INCOMPACT3D USER GUIDE version 2.0

Sylvain Laizet (Imperial College London)

Contents

1 Overview of Incompact3d

Incompact3d is a powerful high order flow solver for academic research. It combines the versatility of industrial codes with the accuracy of spectral codes. Thank to a successful software development project with NAG and HECToR (UK's old national supercomputing facility), **Incompact3d** can be used on up to one million computational cores to solve the incompressible Navier-Stokes equations. This high level of parallelisation is achieved thanks to a highly scalable 2D decomposition library and a distributed Fast Fourier Transform (FFT) interface. This library is available at http://www.2decomp.org and can be freely used for your own solver.

Incompact3d is based on a Cartesian mesh. The use of such a simplified mesh offers the opportunity to implement high-order compact schemes for the spatial discretization whilst a customized Immersed Boundary Method (IBM) allows the implementation of any solid wall/bluff body geometry inside the computational domain. The main originality of the solver is that the Poisson equation (to ensure the incompressibility) is fully solved in the spectral space via the modified wave number formalism, no matter what the boundary conditions are (periodic, free-slip, no-slip, inflow/outflow, etc.). Note finally that the pressure mesh is staggered from the velocity one by half a mesh to avoid spurious pressure oscillations that can be introduced by the IBM.

More information about the numerical methods can be found in:

- Laizet S.& Lamballais E., **High-order compact schemes for incompressible flows: a simple and efficient method with the quasi-spectral accuracy**, J. Comp. Phys., vol 228-15, pp 5989-6015, 2009
- Lamballais E., Fortune V. & Laizet S., **Straightforward high-order numerical dissipation via the viscous term for Direct and Large Eddy Simulation**, J. Comp. Phys., Vol 230-9, pp 3270-3275, 2011

More information about the parallel strategy of the code can be found in:

- Laizet S.& Li N., **Incompact3d, a powerful tool to tackle turbulence problems with up to $O(10^5)$ computational cores**, Int. J. of Numerical Methods in Fluids, Vol 67-11, pp 1735-1757, 2011
- Li N. & Laizet S., **2DECOMP&FFT - a highly scalable 2D decomposition library and FFT interface**, Cray User Group meeting: Simulation comes of age Edinburgh, Scotland -- 24/05-27/05, 2010
- Laizet S., Lamballais E. & Vassilicos J.C., **A numerical strategy to combine high-order schemes, complex geometry and parallel computing for high resolution DNS of fractal generated turbulence**, Computers & Fluids, vol 39-3, pp 471-484, 2010

IMPORTANT:

- **It is strongly recommended to read the previous references before starting using Incompact3d.**
- **We kindly ask you to cite the previous references (when suitable) in your work based on Incompact3d.**

2 Installation

When starting to use **Incompact3d** on your local workstation, it is recommended to install **gfortran** and **openmpi** (preferably with an **Ubuntu** distribution). It is possible to download the latest archive for **openmpi** from the address http://www.open-mpi.org/software/ompi/v1.10/. See below for the installation of **gfortran** and **openmpi**:

```
sudo apt-get install build-essential
sudo apt-get install gfortran
tar -xvzf openmpi-***.tar.gz
cd openmpi***
sudo ./configure  --prefix=/usr/local F77=gfortran FC=gfortran
sudo make all install
```

It is also necessary to add a line in the `.bashrc` file:

```
export LD_LIBRARY_PATH=/usr/local/lib/
```

2.1 Compiling incompact3d

A Makefile is available and the only thing to do is to select the relevant compiler. For the combination **gfortran+openmpi**, it is:

```
FC = mpif90
OPTFC = -O3 -funroll-loops -ftree-vectorize -fcray-pointer -cpp
CC = mpicc
CFLAGS = -O3
```

To compile the code, you just need to type `make clean` and `make`

2.2 Known issues

Some users have experienced difficulties with the MPI librairy `mpich`. Other than that **Incompact3d** has been used on a wide range of supercomputers with various compilers and MPI librairies.

It is strongly recommended to carefully read the documentation of the supercomputer you will be using. You should be able to find all the information you need about the optimum compiler options.

3 The code

In this section, only the files related to the Navier-Stokes equations are described. The 2D domain decomposition files are derived from the open source 2DECOMP&FFT library and are not explained in this user guide.

3.1 incompact3d.f90

This is the main file containing the time loop relative to the evaluation of the incompressible Navier-Stokes equations.

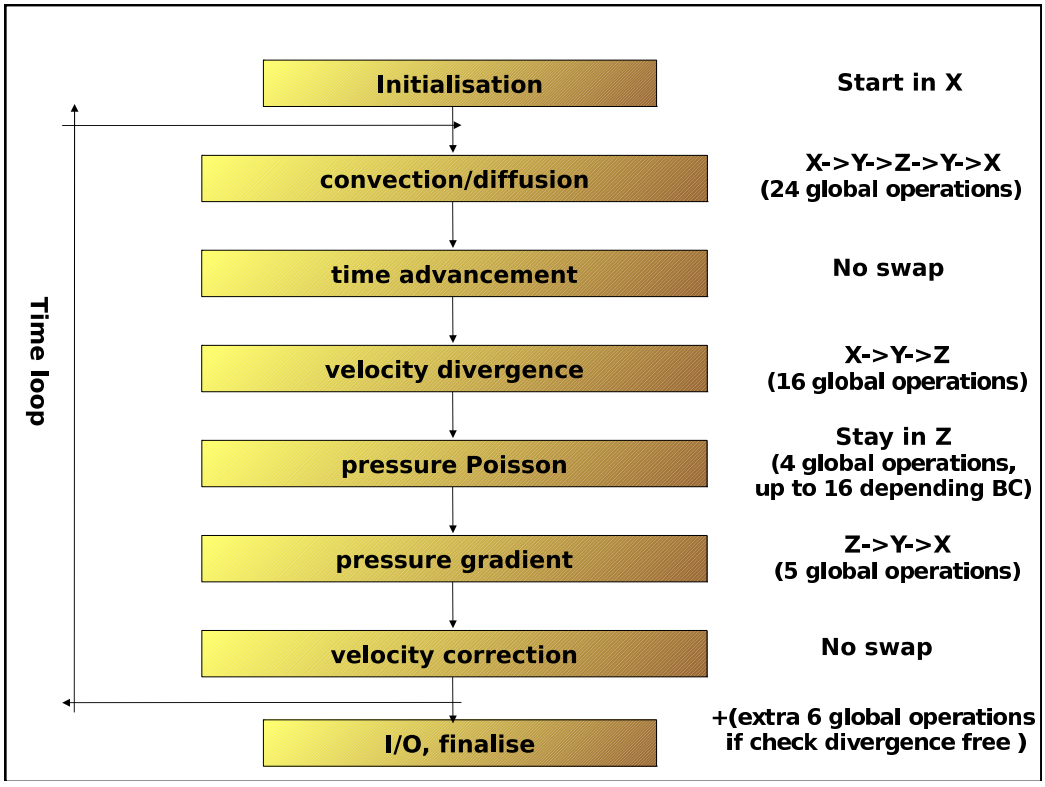

Figure 1: *Stucture of* **Incompact3d** *with the 2D domain decomposition.*

The structure of the solver is presented in figure 1. This figure also shows the management of the pencils swaps. For the Poisson solver in the spectral space, a single division is required and the modified wave numbers combined with the transfer functions are all independent with each other. Note that, when performing the 3D FFT forward, we are in Z-pencils in the physical space, then in X-pencils in the spectral space and finally, in Z-pencils again after the 3D FFT backward, in order to reduce the global transpose operations. When using tri-periodic boundary conditions, 55 global transpose operations need to be performed at each time step. This number can be up to 69 depending on the boundary conditions.

As it can be seen, the code is using various modules (for the constants, 1D and 2D arrays). All the modules can be found in `module_param.f90`.

The variables $t1$ and $t2$ are used to compute the average cost per time step of a simulation.

It is possible to save the statistics and the visualizations on a coarser mesh (every `nstat` and `nvisu` mesh nodes respectively).

```
call init_coarser_mesh_statS(nstat,nstat,nstat,.true.)
call init_coarser_mesh_statV(nvisu,nvisu,nvisu,.true.)
```

The statistics and the visualizations are dealed with in `visu.f90`.

The parameters of the simulations are initialized with

```
call parameter()
call schemes ()
```

Note that because the pressure field is staggered by half a mesh with respect to the velocity field it is necessary to define various mesh sizes (for the interpolations) depending on the boundary conditions of the simulation (variable `phG`).

The following routines are called to initialize the flow field. It is possible to use a checkpoint procedure in order to restart a simulation from a previous one (parameter `ilit`). The restart procedure can be found in `tools.f90` and the restart file is called `sauve.dat`. Note that for some supercomputers (usually the IBM ones) and for very large simulations, you may have to split `sauve.dat` in smaller files as some supercomputers do not deal with very large files.

```
if (ilit==0) call init(ux1,uy1,uz1,ep1,phi1,gx1,gy1,gz1,phis1,hx1,hy1,hz1,phiss1)
if (ilit==1) call restart(ux1,uy1,uz1,ep1,pp3,phi1,gx1,gy1,gz1,&
px1,py1,pz1,phis1,hx1,hy1,hz1,phiss1,phG,0)
call test_speed_min_max(ux1,uy1,uz1)
if (iscalar==1) call test_scalar_min_max(phi1)
```

3.2 incompact3d.prm

The name of each variable is clearly explained in this file. All the quantities are non-dimentionalised with a length and a velocity.

The domain size is $xlx \times yly \times zlz$.

There is no check on the time step so it is your responsability to carefully investigate about the stability conditions for your simulation. The optimum time step has to be defined with a trial-and-error procedure. It is possible to use explicit Adam-Bashforth schemes (recommended when Immersed Boundary Methods are used) or explicite Adam-Bashforth schemes (parameter `ncheme`). Note that in some versions of the code, a semi-implicit strategy has been implemented in order to relax the stability conditions for the time step.

The time step is fixed for a given simulation. It is not possible (yet?) to use a different time step at each time step.

Different boundary conditions can be used in the three spatial direction (`nclx,nclynclz`) as seen in figure 2:

- Periodic conditions corresponding to $ncl = 0$
- Free-slip conditions corresponding to $ncl = 1$
- Open Boundary conditions (Dirichlet conditions for the velocity for no-slip or inflow/outflow conditions), corresponding to $ncl = 2$

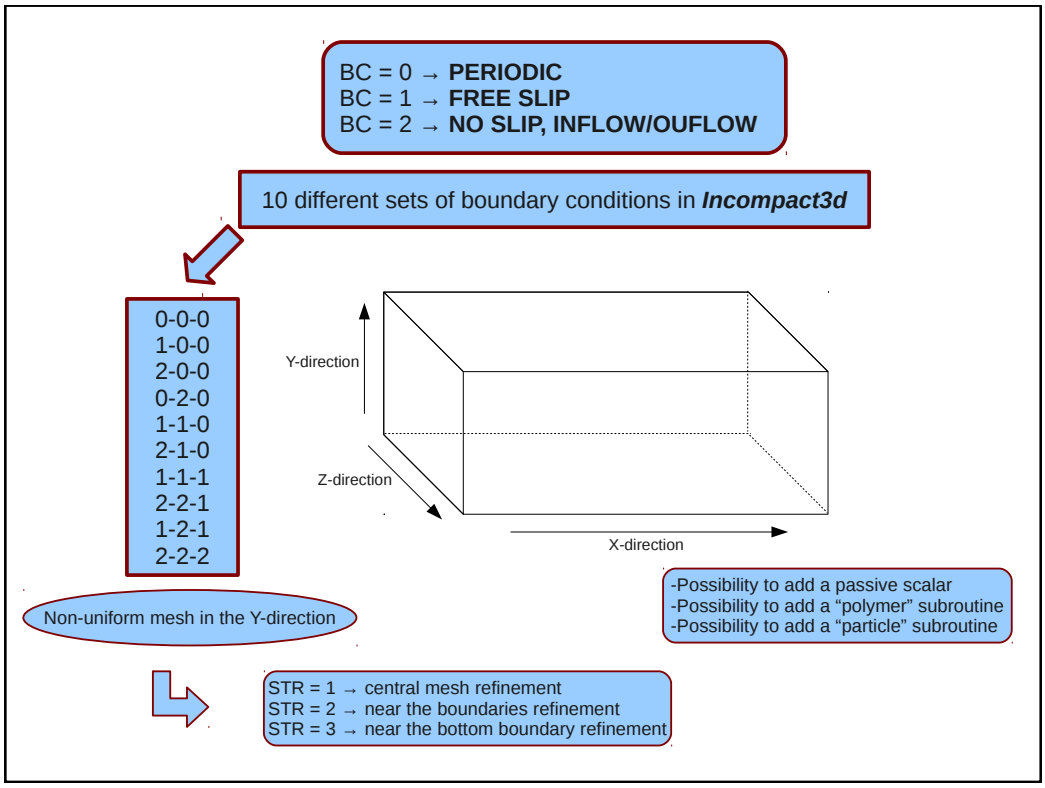

Figure 2: *Description of the different boundary conditions.*

So far and as can been in the figure above, 12 different combinations can be used in the code, covering a wide range of flow configuration: $(0-0-0)$, $(1-0-0)$, $(2-0-0)$, $(0-2-0)$, $(1-1-0)$, $(2-1-0)$, $(1-1-1)$, $(2-2-1)$, $(2-1-2)$, $(1-2-1)$, $(1-1-2)$ and $(2-2-2)$.

For instance the combination for a temporal channel flow is $(0-2-0)$. For a spatially evolving flow, with periodic boundary conditions in the spanwise direction and free-slip boundary conditions in the lateral direction, the combination is $(2-1-0)$. All the combinations have not yet been tested so you may eventually experience some problems.

At the moment, the same boundary conditions have to be imposed in a given direction.

Some coding is needed if you want to impose let's say a no-slip and a free slip boundary condition in a spatial direction (for a turbulent boundary layer for example).

It you want to use a passive scalar with different boundary conditions than the velocity field ones, some coding is needed: you will have to create new derivative subroutines with the correct set of coefficients depending on the boundary conditions.

It is possible to use a stretched mesh in the $y-$direction with the parameter $istret$. If $istret = 0$, no stretched mesh is used. If $istret = 1$ then a stretching is used with a mesh refinement in the centre of the computational domain. If $istret = 2$ then a stretching is used with a mesh refinement at the boundaries of the computational domain. Finally, if $istret = 3$ then a stretching is used with a mesh refinement only at the bottom boundary of the computational domain. $istret = 2$ can be used for a turbulent channel flow, $istret = 3$ can be used for a turbulent boundary layer. The parameter `beta` is the stretching parameter (small values for strong stretching and large values for almost regular mesh).

It is possible to choose different flow configuration with the parameter `itype` but they are not all implemented yet. Defining a flow configuration (and a initial and/or inlet condition) can be done in the subroutines `init` and `ecoule`.

The parameters `ifirst` and `ilast` correspond respectively to the first and last iteration. As there is a checkpointing procedure, it is possible to split a simulation in small ones.

Two different formulations can be used for the convective terms of the Navier-Stokes equation:

- the rotational formulation $H_i = u_j \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right)$ corresponding to $iskew = 0$,
- the skew-symmetric formulation $H_i = \frac{1}{2} \left(\frac{\partial u_i u_j}{\partial x_j} + u_j \frac{\partial u_i}{\partial x_j} \right)$ corresponding to $iskew = 1$.

We recommend to use the skew-symmetric formulation for a better modelisation of the small scales and less aliasing.

It is possible to combine **Incompact3d** with Immersed Boundary Methods (IBM). For the latest IBM implemented in the solver, please have a look at:

Gautier R., Laizet S. & Lamballais E., 2014, **A DNS study of jet control with microjets using an alternating direction forcing strategy**, *Int. J. of Computational Fluid Dynamics*, **28**, 393--410

3.3 parameters.f90

This file contains the subroutine `parameter` which is used for the initialisation of the simulation (for instance, the time coefficients, the mesh sizes $\Delta x,\ \Delta y$ and Δz).

The subroutine `parameter` read the parameter file `incompact3d.prm`. If a stretching is used in the y-direction then the subroutine `stretching` is called to define the 1D array `yp` which contains the coordinates of the mesh in the y-direction.

3.4 variable.f90

The 3D arrays are dynamically defined in this file, depending on the size of the simulation and the number of computational cores. The 3D arrays with a 1 at the end (resp. a 2, a 3) are defined in X-pencils (resp. in Y-pencils, Z-pencils). For the 3D arrays in the X-pencils, two options are available. $*size$ can be used if there is no need to use the coefficients i, j, k in a subroutine (for example in the time advancement subroutine `intt`). For instance, to store the velocity uy at the previous time step, the array gy is defined with $allocate(gy(xsize(1), xsize(2), xsize(3)$ whereas $uy1$ is defined with $allocate(uy1(xstart(1) : xend(1), xstart(2) : xend(2), xstart(3) : xend(3)))$. In pratice, both arrays have the same size but the coefficients i, j, k will be different.

EXAMPLE:
A simulation is based on $n_x \times n_y \times n_z = 129 \times 64 \times 32$ mesh nodes using a 2D mapping $p_{row} \times p_{col} = 4 \times 8$=32 computational cores, ranking from 0 to 31.

For this configuration, $xsize(1) = n_x$, $xsize(2) = n_y/p_{row} = 16$, $xsize(3) = n_z/p_{col} = 4$.

The size of gy is $gy(129, 16, 4)$ with $i = 1, .., 129$, $j = 1, .., 16$ and $k = 1, .., 4$ for all the ranks.

The size of uy is $uy(129, 16, 4)$ with $i = 1, .., 129$ for all the ranks; $j = 1, .., 16$ for ranks 0 to 7; $j = 17, .., 32$ for ranks 8 to 15; $j = 33, .., 48$ for ranks 16 to 23 and $j = 49, .., 64$ for ranks 24 to 31. $k = 1, .., 4$ for ranks 0/8/16 and 24; $k = 5, .., 8$ for ranks 1/9/17 and 25; $k = 9, .., 12$ for ranks 2/10/18 and 26; $k = 13, .., 16$ for ranks 3/11/19 and 27; $k = 17, .., 20$ for ranks 4/12/20 and 28; $k = 21, .., 24$ for rank 5/13/21 and 29; $k = 25, .., 28$ for ranks 6/14/22 and 30 and finally $k = 29, .., 32$ for ranks 7/15/23 and 31.

When developing your own subroutine in the code or before starting a simulation, it is recommended to check/print the $*size, *start$ and $*end$ parameters for a better understanding of the 2D mapping decomposition.

3.5 schemes.f90 and derive.f90

The coefficients for the sixth-order schemes are defined in `schemes.f90`. 1D arrays are used to define the different matrices related to the computation of first and second derivatives. The derivatives and interpolations subroutines are defined in `derive.f90`.

More details are given in the following. The main reference for compact schemes (derivative, mid-point interpolation, for different boundary conditions) is:

K.S. LELE, 1992, **Compact finite difference schemes with spectral-like resolution**, *Journal of computational physics*, **103(1)**, 16-42.

3.5.1 Spatial discretization

Let us consider a uniform distribution of n_x nodes x_i on the domain $[0, L_x]$ with $x_i = (i-1)\Delta x$ for $1 \leq i \leq n_x$. The approximation of values $f'_i = f'(x_i)$ of the first derivative $f'(x)$ of the function $f(x)$ can be related to values $f_i = f(x_i)$ by a finite difference scheme of the form

$$\alpha f'_{i-1} + f'_i + \alpha f'_{i+1} = a\,\frac{f_{i+1} - f_{i-1}}{2\Delta x} + b\,\frac{f_{i+2} - f_{i-2}}{4\Delta x} \tag{1}$$

By choosing $\alpha = 1/3$, $a = 14/9$ and $b = 1/9$, this approximation is sixth-order accurate while having a so-called ``quasi-spectral behaviour'' due to its capabilities to represent accurately a wide range of scales.

Analogous relations can be established for the approximation of the second derivative values $f''_i = f''(x_i)$ with

$$\alpha f''_{i-1} + f''_i + \alpha f''_{i+1} = a\,\frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} + b\,\frac{f_{i+2} - 2f_i + f_{i-2}}{4\Delta x^2} + c\,\frac{f_{i+3} - 2f_i + f_{i-3}}{9\Delta x^2} \tag{2}$$

By choosing $\alpha = 2/11$, $a = 12/11$, $b = 3/11$ and $c = 0$, this approximation is sixth-order accurate. To control the aliasing errors via the viscous term, this type of schemes can also be defined to be over-dissipative at the highest wave numbers (in the spectral range where even a high-order finite difference scheme becomes inaccurate). More details can be found in:

LAMBALLAIS E., FORTUNE V. & LAIZET S., 2011 **Straightforward high-order numerical dissipation via the viscous term for Direct and Large Eddy Simulation**, *J. Comp. Phys.*, **230(9)**, 3270-3275.

As already stressed, four different boundary conditions can be considered in each spatial direction. The periodic and free-slip (equivalent to symmetric and antisymmetric conditions) boundary conditions allow the use of schemes (1, 2) for all the nodes considered. More precisely, the schemes (1, 2) just need to be relevantly modified near the borders $i = 1$ and $i = n_x$ through the substitution of the extra-node (sometimes called ``ghost'') values f_0, f_{-1}, f_{n_x+1}, f_{n_x+2} by their counterparts f_1, f_2, f_{n_x-1}, f_{n_x-2}. For a periodic boundary condition, this type of substitution for f, f' and f'' can be written

$$f_0 \to f_{n_x},\ f_{-1} \to f_{n_x-1},\ f'_0 \to f'_{n_x},\ f''_0 \to f''_{n_x}, \tag{3}$$

while symmetric and antisymmetric conditions lead to

$$f_0 \to f_2,\ f_{-1} \to f_3,\ f'_0 \to -f'_2,\ f''_0 \to f''_2, \tag{4}$$

and

$$f_0 \to -f_2,\ f_{-1} \to -f_3,\ f'_0 \to f'_2,\ f''_0 \to -f''_2, \tag{5}$$

respectively. For simplicity, only relations on the left boundary condition (near $i = 1$) are given here, their right counterparts (near $i = n_x$) being easily deduced.

When no-slip or open (i.e. Dirichlet for velocity) boundary conditions are used, nothing is assumed concerning the flow outside the computational domain. Single sided formulations are used for the approximation of first and second derivatives for these types of boundaries using relations of the form

$$\begin{aligned} f'_1 + 2f'_2 &= \frac{1}{2\Delta x}(-5f_1 + 4f_2 + f_3) \\ f''_1 + 11f''_2 &= \frac{1}{\Delta x^2}(13f_1 - 27f_2 + 15f_3 - f_4) \end{aligned} \tag{6}$$

that are third-order accurate. At the adjacent nodes, because a three-point formulation must be used, Padé schemes are employed with

$$\begin{aligned} \frac{1}{4}f'_1 + f'_2 + \frac{1}{4}f'_3 &= \frac{3}{2}\,\frac{f_3 - f_1}{2\Delta x} \\ \frac{1}{10}f''_1 + f''_2 + \frac{1}{10}f''_3 &= \frac{6}{5}\,\frac{f_3 - 2f_2 + f_1}{\Delta x^2} \end{aligned} \tag{7}$$

these schemes being fourth-order accurate.

3.5.2 Matrix formulation

The previous relations can be written as matrices:

$$\mathbf{A}_x\mathbf{f}' = \frac{1}{\Delta x}\mathbf{B}_x\mathbf{f} \tag{8}$$

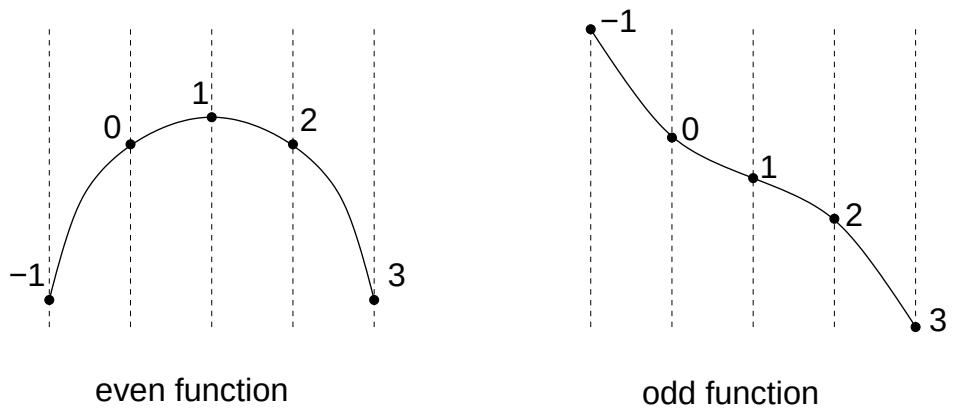

Figure 3: *symmetric and antisymmetric boundary conditions.*

$$\mathbf{A}'_x \mathbf{f}'' = \frac{1}{\Delta x^2} \mathbf{B}'_x \mathbf{f} \tag{9}$$

where $\mathbf{A}_x, \mathbf{A}'_x, \mathbf{B}_x, \mathbf{B}'_x$ are matrices with $n_x \times n_x$ coefficients. **f**, **f**′ et **f**′′ are vectors of size n_x.

1. *periodic case* : We have $f_{n_x+1} = f_1$ and $f_0 = f_{n_x}$. $\mathbf{A}_x$ and $\mathbf{B}_x$ can be written as:

$$\mathbf{A}_x = \begin{pmatrix} 1 & \alpha & & & & & & \alpha \\ \alpha & 1 & \alpha & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & \alpha & 1 & \alpha \\ \alpha & & & & & & \alpha & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{B}_x = \begin{pmatrix} 0 & a & b & & & & -b & -a \\ -a & 0 & a & b & & & & -b \\ -b & -a & 0 & a & b & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & -b & -a & 0 & a & b \\ b & & & & -b & -a & 0 & a \\ a & b & & & & -b & -a & 0 \end{pmatrix}$$

2. *free slip case* : We can have two situations, either symmetric or anti-symetric boundary conditions. f is either odd or even as shown in figure 3. When f is even we have:

$$\mathbf{A}_x = \begin{pmatrix} 1 & 0 & & & & & & \\ \alpha & 1 & \alpha & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & \alpha & 1 & \alpha \\ & & & & & & 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{B}_x = \begin{pmatrix} 0 & & & & & & & \\ -a & -b & a & b & & & & \\ -b & -a & 0 & a & b & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & -b & -a & 0 & a & b \\ & & & & -b & -a & b & a \\ & & & & & & & 0 \end{pmatrix}$$

When **f** is odd, we have:

$$\mathbf{A}_x = \begin{pmatrix} 1 & 2\alpha & & & & & & \\ \alpha & 1 & \alpha & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & \alpha & 1 & \alpha \\ & & & & & & 2\alpha & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{B}_x = \begin{pmatrix} 0 & 2a & 2b & & & & & \\ -a & b & a & b & & & & \\ -b & -a & 0 & a & b & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & -b & -a & 0 & a & b \\ & & & & -b & -a & b & a \\ & & & & & 2b & 2a & 0 \end{pmatrix}$$

3. *Dirichlet case* : In this case we need to use the non-centered schemes and we have:

$$\mathbf{A}_x = \begin{pmatrix} 1 & \alpha_1 & & & & & & \\ \alpha_2 & 1 & \alpha_2 & & & & & \\ & \alpha & 1 & \alpha & & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & \alpha_{n_x-1} & 1 & \alpha_{n_x-1} \\ & & & & & & \alpha_{n_x} & 1 \end{pmatrix}$$

$$\mathbf{B}_x = \begin{pmatrix} a_1 & b_1 & c_1 & & & & & \\ -a_2 & 0 & a_2 & & & & & \\ -b & -a & 0 & a & b & & & \\ & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \cdot & \cdot & \\ & & & -b & -a & 0 & a & b \\ & & & & & -a_{n_x-1} & 0 & a_{n_x-1} \\ & & & & & c_{n_x} & b_{n_x} & a_{n_x} \end{pmatrix}$$

IMPORTANT COMMENT: Note that the parameter `npaire` is only active when `ncl=1`. Please see below for the values depending on the velocity component and the spatial direction:

	X	Y	Z
ux	0	1	1
uy	1	0	1
uz	1	1	0
uxux	1	1	1
uyuy	1	1	1
uzuz	1	1	1
uxuy	0	0	1
uxuz	0	1	0
uyuz	1	0	0
dux/dx	1	1	1
duy/dx	0	0	1
duz/dx	0	1	0
dux/dy	0	0	1
duy/dy	1	1	1
duz/dy	1	0	0
dux/dz	0	1	0
duy/dz	1	0	0
duz/dz	1	1	1

3.6 tools.f90

3.6.1 Subroutine test_speed_min_max

This subroutine is used to calculate maxima and minima of the velocity field. Note that the value 1609 does not have a physical meaning. It corresponds to the date of birth of Sylvain Laizet (16th September) and is used for the initialisation of the min/max values.

3.6.2 Subroutine test_scalar_min_max

This subroutine is used to calculate maxima and minima of the scalar field.

3.6.3 Subroutine restart

This subroutine is used to either generate a restart file or to read a restart file (parameter `ilit` in `incompact3d.prm`). As already stated, you may have to split the restart file on IBM supercomputers when the simulation is too large.

3.6.4 Subroutine stretching

This subroutine is only active when a stretched mesh is used in the y-direction. It defines a stretched mesh and some 1D arrays which are used for derivatives and interpolations. More details can be found in

LAIZET S.& LAMBALLAIS E., 2009, **High-order compact schemes for incompressible flows: a simple and efficient method with the quasi-spectral accuracy**, *J. Comp. Phys.*, **228(15)**, 5989-6015.

3.6.5 Inversion subroutines

Several subroutines are available for the inversion of pentadiagonal matrices. It is requires when a stretched mesh is used in the y-direction.

3.7 decomp_2d.f90

This file is part of the 2D `Decomp%FFT` library which is freely available at http://www.2decomp.org/. It implements a general-purpose 2D pencil decomposition into `Incompact3d`. The files related to this library are not explained in this document. As a communication library, it implements so-called 2D pencil decomposition for partitioning three-dimensional data sets on distributed systems and performing transpose-based communication. It also provides a highly scalable and efficient interface to perform three-dimensional Fast Fourier Transforms (FFT). The library is written in Fortran and built on top of MPI. It is recommended not to modify this file.

3.8 convdiff.f90

The convective and diffusive terms of the Navier-Stokes equations are calculated in this file. As already stated, two different formulations can be used for the convective terms:

- the rotational formulation $H_i = u_j \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right)$ corresponding to $iskew = 0$,
- the skew-symmetric formulation $H_i = \frac{1}{2} \left(\frac{\partial u_i u_j}{\partial x_j} + u_j \frac{\partial u_i}{\partial x_j} \right)$ corresponding to $iskew = 1$.

We recommend to use the skew-symmetric formulation for a better modelisation of the small scales and for less aliasing. Note that at the beginning and at the end of the subroutine, the data are stored in X-pencils.

Note that when a stretched mesh is used in the y-direction, the evaluation of the second derivative is

$$\frac{\partial^2 f}{\partial y^2} = \frac{\partial^2 f}{\partial s^2} \left(\frac{ds}{dy} \right)^2 + \frac{\partial f}{\partial s} \frac{d^2 s}{dy^2} = \frac{1}{h'^2} \frac{\partial^2 f}{\partial s^2} - \frac{h''}{h'^3} \frac{\partial f}{\partial s}$$

with

$$y = h(s), \quad 0 \leq s \leq 1, \quad 0 \leq y \leq L_y$$

where $h(s)$ is the mapping from the equally spaced coordinate s to the stretched spaced coordinate y.

Large-Eddy Simulations (LES) models should be implemented at the end of the `convdiff` subroutine.

Any forcing term should also be implemented here (for instance a Coriolis forcing).

3.9 navier.f90

3.9.1 Subroutine intt

Subroutine for the explicit time advancement of the Navier-Stokes equation. Note that the 3D loops are fully vectorised, hence the `(ijk,1,1)` notation. Data are stored in X-pencils.

3.9.2 Subroutine corgp

Subroutine for the correction step in the fractional step method. The velocity field is corrected by the pressure gradients in order to become incompressible. Data are stored in X-pencils.

In the case of a temporal channel flow, it is necessary to modify the flow rate to account for the pressure loss at the wall (subroutine `channel`).

3.9.3 Subroutines inflow and outflow

Subroutines only used if inflow/outflow boundary conditions are used in the streamwise direction of the flow (`nclx=2`). The outflow condition is a simple 1D convection equation. The inflow condition can be adapted to any kind of inlet condition via the subroutine `ecoule`.

A random noise can be added to the inlet if necessary.

3.9.4 Subroutine ecoule

In this subroutine, the inlet condition is defined. At the moment, only few different flow configurations are implemented (`itype==1` for a constant flow field, `itype==2` for a Poisseuille profile, `itype==6` for the Taylor-Green vortices).

Note that the subroutine `ecoule` is called in the subroutine `inflow` at each time step.

3.9.5 Subroutine init

Subroutine dedicated to the initialisation of the simulation via the subroutine `ecoule`. Data are stored in X-pencils.

Random noise can be added to the initial condition.

3.9.6 Subroutine divergence

Subroutine dedicated to the evaluation of $\boldsymbol{\nabla}.\mathbf{u}^{**}$ when `nlock=1` and to $\boldsymbol{\nabla}.\mathbf{u}^{n+1}$ when `nlock=2`.

Note that the inputs are the three components of the velocity defined in X-pencils and the output is defined in Z-pencil (first argument of the subroutine). The output is defined on the staggered mesh.

3.9.7 Subroutine gradp

Evaluation on the X-pencils of the three pressure gradients (on the velocity mesh) from the pressure field (defined on the staggered mesh) which was evaluated in Z-pencils.

3.9.8 Subroutine corgp_IBM

Only used when Immersed Boundary Methods are used. It has to do with a pre- and post-correction by the pressure gradient of the flow inside the solid regions. More details about the method can be found in:

LAIZET S.& LAMBALLAIS E., 2009, **High-order compact schemes for incompressible flows: a simple and efficient method with the quasi-spectral accuracy**, *J. Comp. Phys.*, **228(15)**, 5989-6015.

3.9.9 Subroutine body

Only used when Immersed Boundary Methods (IBM) are used. In this subroutine, the velocity field is forced to zero inside the solid regions.

Note that new IBM are currently under development. The latest method implemented in the solver is described in:

GAUTIER R., LAIZET S. & LAMBALLAIS E. 2014 **A DNS study of jet control with microjets using an alternating direction forcing strategy**, *Int. J. of Computational Fluid Dynamics*, **28**, 393--410

3.9.10 Subroutine pre_correc

It is impossible to impose boundary conditions on $\mathbf{u}^{k+1}$ because of incompressibility (the velocity field cannot be modifoed after the correction step by the pressure gradients). Therefore the imposition of boundary conditions is on $\mathbf{u}^{**}$ instead of $\mathbf{u}^{k+1}$.

$\Rightarrow$ See example for a channel flow:

$$\mathbf{u}^{*}_{wall} = 0$$

$$\mathbf{u}_{wall}^{k+1} = \mathbf{u}_{wall}^{*} - \boldsymbol{\nabla}\tilde{p}^{k+1}$$

$$\mathbf{u}_{wall}^{k+1} = -\boldsymbol{\nabla}\tilde{p}^{k+1} \neq 0$$

It is not satisfactory. It is better to do the following:

$$\mathbf{u}_{wall}^{*} = \boldsymbol{\nabla}\tilde{p}^{k}$$

$$\mathbf{u}_{wall}^{k+1} = \mathbf{u}_{wall}^{*} - \boldsymbol{\nabla}\tilde{p}^{k+1}$$

$$\mathbf{u}_{wall}^{k+1} = \boldsymbol{\nabla}\tilde{p}^{k} - \boldsymbol{\nabla}\tilde{p}^{k+1} \approx 0$$

As it can be seen, corrections with the pressure gradients (at the previous time step) are needed so that the boundary conditions are more accurate.

Note that boundary conditions only need to be imposed in the subroutine `pre_correc` when `ncl`=2.

For particular flow configurations, it is necessary to check that the flow rate at the outflow is the same as the flow rate at the inflow. This correction can be performed in the subroutine `pre_correc`.

3.10 poisson.f90

This file has to do with the Poisson equation which is performed in the spectral space even when Dirichlet boundary conditions are used for the velocity field.

Note that the pressure field is obtained on a staggered mesh, hence the use of a pre and post processing for the Fast Fourier Transforms.

The modified wave numbers are defined in the subroutine `waves`.

Note that the implementation of the Poisson solver for the stretched mesh is fairly complicated and involved an inversion of one or two penta-diagonal matrices. The theory can be found in

LAIZET S.& LAMBALLAIS E., 2009, **High-order compact schemes for incompressible flows: a simple and efficient method with the quasi-spectral accuracy**, *J. Comp. Phys.*, **228(15)**, 5989-6015.

Please pay attention in particular to the appendix B.

It is recommended not to modify this file.

12 different combinations can be used for the Poisson equation, covering a wide range of flow configuration: $(0-0-0)$, $(1-0-0)$, $(2-0-0)$, $(0-2-0)$, $(1-1-0)$, $(2-1-0)$, $(1-1-1)$, $(2-2-1)$, $(2-1-2)$, $(1-2-1)$, $(1-1-2)$ and $(2-2-2)$. For the Poisson solver the case `ncl=1` and `ncl=2` are dealt in the same way.

3.11 visu.f90

File which contain all the post-processing subroutines. Several tools have been developed to manage efficiently Input/Output in the code. As previously explained there is a restart procedure in the code (file `tools.f90`, subroutine `restart`). A restart file `sauve.dat` is generated every `isave` time step. Note that it is possible to restart the simulation with different number of computational cores. In terms of 2D/3D snapshots, several subroutines can be used:

- **2D snapshots (full resolution):**
  subroutine `decomp_2d_write_plane(pencil,var,iplane,n,filename)`:
  `ipencil` is equal to 1,2,3 for X-pencil, Y-pencil, Z-pencil respectively. `var` is the name of the 3D array. `iplane` is equal to 1,2,3 to save a X-plane, Y-plane, Z-plane respectively. `n` corresponds to the location of the plane. `filename` is the name of the output file.

EXAMPLE:
`call decomp\_2d\_write\_plane(1,ux1,1,112,'ux2d')` is going to write in `ux2d` a 2D (y-z)-plane of the 3D array ux1 (defined in X-pencil), for i=112.

- **3D snapshots (full resolution):**
subroutine `decomp_2d_write_one(nx,ny,nz,ipencil,var,filename)`:
`ipencil` is equal to 1,2,3 for X-pencil, Y-pencil, Z-pencil respectively. `var` is the name of the 3D array. `filename` is the name of the output file.

  EXAMPLE:
  `call decomp_2d_write_one(nx,ny,nz,2,uy2,'uy2.dat')` is going to write in `uy2.dat` the 3D array `uy2` (defined in Y-pencil).

- **3D snapshots (coarse resolution):**
subroutine `decomp_2d_write_one(ipencil,var,filename,icoarse)`:
`ipencil` is equal to 1,2,3 for X-pencil, Y-pencil, Z-pencil respectively. `var` is the name of the 3D array. `filename` is the name of the output file. `icoarse` is equal to `nstat` or `nvisu` (see file `module_param.f90`). The array `var` is defined on a coarse mesh (every `nstat` or `nvisu` mesh node). Before calling the subroutine it is necessary to call the subroutine
`fine_to_coarseV(ipencil,var_full,var_coarseV)` or
`fine_to_coarseS(ipencil,var_full,var_coarseS)`.
The size of `var_coarseV` is `(xszV(1),xszV(2),xszV(3))`.
The size of `var_coarseS` is `(xszS(1),xszS(2),xszS(3))`.

  EXAMPLE:
  `call fine_to_coarseV(1,uz1,uvisu)` will write in the coarse array `uvisu` the 3D arrays (full resolution) `uz1` (defined in Z-pencil).
  Then `call decomp_2d_write_one(1,uvisu,'uz_coarse.dat',2)` will write in `uz_coarse.dat` the 3D array `uvisu` (defined every `nvisu` mesh nodes in X-pencil).

- **X-pencils (various resolutions):**
It is obviously possible to save data for only a few numbers of X-pencils.

  EXAMPLE:
  In this example, a simulation with $n_x \times n_y \times n_z = 2881 \times 360 \times 360$ is running on 3600 computational cores with a 2D mapping $p_{raw} \times p_{col} = 60 \times 60$. I want to save the three components of the velocity at each time step for $j = k = 181$ and every 8 mesh nodes in the $x-$direction. I will simply use the following lines:

```
if (nrank==4095) then
if (itime==ifirst) then
write (filename,923) nrank
open (nrank,file=filename,form='unformatted')
endif
write (nrank)(((ux1(i,j,k),i=1,xsize(1),8),j=1,xsize(2),2),k=1,xsize(3),2),&
(((uy1(i,j,k),i=1,xsize(1),8),j=1,xsize(2),2),k=1,xsize(3),2),&
(((uz1(i,j,k),i=1,xsize(1),8),j=1,xsize(2),2),k=1,xsize(3),2)
if (itime==ilast) close (nrank)
endif
```

  To identify which computational core corresponds to $j = k = 181$, it is recommended to use the arrays $xstart$ and $xend$. A similar procedure can be used to save data in Y-pencils and Z-pencils.

- **sub-3D domains:**
subroutine `decomp_2d_write_subdomain(ipencil,var,imin,imax,jmin,jmax,kmin,kmax,filename)`:
This subroutine is only available in the latest version of the solver.

  EXAMPLE:
  `call decomp_2d_write_subdomain(1,ux1,851,1600,311,411,311,511,'ux_sub.dat')` will write in `ux_sub.dat` part of the 3D array `ux1` for $i = 851, 1600$, $j = 311, 411$ and $k = 311, 511$