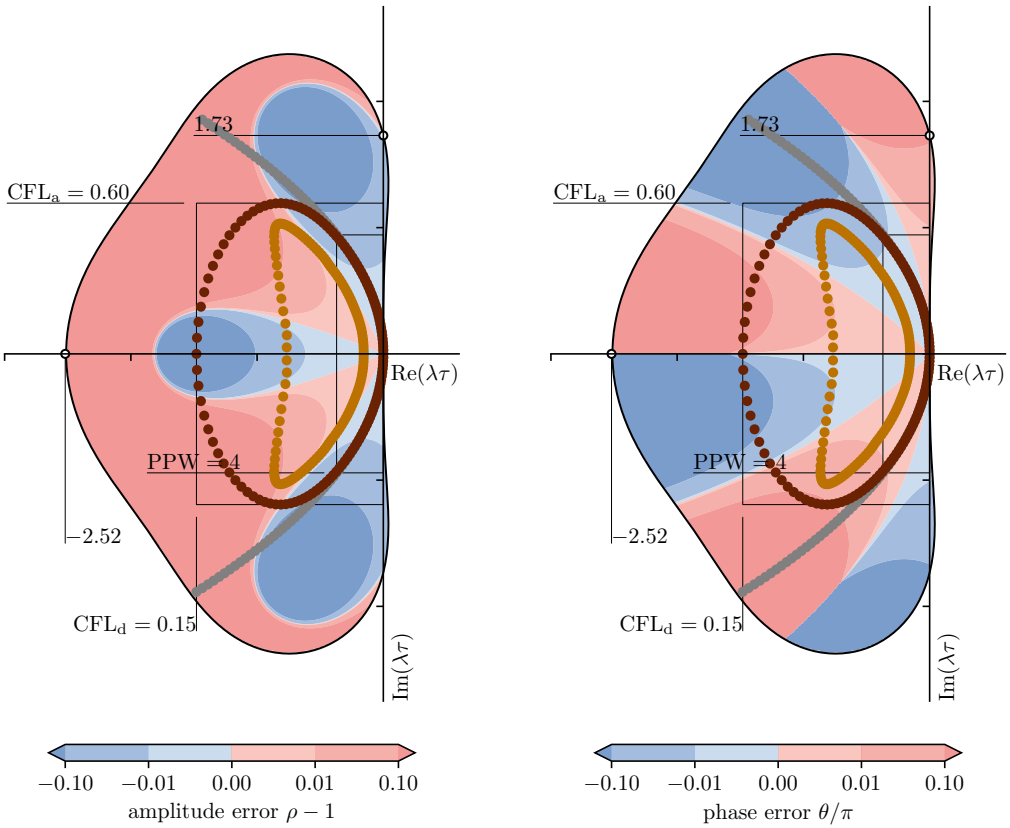

Figure 6.3: Third-order explicit Runge-Kutta scheme. Colored area indicates the stability region. The five zeros of the polynomial $r = r(\lambda\tau)$ are enclosed by the blue closed regions in the left panel. Markers indicate the eigenvalues for the advection-diffusion equation and $n = 128$: gray, periodic case with spectral FDMs; dark brown, periodic case with compact FDMs; light brown, Dirichlet case with compact FDMs.

We have considered the periodic case, which is easier because the eigenvalues can be obtained analytically and both the diffusion and advection operator have the same eigenvectors. For the non-periodic case, we generally need to calculate the eigenvalues numerically. For instance, let us consider the advection equation inside the domain $[x_1, x_n]$ with a positive advection velocity u and (therefore) the boundary condition imposed at the left boundary x_1, that is

$$\left.\begin{array}{l} d\mathbf{s}/dt|_j = -c\;\delta_x \mathbf{s}|_j \quad j = 2, \ldots, n \\ s_1 = \alpha \end{array}\right\} \,, \qquad c \geq 0 \,. \tag{6.43}$$

We know that $\delta_x \mathbf{s} = (1/2)A_1^{-1}B_1\mathbf{s}$, but we only need a relation involving the last $n-1$ components of the vector $\mathbf{s}$, not all of them. This can be obtained by introducing the following block matrices

[Lomax et al., 1998, Mellado and Ansorge, 2012]

$$A_1 = \begin{pmatrix} a_{11} & \mathbf{a_{12}}^T \\ \mathbf{a_{21}} & A_{22} \end{pmatrix} , \qquad B_1 = \begin{pmatrix} b_{11} & \mathbf{b_{12}}^T \\ \mathbf{b_{21}} & B_{22} \end{pmatrix} .$$

Then, eliminating s'_1 in the original system, yields

$$hA^R_{22} \begin{pmatrix} s'_2 \\ \vdots \\ s'_n \end{pmatrix} = B^R_{22} \begin{pmatrix} s_2 \\ \vdots \\ s_n \end{pmatrix} + s_1 \mathbf{b^R_{21}} , \tag{6.44}$$

where the $(n-1)\times(n-1)$ matrices $\{A^R_{22}\,, B^R_{22}\}$ and the column vector $\mathbf{b^R_{21}} \in \mathbb{R}^{n-1}$ are

$$A^R_{22} = A_{22} - \frac{1}{a_{11}} \mathbf{a_{21}} \mathbf{a_{12}}^T , \qquad B^R_{22} = B_{22} - \frac{1}{a_{11}} \mathbf{a_{21}} \mathbf{b_{12}}^T , \qquad \mathbf{b^R_{21}} = \mathbf{b_{21}} - \frac{b_{11}}{a_{11}} \mathbf{a_{21}} . \tag{6.45}$$

Note that A^R_{22} and B^R_{22} have the same bandwidths as A and B, respectively. The element s'_1 can be calculated by

$$s'_1 = \frac{1}{h a_{11}} \begin{pmatrix} b_{11} & \mathbf{b_{12}}^T \end{pmatrix} \mathbf{s} - \frac{1}{a_{11}} \mathbf{a_{12}}^T \begin{pmatrix} s'_2 \\ \vdots \\ s'_n \end{pmatrix} . \tag{6.46}$$

Then, the original equation can be written as

$$\frac{d}{dt} \begin{pmatrix} s_2 \\ \vdots \\ s_n \end{pmatrix} = -(c/h)(A^R_{22})^{-1} B^R_{22} \begin{pmatrix} s_2 \\ \vdots \\ s_n \end{pmatrix} + \alpha \mathbf{b^R_{21}} , \tag{6.47}$$

so that the set of complex numbers $-(c\tau/h)\mathrm{eig}\{(A^R_{22})^{-1} B^R_{22}\}$ have to fall inside the stability region. [Carpenter et al., 1993].

The same can be done for the diffusion operator. In this case, we impose 2 boundary conditions. If one of the boundary conditions is imposed at $x = x_n$, then we need to eliminate s_n as we have eliminated before s_1 when the boundary condition is imposed at $x = x_1$. We obtain now reduced matrices of size $(n-2)\times(n-2)$. For the compact schemes used in the code, this analysis shows that the CFL conditions derived from the spectral FDM approximations are also applicable to the biased schemes used in non-periodic directions, as observed in figure 6.2.

In case of a nonuniform grid, we need to consider the diagonal matrices D_1 and D_2 in (6.14) and (6.16). The effect of a varying grid spacing can be studied with the script `figure.py`. When we use the minimum of grid spacing in the CFL definitions, part of the eigenvalues approach the origin from the stable side.

For the multidimensional case, we need to consider the sum of all possible combinations of eigenvalues in each direction [Lomax et al., 1998].

6.2.2 Implicit schemes

To be developed. See Spalart et al. [1991].

The dissipation and dispersion error maps corresponding to the third-order implicit Runge-Kutta scheme are shown in figure 6.4. The algorithm is unconditionally stable but we need to control accuracy of the diffusion operator for which it is used. The reference value $\mathrm{CFL}_d = 1.7$ as it gets most of the eigenvalues within the 1%-error region.

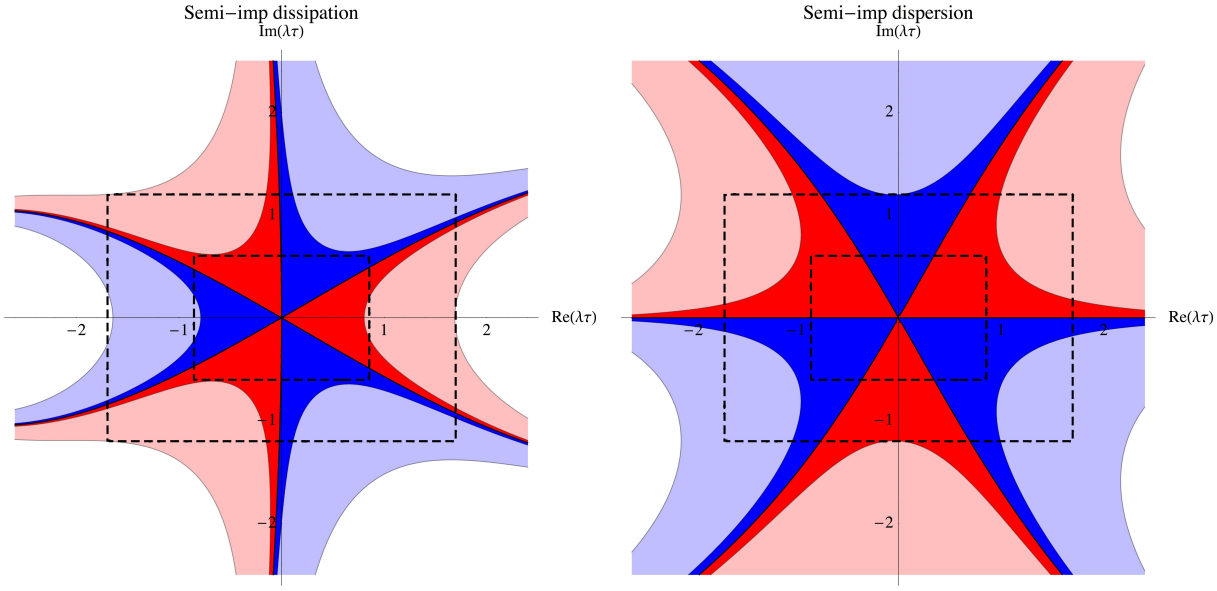

Figure 6.4: Dissipation error (left) associated with the third-order implicit Runge-Kutta scheme: dark blue, $0.99 < \rho < 1$; light blue, $0.90 < \rho < 0.99$; dark red, $1 < \rho < 1.01$; light red, $1.01 < \rho < 1.10$. Dispersion error (right) associated with the Runge-Kutta scheme: dark blue, $-0.01 < \theta/\pi < 0$; light blue, $-0.10 < \theta/\pi < -0.01$; dark red, $0 < \theta/\pi < 0.01$; light red, $0.01 < \theta/\pi < 0.10$.

6.3 Particle algorithms

See directory `particles`.

The algorithm assumes that the grid spacing in the first and third directions are uniform, which is used to locate the grid nodes to interpolate the Eulerian fields into the particle position, and to extrapolate particle information into the surrounding grid points. Stretching is still allowed in the second direction.

6.3.1 Interpolations

To be done.

7 Memory management

See file `modules/tlab_arrays` and `modules/tlab_procs` for their allocation.

Major arrays are allocated during initialization and not anymore later. These arrays are indicated in the following table. Intermediate variables are to be stored in `txt` and not in scratch arrays. Scratch arrays can always be used in low level procedures, and it is better not to use them in high level procedures (above `operators`).

array	size	content
`x`	number of points in Ox × number of arrays for numerics	Ox-coordinate information
Same for Oy and Oz		
`q`	number of points × number of flow fields	flow variables
`s`	number of points × number of scalar fields	scalar variables
`txc`	extended number of points × number of temporary fields	temporary variables
`wrk3d`	extended number of points	scratch
`wrk2d`	maximum number of points in 2D planes × number of scratch planes	scratch
`wrk1d`	maximum number of points in 1D lines × number of scratch lines	scratch

Table 7.1: Major arrays indicating their sizes in terms of the number of points.

Pointers are also defined during initialization to access these memory spaces with arrays of different shape and even different type, more specifically, complex type needed in Fourier decomposition. See corresponding procedures in `modules/tlab_procs`.

8 Parallelization

8.1 Domain decomposition

The domain decomposition is performed along the first and last indexes, typically i and k, respectively, that is, along directions Ox and Oz. Initially, the code only supported 1D decomposition along Oz, the outer-most index. The reason to chose that direction was to simplify I/O and to maintain homogeneity in the serial part of the algorithm (the largest part) for the cases with periodicity along that direction (boundary conditions where only needed in the other two directions, for instance, in a spatially evolving flow like a jet). When the domain decomposition was extended to a second direction, we chose Ox, the reason being again to keep the algorithm equal in every task in the cases where homogeneity and periodicity apply along those two directions. Figure 8.1 sketches this 2D decomposition and summarizes part of the main code variables. We will use the term MPI task or simply task (instead of processor, node, core, ...) – that is, we decompose the problem into `ims_npro_i`×`ims_npro_k` tasks. The mapping is established at read/write time and details follow below. For each task, each array can be interpreted as `jmax`×`kmax` lines of size `imax`, as illustrated in figure 8.1.

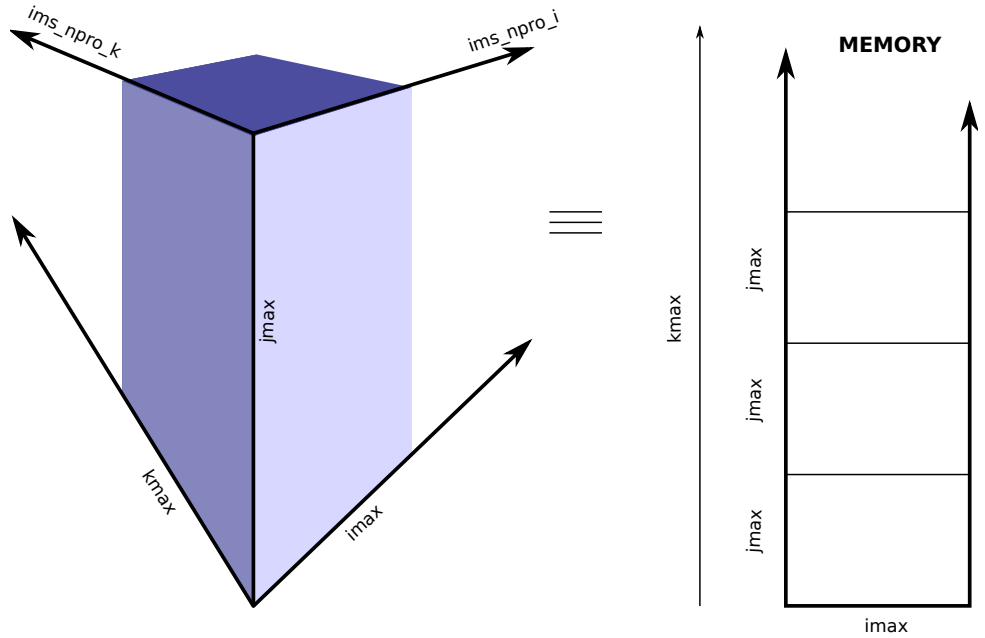

Figure 8.1: Domain decomposition of the global array of size `imax_total`×`jmax_total`×`kmax_total` into the the local arrays of size `imax`×`jmax`×`kmax` using `ims_npro_i` MPI tasks along the first (inner-most) index and `ims_npro_k` along the last (outer-most) index. The structure in memory is shown in a two-dimensional array where the inner-most index runs up to `imax` and the outer-most index runs up to `jmax`×`kmax`; it can also be interpreted as `kmax` pages of size `imax`×`jmax`.

Two main transpositions are needed to perform the derivatives or any other implicit operation in which we only need a set of complete lines along the desired direction contiguously in memory. This is represented in figure 8.2. For instance, if we need a derivative along Ox of the field in array a,

we can interpret the algorithm as follows. Consider that array as jmax×kmax lines of size imax, as illustrated before in figure 8.1. Divide jmax×kmax, the outer index, by ims_npro_i, so as to have precisely ims_npro_i blocks (or colors) of size imax times whatever number you got before. Each of those blocks is send to the corresponding processor. This operation is masked by creating an appropriate MPI type, which is done during the initialization of the MPI part of the code. The constraint we impose is that the ratio jmax×kmax/ims_npro_i needs to be an integer – take this into account when defining the grid. (This constraint could be avoided using padding, but we do not do it in these main transposition operations.)

Let us consider now an implicit operation along Oz. For this case, each of the pages imax×jmax needs to be divided by the number of tasks ims_npro_k, and this ratio is what needs to be an integer. It is also seen in figure 8.2 that now we need a stride in the MPI type. The rest of the transposition algorithm is similar to the previous case. The code variables containing the corresponding MPI types for the transposition operations described in the previous and this paragraphs are DNS_MPI_I_PARTIAL and DNS_MPI_K_PARTIAL, respectively.

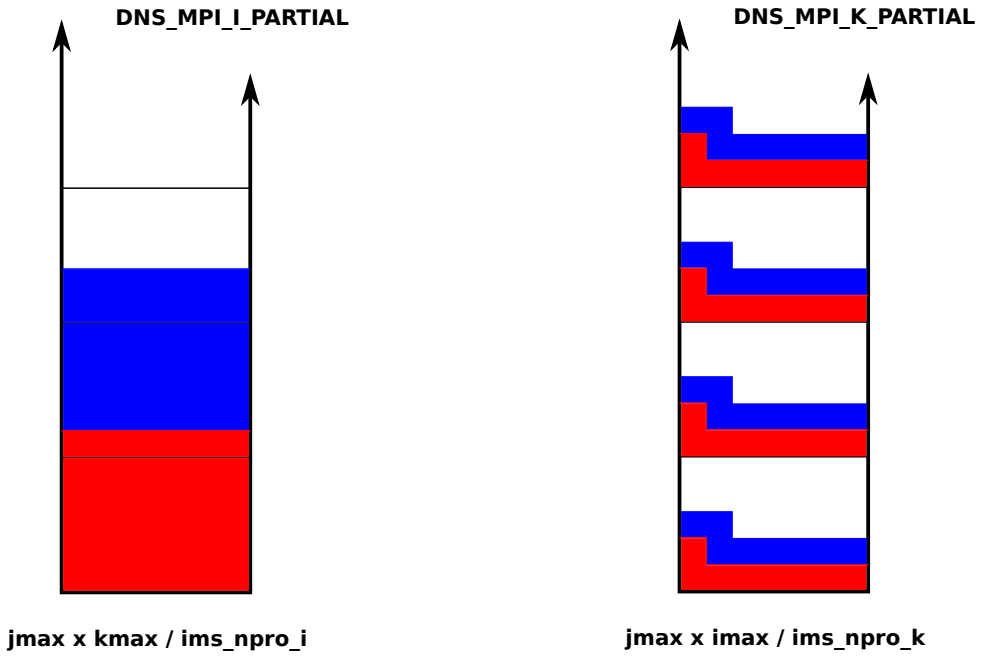

Figure 8.2: Memory management in transposition operations, from sketch in figure 8.1. Each color indicates the block of memory that goes into a common task. Based on these graphs, the offsets, strides and sizes in the MPI derived types are defined. The ratios at the bottom need to be an integer. You have as many colors as tasks involved in the corresponding transposition.

The I/O is done using MPI_IO library. We read

$$\texttt{ims\_npro} = \texttt{ims\_npro\_i} \times \texttt{ims\_npro\_k}$$

contiguous blocks of contiguous data, each block into one task. The corresponding state is precisely equal to that obtained after the PARTIAL_I transposition, and so the only additional thing we need to do is an inverse transposition of that type, and we already have the structure described in figure 8.1. By this procedure we also defined the mapping, which is sketched in figure 8.3. From that mapping we see that a task ims_pro contains the block given by

$$\texttt{ims\_pro\_i} = \texttt{MOD}(\texttt{ims\_pro}, \texttt{ims\_npro\_i})$$
$$\texttt{ims\_pro\_k} = \texttt{INT}(\texttt{ims\_pro}, \texttt{ims\_npro\_i})$$

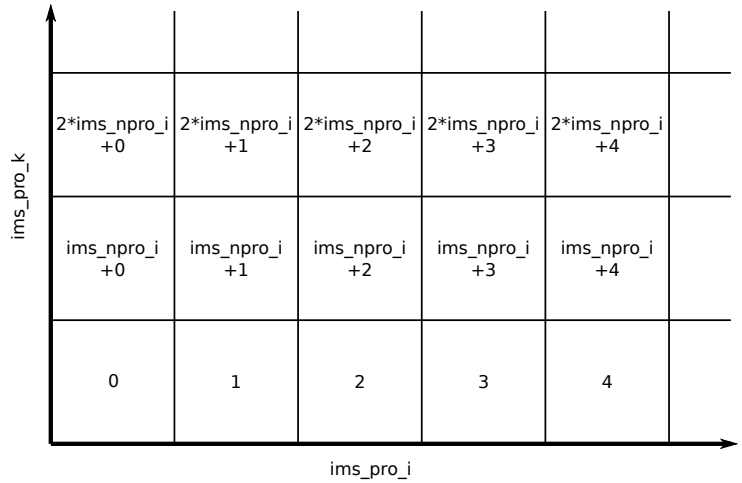

Figure 8.3: Mapping.

When needed, the information defining the boundary conditions is read from disk using a similar procedure. we just need to define new MPI types for the transposition along Ox according to the specific size of the corresponding arrays. This is done inside the I/O routines, as appropriate.

For the transpositions required in the Poisson equation, the procedure is similar to the main algorithms defined above and shown in figures 8.1 and 8.2, although two additional planes in Oy containing the boundary conditions, one at the bottom and one at the top, and one in Ox for the pseudo-Nyquist frequency are added (pseudo meaning that only one should be added, but space is reserved in each processor along that direction to keep the algorithm homogeneous; this could be redefined). For these cases, padding is used inside each pages (defined above) so that the only constraints in the grid are those imposed before in the two major kinds of transposition. Hence, for a transposition along Oz, the same structure as shown in figure 8.2 is used but with a page of size `isize_txc_dimz` instead of `imax`×`jmax`, such that `isize_txc_dimz` is a multiple of 2×`ims_npro_k`, (the factor of 2 for real and imaginary parts of the same complex number to remain in the same processor) and larger than (`imax`+2)×(`jmax`+2).

The same description applies to the transposition along Ox. The only difference here is that a second type is added for the transformation without the Nyquist frequency. This is used for instance for the first forward transposition of data. *More here?*

9 Scaling

For spatial discretization implicit schemes are used. Hence, the calculation of a derivative always involves communication amongst all processors in a line along which a derivative is computed. From ad-hoc considerations it is not clear which is the optimum two-dimensional domain-decomposition. While for small numbers of cores one would expect the network latency of an MPI call to dominate, for larger numbers it is the number of processors involved in the call that makes MPI calls expensive. Therefore, below a certain threshold, a 1D decomposition is expected to be beneficial. Where this turnover takes place is subject to many factors such as the CPU clock speed, network latency, MPI implementation, number of grid points, etc.

We briefly introduce now definitions and notation used in the rest of this section. We define $Q := q_x \times q_y \times q_z$, the number of grid points, as a measure of the size of the simulations and choose in the following to label simulations by Q. To label the simulations, we use the common abbreviations

$$\mathrm{k} = 2^{10}; \qquad \mathrm{M} = 2^{20}; \qquad \mathrm{G} = 2^{30} \tag{9.1}$$

for readability. The memory necessary to save one 3D array in double precision data format is $8 \times Q$ Byte. The scaling of the code is discussed in terms of speedup S and efficiency η defined as

$$S := \frac{T}{T_{\mathrm{ref}}} \qquad \eta = \frac{T}{NT_{\mathrm{ref}}} \times 100\%, \tag{9.2}$$

where T is the real time to run a particular case, N the number of processors and the subscript 'ref' indicates a reference value.

The code has been instrumented to measure the real time that is necessary to perform one stage of the multi-stage Runge-Kutta time-stepping scheme. This corresponds essentially to the evaluation of the right-hand-side term of the governing equations solved. The pre-processor flag `-DUSE_PROFILE` activates it and data is written into the log file `tlab.log`. The aim is to remove the overhead time associated with I/O and initialization.

9.1 Scaling on the cluster `jugene@fz-juelich.de`

Performance of the DNS code has been measured for various geometries varying the total number of grid points by two orders of magnitude. Many-core scaling properties of the DNS code, version 5.6.6 on the machine `jugene@fz-juelich.de` (site Jülich Supercomputing Center) were investigated. All simulations have been run in SMP mode using 4 OpenMP threads, reaching up to up to 8k MPI tasks distributed over 8k nodes (1 rack contains 1k nodes), using 4 cores per node (i.e. 32k cores in total). Linear scaling is observed for up to 4096 MPI tasks using about 6-8 M grid points per processor (domains of 24-32 G grid points). The maximum efficiency is usually reached when simulations are carried out within one mid-plane. In this case, a slightly super-linear scaling (with respect to the reference at 32 nodes) is observed for the standard domain sizes of up to 4 G grid points.

The code has been run for 2 iterations, i.e. 10 Runge-Kutta stages using the fourth-order, five-stages algorithm, and the variance of measured times was found to be of the order of 1%. The cases considered here are listed in Table 9.1.

Name	q_x	q_y	q_z	Q
1024x0384	1024	1024	384	384 M
2048x0192	2048	2048	192	768 M
2048x1024	2048	2048	1024	4 G
3072x1536	3072	3072	1536	12 G
4096x1536	4096	4096	1536	24 G
4096x2048	4096	4096	2048	32 G

Table 9.1: Geometry and labels of the 3D cases.

9.1.1 Strong Scaling

The total number of MPI tasks (nodes) available to a simulation are distributed as $N =$`ims_npro` $=$ `ims_npro_k` $\times$ `ims_npro_i` in the directions of k (z) and i (x). A series of measurements has been carried out to determine the optimum configuration for the six cases listed in table 9.1. Scaling matrices are shown in Figures 9.1 and 9.2 for the different cases. In these matrices the number of processors is constant along diagonals from the lower left to the upper right. In every matrix, the diagonal for $N = 1k$ MPI tasks is outlined by solid borders. In each of these diagonals, the best and worst configurations are marked by green, respectively red, color. In these matrices, the speed-up and efficiency is for strong scaling, and always calculated with respect to the time T_{ref} for the lowest number of cores with the lowest `ims_npro_i`. Efficiency η and speed-up S are calculated as above. The shapes used in the simulations, that is, the relative connection among mid-planes, is $1 \times 1 \times 1$, $2 \times 1 \times 1$, $2 \times 2 \times 1$, $2 \times 2 \times 2$, $4 \times 2 \times 2$, ordered from 1 mid-plane (512 nodes) to 16 mid-planes (8192 nodes).

9.1.2 Scaling from 32 to 8192 nodes

A strong scaling analysis over the entire range of nodes from 32 to 8192 is not possible. Hence, we restrict ourselves to a scaling analysis where we consider the number of grid points that is handled per processor and time unit. A straightforward definition for a metric of performance P is

$$P := \frac{Q}{NT} \,, \tag{9.3}$$

the number of grid points processed per node and per time. T is the time needed by each of the cases to advance exactly the same amount of instructions in the main algorithm, e.g. one stage of the Runge-Kutta scheme. P is a metric that makes performance comparable over *almost* arbitrary problem sizes and numbers of cores.

Note, that the caveat here is the operation count for the Fourier transforms which goes as $q_i \log q_i$ and $q_i \approx Q^{1/3}$ if domains are expanded by the same factor in each direction. Hence, for the overall operation count of the Fourier transforms Σ_{FFT} we get

$$\frac{\Sigma_{\text{FFT}}}{q_i^2} \propto 3 q_i \log q_i = Q^{1/3} \log Q \Rightarrow \Sigma_{\text{FFT}} \propto Q \log Q. \tag{9.4}$$

For the range of Q considered here, this effect is, however, small since the super-linear contribution of Σ_{FFT} is only $\frac{\log 32\text{G}}{\log 384\text{M}} \approx 1.2$ and the FFT accounts for a negligible part of the computational time

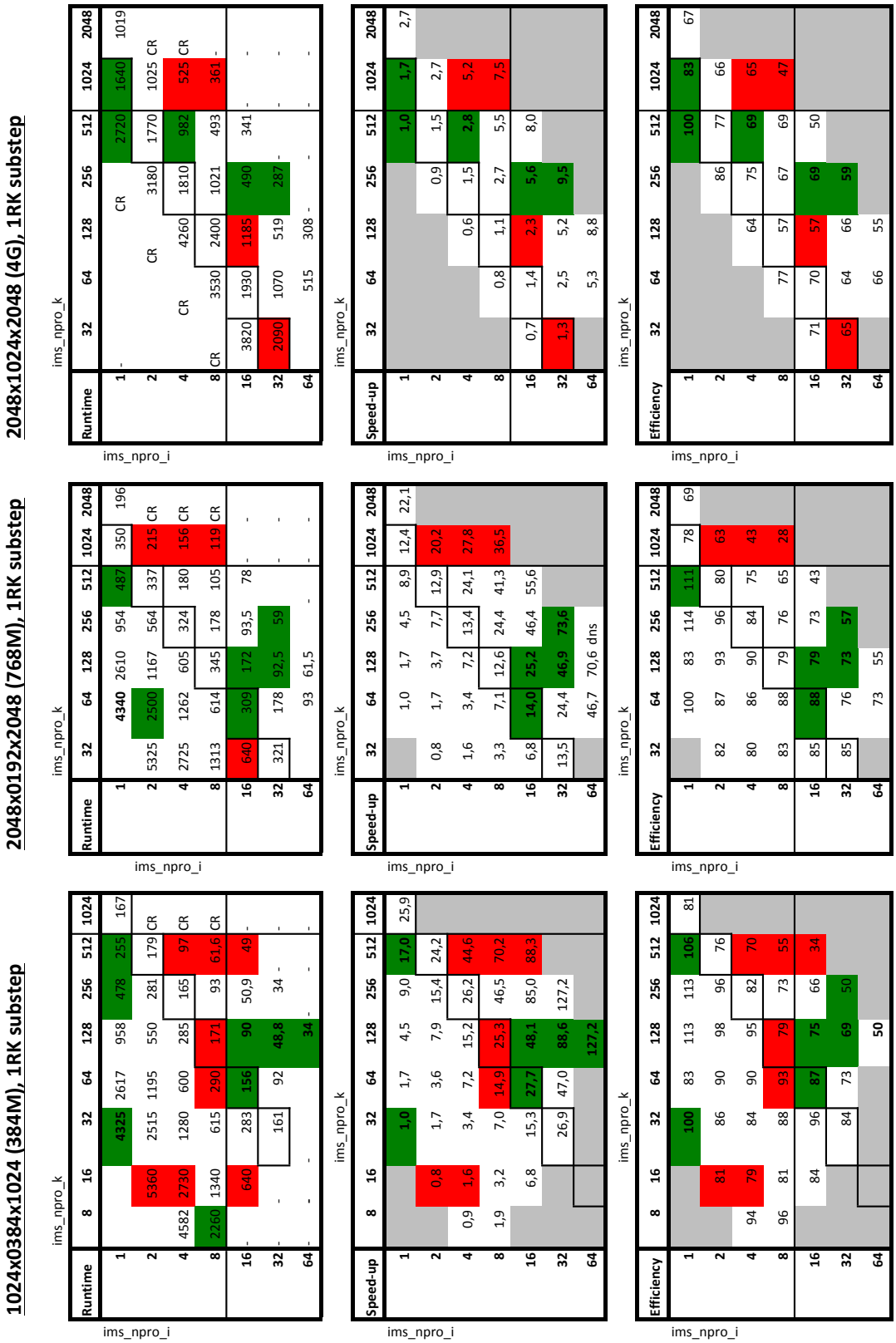

Figure 9.1: Matrices for scaling and 2D domain decomposition. Time is in hundredth of a second per single Runge-Kutta stage. In the time-matrices, simulations that crashed because of too little memory (above diagonals) and page problems (below diagonals) are marked by CR. If no measurement for a certain configuration was attempted, the cell is left empty. For speed-up and efficiency all configurations not available are marked gray.

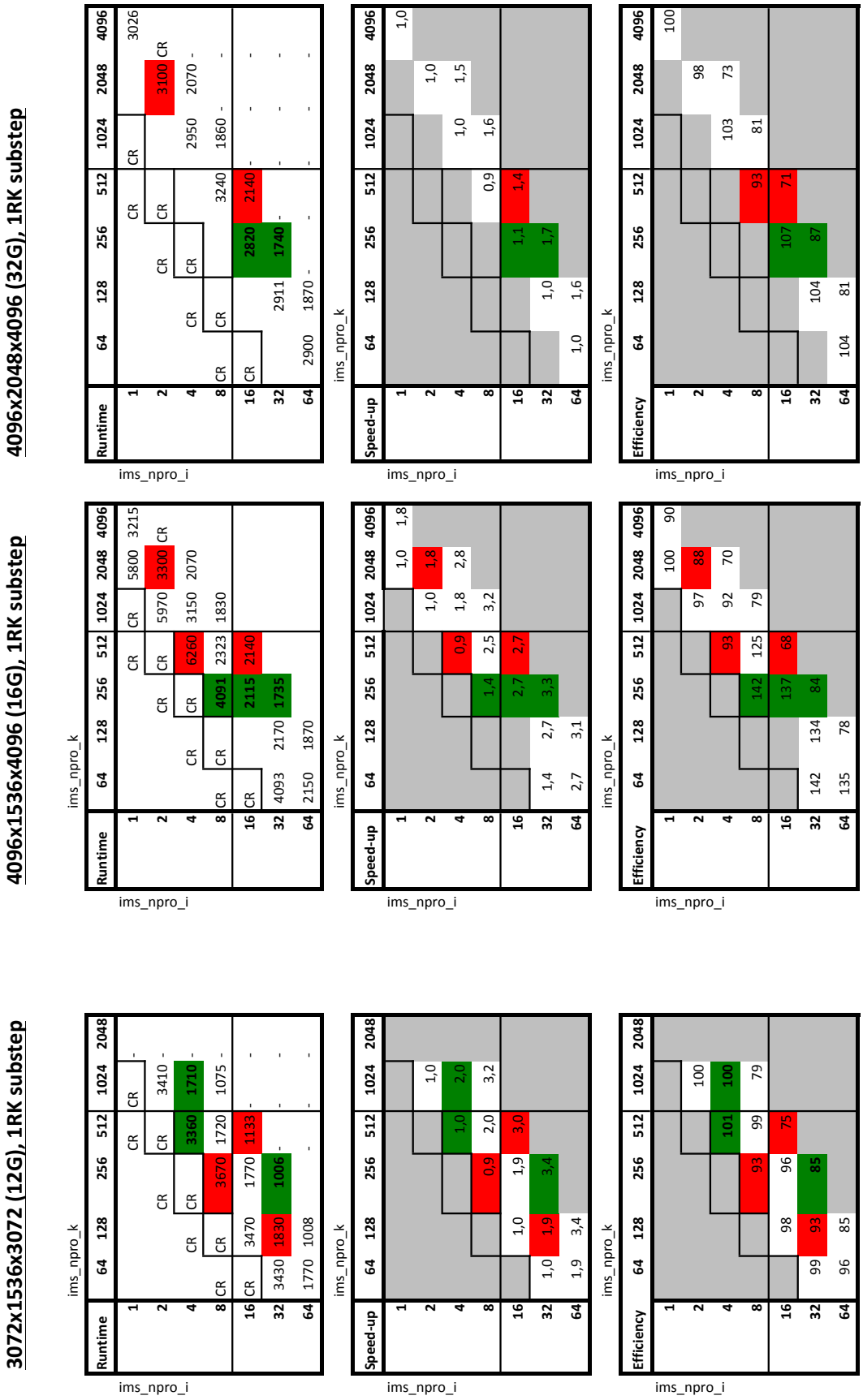

Figure 9.2: More cases. Same legend as in Figure 9.1.

T

N [k] \ Q [M]	384	768	4.096	13.824	24.576	32.768
1/32	4.325					
1/16	2260	4340				
1/8	1340	2500				
1/4	478	954				
1/2	270	487	2720			
1/1	156	309	1640			
2/1	90	172	982	3410		
4/1	48,8	93	490	1710	2115	2820
8/1	34	59	287	1006	1400	1740

Efficiency w.r.t 32 cores of smallest case

N [k] \ Q [M]	384	768	4.096	13.824	24.576	32.768
1/32	100%					
1/16	96%	100%				
1/8	81%	87%				
1/4	113%	113%				
1/2	100%	111%	106%			
1/1	87%	87%	88%			
2/1	75%	79%	73%	71%		
4/1	69%	73%	74%	71%	102%	102%
8/1	50%	57%	63%	60%	77%	83%

S (Speed-up)

N [k] \ Q [M]	384	768	4.096		24.576	32.768
1/32	1,0					
1/16	1,9	2,0				
1/8	3,2	3,5				
1/4	9,0	9,1				
1/2	16,0	17,8	17,0			
1/1	27,7	28,0	28,1			
2/1	48,1	50,3	47,0	45,7		
4/1	88,6	93,0	94,1	91,1	130,9	130,9
8/1	127,2	146,6	160,7	154,8	197,7	212,1

P (Megapoints per k-procs per 0.01 seconds)

N [k] \ Q [M]	384	768	4.096	13.824	24.576	32.768
1/32	2,84					
1/16	2,72	2,83				
1/8	2,29	2,46				
1/4	3,21	3,22				
1/2	2,84	3,15	3,01			
1/1	2,46	2,49	2,50			
2/1	2,13	2,23	2,09	2,03		
4/1	1,97	2,06	2,09	2,02	2,90	2,90
8/1	1,41	1,63	1,78	1,72	2,19	2,35

Figure 9.3: Scaling results in term of Speed-Up S_W, Efficiency E_W and Performance P as defined above.

($1\% - 5\%$ of the computational part, which is 0.5%-2.5% of the overall time). The operation count of the rest of the algorithms is linear.

Given P, one can compute a virtual efficiency and speed-up η_v and S_v as

$$\eta_v := \frac{P}{P_{\text{ref}}}; \qquad S_v := \frac{N}{N_{\text{ref}}} \frac{P}{P_{\text{ref}}} \tag{9.5}$$

with respect to a reference. Here, we use the same reference for all simulations and cases, namely, the 32 x 1 decomposition of the case with $Q = 384$ M. For larger numbers of cores N and simulation sizes Q we always choose the optimum configuration which is marked by green color in Figures 9.1 and 9.2.

The scaling as described in the above paragraph is summarized in the tables shown in Figure 9.3. The first table contains the real time needed for one Runge-Kutta stage, measure in hundredths of a second. As already said before, these data is simply collected from the matrices on Figures 9.1 and 9.2. The other 3 tables in Figure 9.3 contain the corresponding values of P, η_v and S_v. Figure 9.4 shows the speed-up S_v. In the upper panel there is one line for each cases. Note, that the definition of S_v does **not imply that cases start on the linear scaling line**. In this case, it is part of the measurements and simply means that $\eta_v \approx 100\%$ or $P \approx P_{\text{ref}}$.

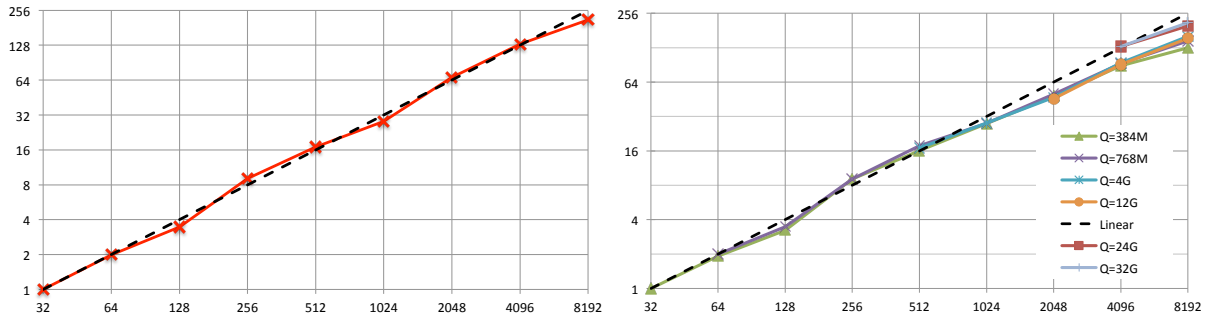

Figure 9.4: Upper panel: S_W versus number of nodes for all geometries. Lower panel: maximum speedup for a given number of processors $\max(S_W)|_N$; axes as in upper panel.

MISSING FIGURE

Figure 9.5: Strong scaling on juwels (solid lines); dashed lines indicated weak scaling.

9.2 Scaling on the cluster `juwels@fz-juelich.de`

With two sockets each holding 24 CPU cores, juwels features 48 physical cores per node and thus introduces a single prime factor three to the number CPUs available. To optimally utilize this, domain sizes should also contain a prime factor three.

The cluster is heterogeneous and features

(1) a partition `batch` with 2271 standard-memory nodes (2x24 cores, 96 GB)

(2) a partition `mem192` with 240 enhanced-memory nodes (2x24 cores, 192 GB)

(3) a booster module with 56 GPU-accelerated nodes (2x20 cores + 4 GPUs, 192 GB; to be extended in 2020).

Scaling was tested using up to 512 nodes on the `batch` partition (Tab. 9.2 and Fig. 9.5 and up to 64 nodes of the `mem192` partition (Tab. 9.3). There appears to be a significnat memory-overhead of the underlying MPI which sometimes enables to run particular cases in much less than half the number of large-memory nodes as compared to running the same configuration in the `batch partition`.

In comparison with juwels, the normaliezd performance (CPU-h per 1024^3 points per iteration) is better by up to a factor of 3, but only for relatively small cases that can be run within a few (up to 4) nodes. For operational cases (using 32 to 128 nodes), the performance boost is around 2. For even larger cases utilizing 256 or more nodes, performance boost in comparison with juwels is smaller than 2, and it approaches one from above for jobs using 10000 or more MPI tasks.

9.3 Scaling on the cluster `blizzard@dkrz.de`

To be done.

Table 9.2: Scaling of two cubed cases in the batch partition of `juwels@fz-juelich.de`; only computation no initialization, no I/O

Case	1536^3			3072^3		
#nodes	time/it $[s]$	Speed-up	Eff.	Time/it $[s]$	Speed-up	Eff.
8	113.5	1.0	1.00			
16	57.5	2.0	0.99			
32	36.3	3.1	0.78			
64	21.2	5.4	0.67	163.0	1.0	1.00
96	15.1	7.5	0.63	98.4	1.7	1.10
128	11.5	9.9	0.62	92.8	1.8	0.88
192	9.1	12.5	0.52	65.1	2.5	0.83
256	6.7	16.9	0.53	51.4	3.2	0.79
384				38.1	4.3	0.71
512				29.3	5.6	0.70

Table 9.3: Average timing for total production jobs in the large-memory partition as compared to the batch partition. The batch case using 128 nodes is the smallest configuration for which the case runs in the batch queue.

Partition		mem192				batch
#nodes	[1]	20	24	32	64	128
#tasks per node	[1]	16	48	48	48	48
Wall-clock time/it	[s]	422	286	230	115	76
Memory used / node	[GB]	187	160	124	81	52
total memory	[TB]	3.65	3.75	3.88	5.06	6.5
node-h / it	[h]	2.34	1.91	2.04	2.05	2.70
Speed-up w.r.t. batch	[%]	11	**31**	22	22	-

10 Profiling

We include here results from profiling to identify where to put the effort to further optimize the code. The diagrams shown below have been constructed with `gprof2dot.py`[1].

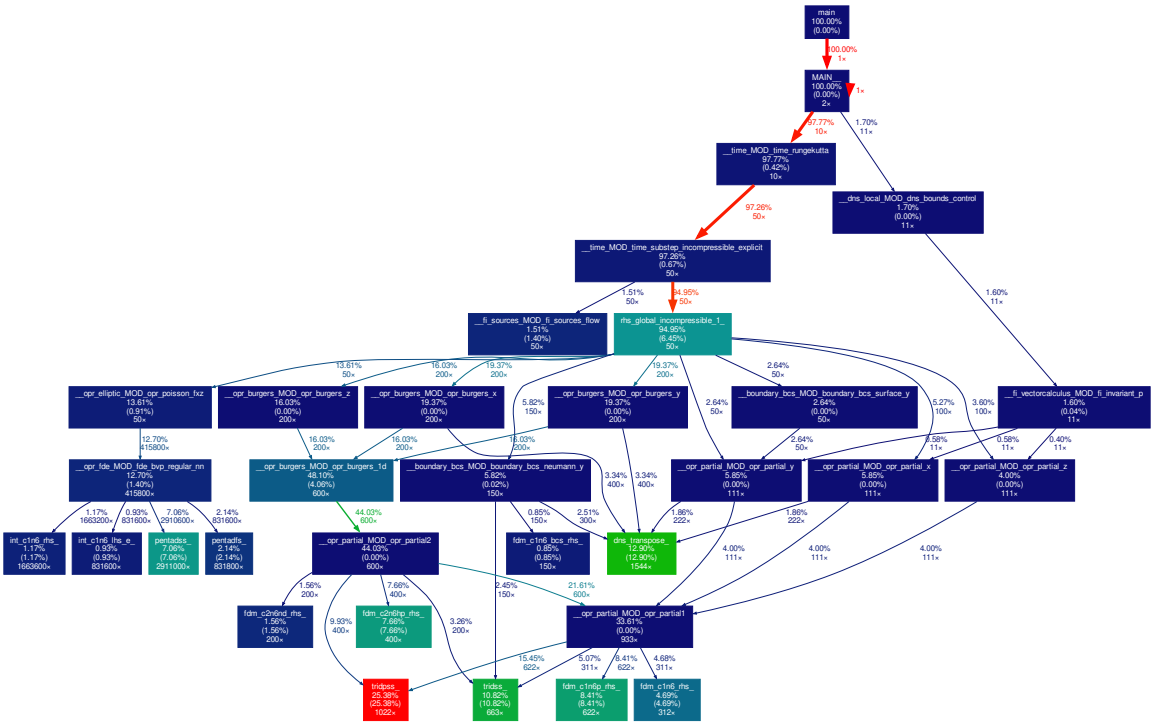

Figure 10.1: Profiling diagram of `examples/Case44` running 10 iterations in serial mode. Profiling data obtained from `gfortran -pg` and processed with `gprof`, running the command `gprof path/to/your/executable | gprof2dot --color-nodes-by-selftime | dot -Tpdf -o output.pdf`.

[1] `https://github.com/jrfonseca/gprof2dot`

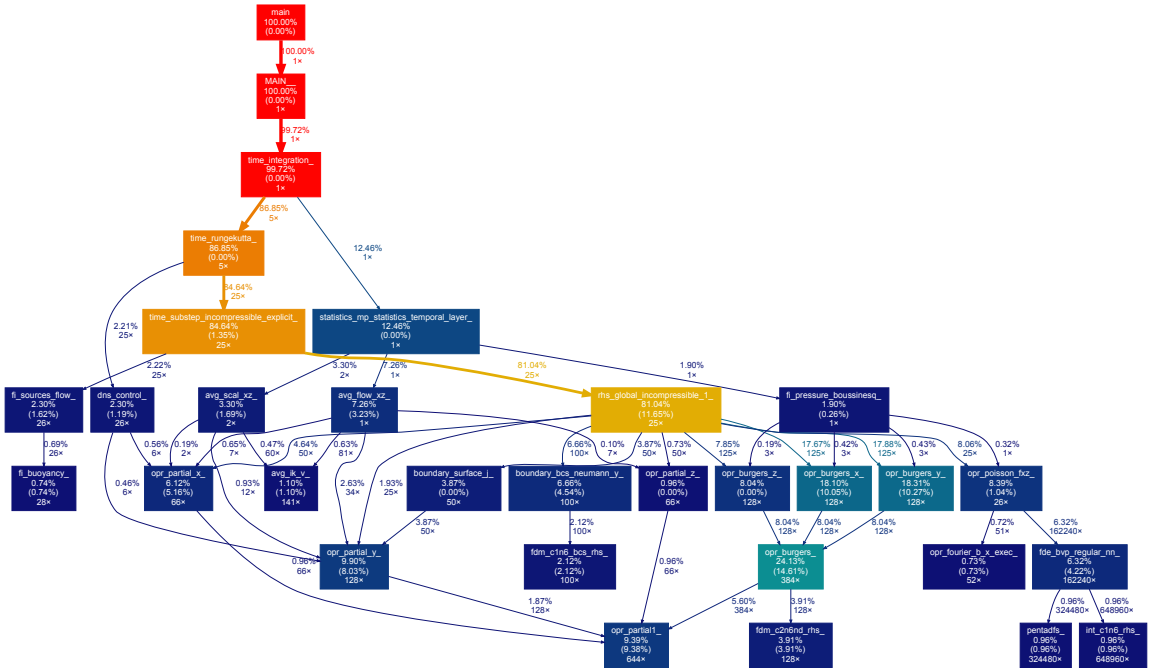

Figure 10.2: Profiling diagram of modified `examples/Case44` (768 cube) running 25 iterations in parallel mode with 48 tasks (1 node on juwels).

Bibliography

G. A. Blaisdell, E. T. Spyropoulos, and J. H. Qin. The effect of the formulation of nonlinear terms on aliasing errors in spectral methods. *App. Num. Math.*, 21:207–219, 1996.

M. H. Carpenter and C. A. Kennedy. Fourth-order 2N-storage Runge-Kutta schemes. Technical Report TM-109112, NASA Langley Research Center, 1994.

M. H. Carpenter, D. Gottlieb, and S. Abarbanel. The stability of numerical boundary treatments for compact high-order finite-difference schemes. *J. Comput. Phys.*, 108:272–295, 1993.

G. Erlebacher, M. Y. Hussaini, H. O. Kreiss, and S. Sarkar. The analysis and simulation of compressible turbulence. *Theor. Comput. Fluid Dynamics*, 2:73–95, 1990.

C. Foias, D. D. Holm, and E. S. Titi. The Navier-Stokes-alpha model of fluid turbulence. *Physica D*, 152-153:505–519, 2001.

M. Frigo and S. G. Johnson. The design and implementation of FFTW3. In *Proceedings of the IEEE, vol. 93*, pages 216–231, 2005.

F. J. Higuera and R. D. Moser. Effect of chemical heat release in a temporally evolving mixing layer. *CTR Report*, pages 19–40, 1994.

F. Q. Hu. On absorbing boundary conditions for linearized Euler equations by a perfectly matched layer. *J. Comput. Phys.*, 129:201–219, 1996.

F. Q. Hu, M. Y. Hussaini, and J. L.Manthey. Low-dissipation and low-dispersion Runge-Kutta schemes for computational acoustics. *J. Comput. Phys.*, 124:177–191, 1996.

A. G. Kravchenko and P. Moin. On the effect of numerical errors in large-eddy simulations of turbulent flows. *J. Comput. Phys.*, 131:310–322, 1997.

E. Lamballais, V. Fortuné, and S. Laizet. Straightforward high-order numerical dissipation via the viscous term for direct and large eddy simulation. *J. Comput. Phys.*, 230:3270–3275, 2011.

S. K. Lele. Compact finite difference schemes with spectral-like resolution. *J. Comput. Phys.*, 103:16–42, 1992.

G. Lodato, P. Domingo, and L. Vervisch. Three-dimensional boundary conditions for direct and large eddy simulation of compressible viscous flows. *J. Comput. Phys.*, 227:5105–5143, 2008.

H. Lomax, T. H. Pulliam, and D. W. Zingg. *Fundamentals of Computational Fluid Dynamics*. Springer, 1998.

J. P. Mellado and C. Ansorge. Factorization of the Fourier transform of the pressure-Poisson equation using finite differences in colocated grids. *Z. Angew. Math. Mech.*, 92:380–392, 2012.

R. K. Shukla and X. Zhong. Derivation of high-order compact finite difference schemes for non-uniform grid using polynomial interpolation. *J. Comput. Phys.*, 204:404–429, 2005.

P. R. Spalart, R. D. Moser, and M. M. Rogers. Spectral methods for the Navier-Stokes equations with one infinite and two periodic directions. *J. Comput. Phys.*, 96:297–324, 1991.

K. W. Thompson. Time-dependent boundary conditions for hyperbolic systems. *J. Comput. Phys.*, 68:1–24, 1987.

K. W. Thompson. Time-dependent boundary conditions for hyperbolic systems, II. *J. Comput. Phys.*, 89:439–461, 1990.

F. A. Williams. *Combustion Theory*. Addison Wesley, second edition, 1985.

J. H. Williamson. Low-storage Runge-Kutta schemes. *J. Comput. Phys.*, 35:48–56, 1980.

R. V. Wilson, A. O. Demuren, and M. Carpenter. Higher-order compact schemes for numerical simulation of incompressible flows. Technical Report CR-1998-206922, NASA Langley Research Center, 1998.