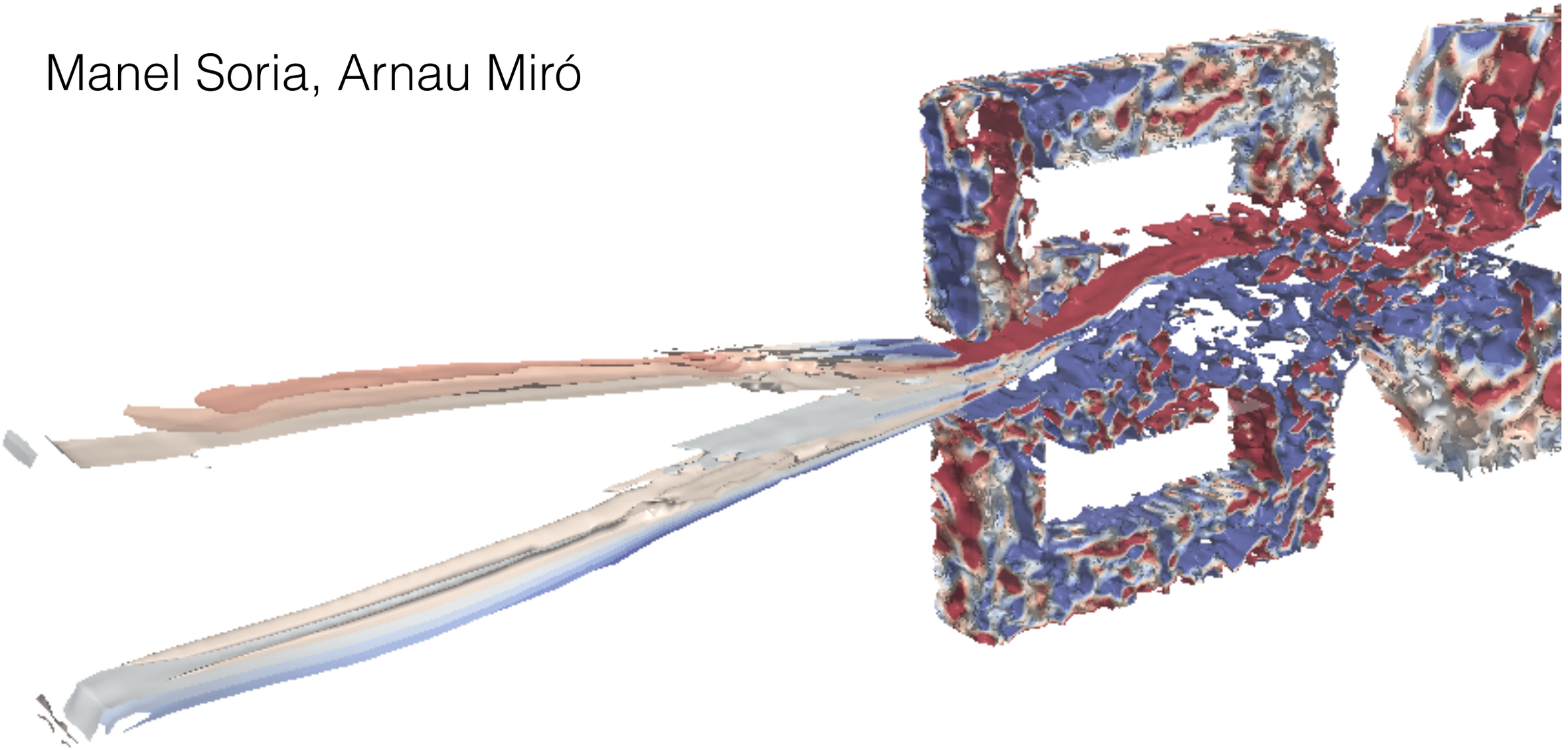
Introduction to open-source CFD with Code_Saturne

Manel Soria, Arnau Miró



ESEIAAT – UPC
Department of Physics / Aerospace Division

Contents

Introduction to *Code_Saturne*

Code_Saturne introduction

Why *Code_Saturne*

The Unified European Application Benchmark Suite (UEABS) […], with the objective of providing a set of **scalable**, currently relevant and publically available codes which can realistically be run on **large systems**, and maintained into the future. (PRACE)

Need for a large system (towards hexascale), being able to be run using a high number of processors in an efficient way.

Code_Saturne introduction

Overview

Code_Saturne is a 3D unstructured finite volume code developed by EDF since 1998. It was made open-source in 2007 under the GPL license.

Its main features are:

- Written in FORTRAN (49%), C (41%) and Python (10%).
- Graphical User Interface available in Qt (xml format files).
- Parallel coding using MPI.
- Highly portable: Linux/UNIX, Solaris, Irix64, HP-UX, Mac OS.
- Validated on different supercomputing architectures: Opteron, Itanium, Power5-6, BlueGene.

Code_Saturne introduction

Features

Code_Saturne can solve a variety of problems:

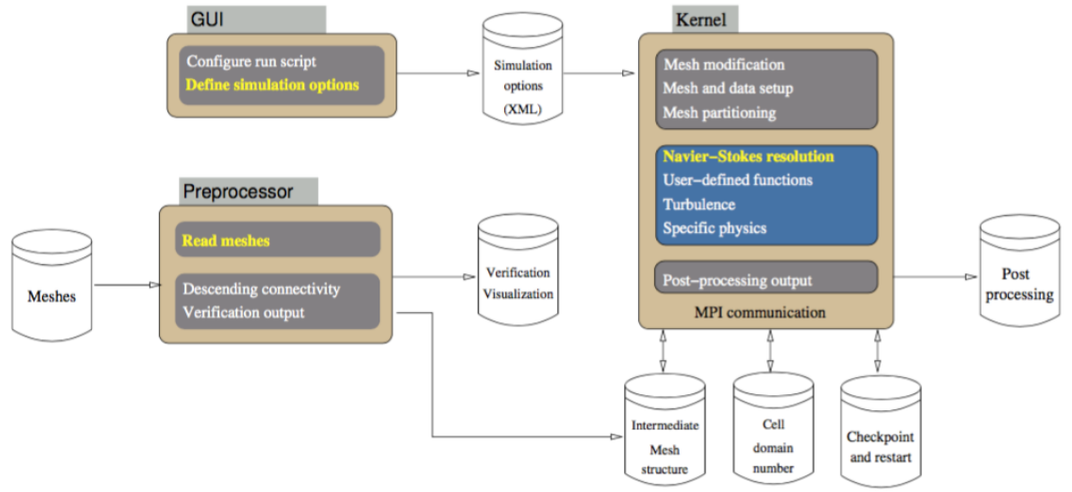
- Single phase turbulent flows (DNS, LES and various RANS models).
- Compressible flow.
- Radiative heat transfer.
- Combustion: coal, heave fuel oil, gas.
- Electric arc and Joule effect.
- Lagrangian module for dispersed particle tracking.
- ALE method for deformable meshes.
- Conjugate heat transfer (SYRTHES and 1D module).
- Turbomachinery.
- Specific modules for nuclear waster storage and cooling towers.

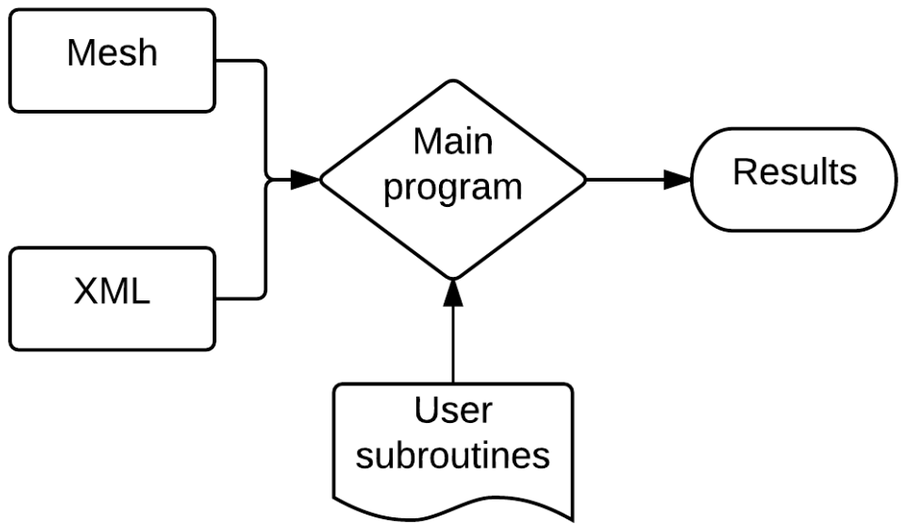
Code_Saturne introduction

Workflow

Code_Saturne consists of:

- **GUI** for configuring main parameters (in xml file).
- **Preprocessor** for mesh check and operations (e.g., joining, partitioning,…).
- **User subroutines** for advanced parameter tweaking or extra operations.
- **Kernel** (i.e., main program) recompiles the subroutines and reads the xml, then launches the case.
- **Postprocessing**, which is done in Paraview.



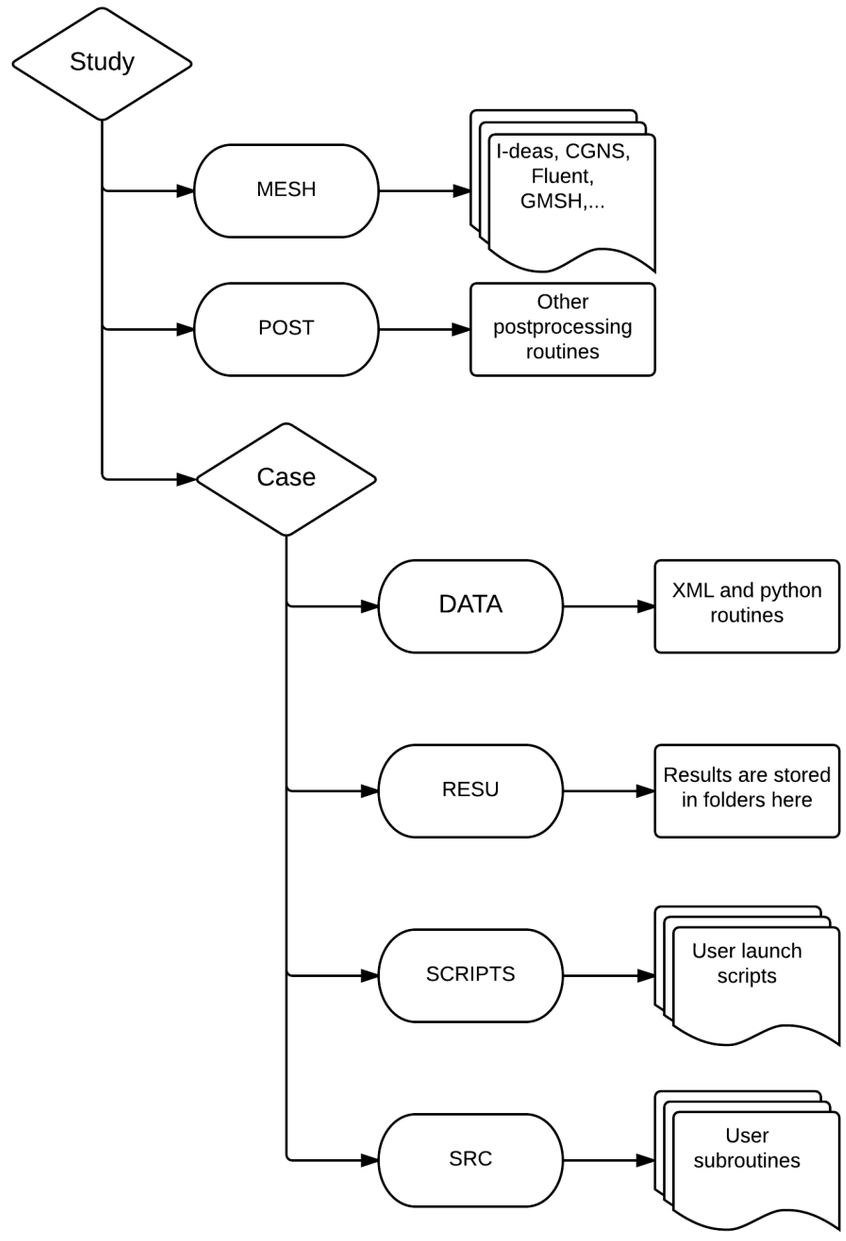


Code_Saturne introduction

Structure

Code_Saturne is structured into studies that compress several cases. In short:

- STUDY contains the cases, the mesh and a post-processing (empty) directories.
- CASE contains all input data, running scripts and results files.
  - DATA has the graphical interface script and xml file.
  - RESU has the saved results after the simulation has finished.
  - SCRIPT contains the runcase file to launch the calculation.
  - SRC contains the user subroutines.



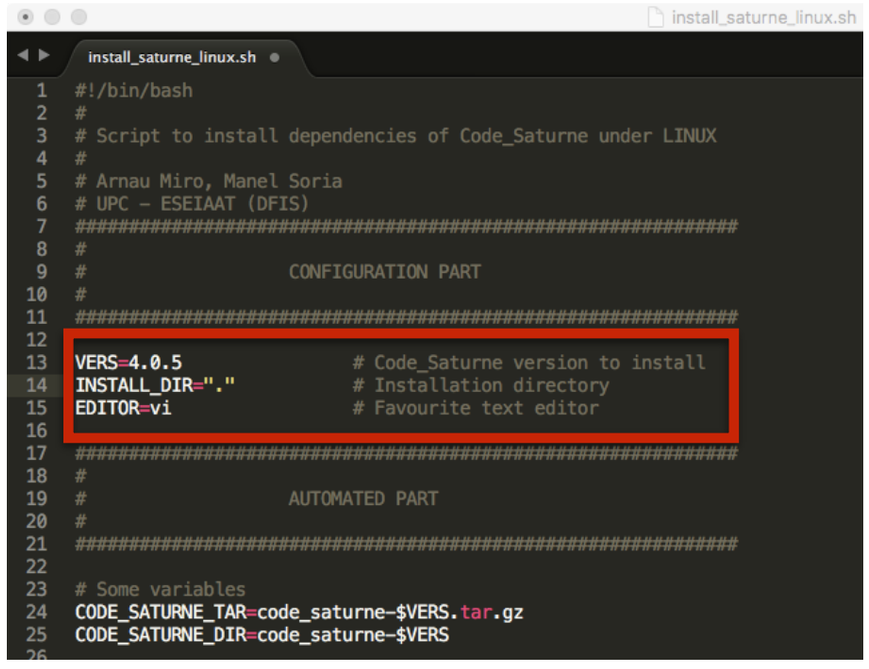
Installation notes

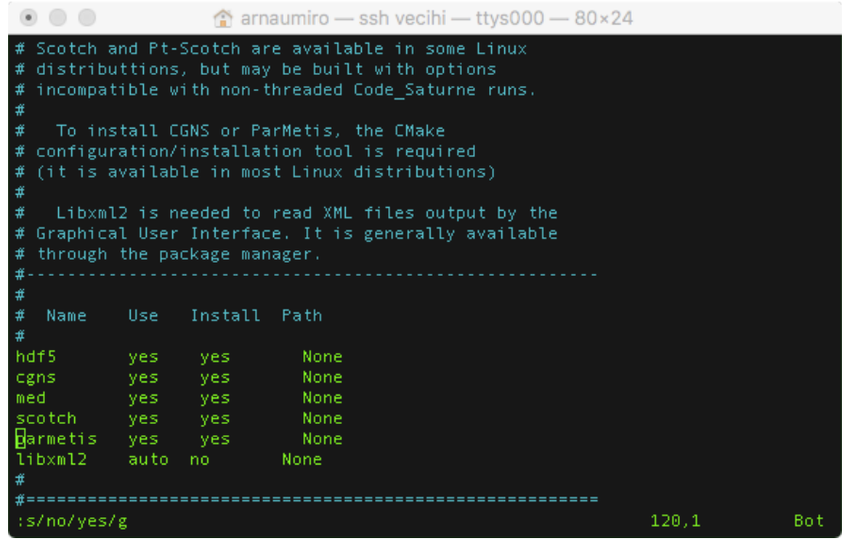
Code_Saturne installation

Linux/Unix using script

For easy installation, use the provided *install_saturne_linux.sh* script.

1. Open the script in your favorite text editor and change the variables VERS, INSTALL_DIR and EDITOR to the ones that suit you most.
   - VERS: Code_Saturne version to install (current 4.0.5).
   - INSTALL_DIR: Directory where to install (recommended “/opt”).
   - EDITOR: Your favorite text editor (gedit, nano, vim,…).
2. Run the script “*. install_saturne_linux.sh*”.
3. Once in the setup script, check that the prefix folder is ok and set the optional packages to “yes” (except libxml2 that should appear auto).
4. Save and let it install.





Code_Saturne installation

Other builds

For other builds under Linux/Unix please refer to the installation manual.

For a windows build, a compiled version exists on the *code_saturne* webpage. It is advisable to put the "code_saturne.com" to the cmd path so it can be launched from cmd as if in Linux/Unix.

For MacOS builds, *code_saturne* generally does not work and needs a advanced installation procedure.

Features

Code_Saturne features

Overview

Code_Saturne features a number of advanced parameters, among them:

- Mesh preprocessor
- Special physics
- Couplings
- Numerical parameters
- Output parameters
- Post processing

Code_Saturne features

Mesh formats

No mesh generator included in *Code_Saturne* but it can read a wide range of formats:

- Universal I-DEAS (.unv). From NX or ICEM software.
- Hexa NUMECA (.hex)
- MED 2.2 (.med). From Salomé platform .
- CGNS (.cgns). Generic format.
- Gmesh.
- Gambit Neutral (.neu).
- StarCD (.ccm). Via libccmio available from StarCD on request.

Code_Saturne features

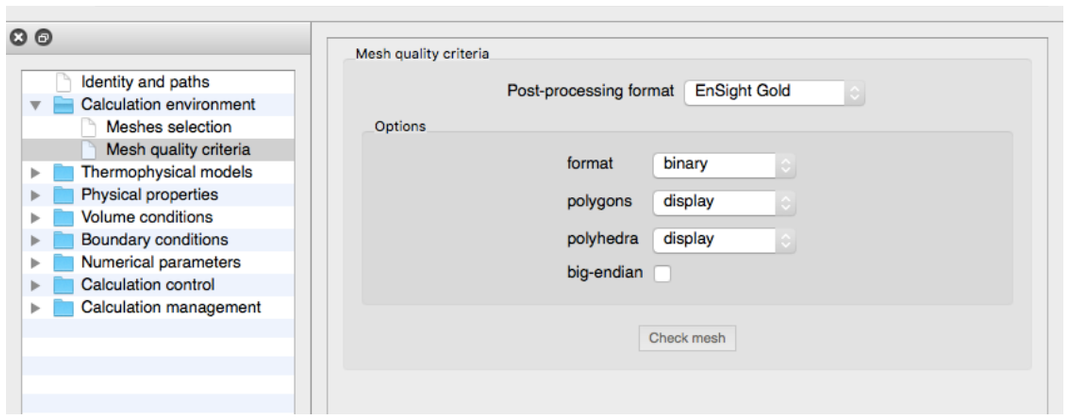
Mesh preprocessor

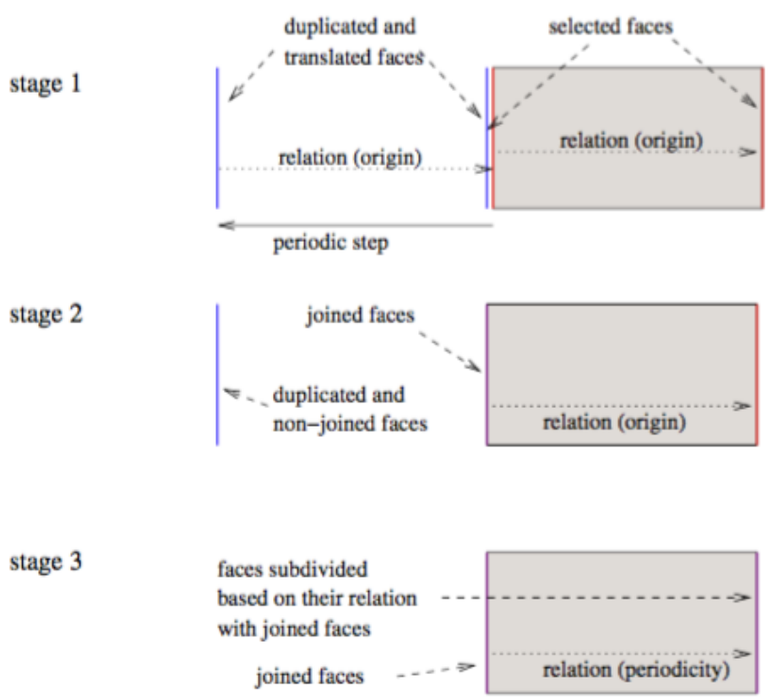
Mesh operations are handled by the preprocessor (which works independently from the kernel). Interesting features are:

- **Mesh manipulation**
- **Joining**
- **Periodicity**
- **Mesh quality**

Note that faces ids must be selected in pairs for joining or periodicity.

Mesh quality is recommended to run before launching the case to quickly detect mesh issues.

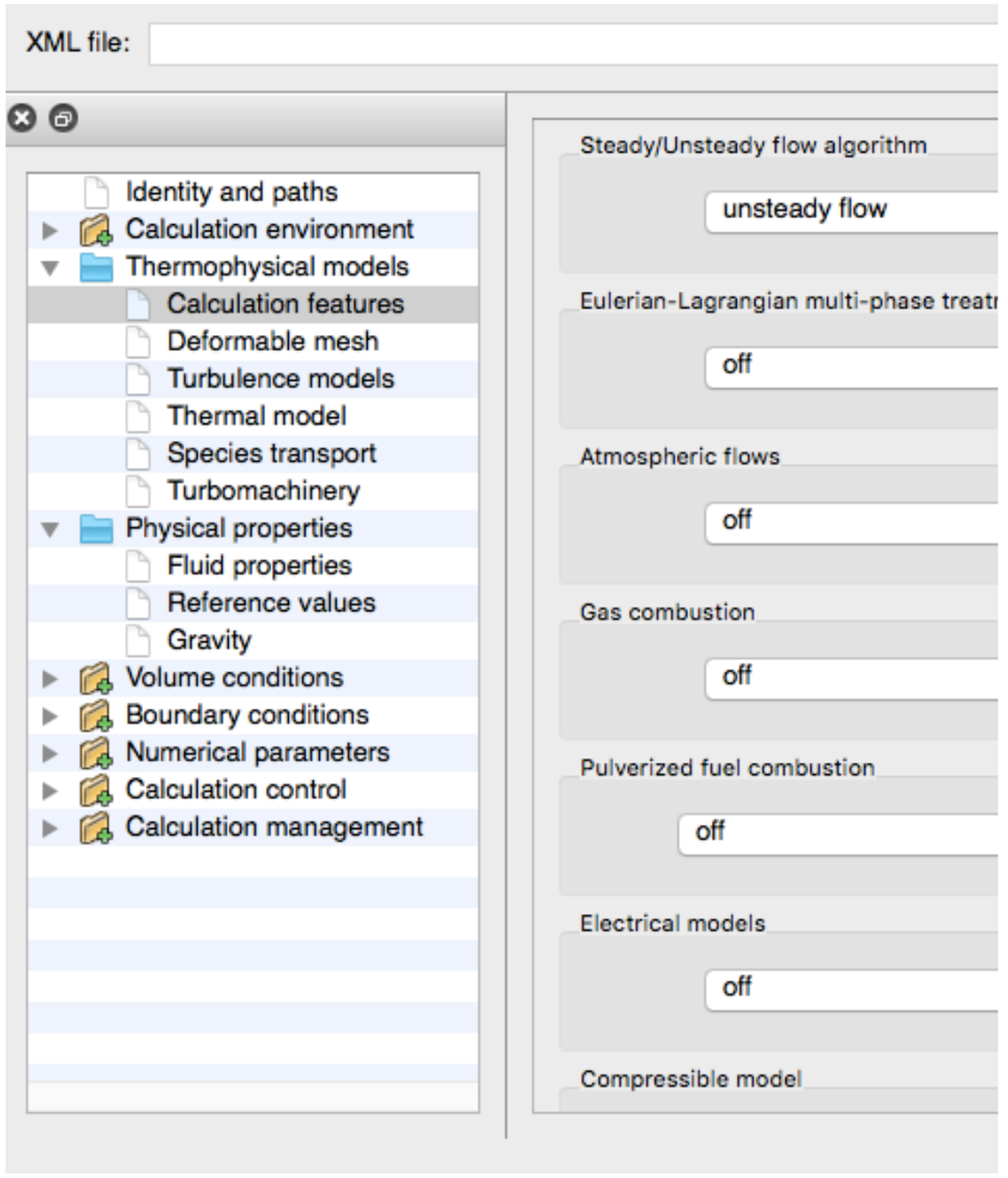




Code_Saturne features

Special physics

- Deformable mesh:
  - Moving mesh (turbomachinery)
  - ALE
- Thermal model
- Compressible model
- Multiphase flows
- Electrical models
- Combustion
  - Gas
  - Pulverized fuel



Code_Saturne features

Code coupling

Code_Satune can be coupled with other softwares for joint computations:

- **Syrthes**: for thermal-fluid computations.
- **Code_Aster**: for structure-fluid computations.
- **Code_Saturne**: for fluid-fluid computations.

Code_Saturne features

Numerical parameters overview

There are a number of solver and numerical parameters that can be set in *Code_Saturne*.

- **Gradient calculation (imrgra)**: Type of approximation for gradient computation.
- **Solution scheme**: generally select centered (central difference, 2nd order in space), with no blending factor, no slope test and no flux reconstruction. Generally select ~10 RHS sweep reconstructions for time scheme of 2nd order.
- **Time scheme**: generally defaults to a 1st order temporal scheme; a 2nd order temporal scheme is also available.
- **Time step**: generally set as constant with reference time step (dt) and number of iterations (N) so that t=N*dt.

Code_Saturne features

Output parameters

Code_Saturne generally saves two output meshes:

- **Fluid domain**: for fluid variables such as pressure, velocity, etc.
- **Boundary**: for wall defined variables such as y+, efforts, fluxes, etc.

They can all be set through the GUI. Additional output can be achieved through subroutines.

Additionally, time averages (time moments) can be set such as <u>, <uu> or <uv>, which are useful to compute flow statistics.

Code_Saturne features

Checking the results

After the calculation has finished, The RESU directory contains directories by date: YYYYMMDD-HHMM (i.e. 20161003-1020):

- postprocessing: Has the files for visualisation using Ensight or ParaView.
- monitoring: Contains history files of variables at given locations (probes).
- checkpoint: Has the necessary files to restart a calculation.
- listing: Summary of the computations
- src saturne: A copy of the FORTRAN files used to run the case.
- Any user output files.
- preprocessor.log and listing files for the preprocessor and kernel executions.
- Copy of the .xml file used.
- The files summary and setup.log with info about the calculation.

Code_Saturne features

Postprocessing

On the postprocessing folder…

- A case file with info on files RESULTS.case.
- A geometry file results.geo.
- One file per variable per time iteration (frequency set by the variable NTCHR in “cs_user_parameters.f90” or in the Calculation Control - Output control page of the GUI).
- Files can be opened using ParaView (open-source under GPL).

Parts can be created by the user for specific sections of the geometry.

Code_Saturne features

Monitoring points

On the monitoring points…

- One file per variable.
- Variation over number of iterations.
- Can be visualised via xmgrace, gnuplot (2D open source plot program), matlab (.dat files) or ParaView (.csv files).
- Monitoring points defined in “cs_user_parameters.f90” or in Calculation control - Output control page in the GUI.

Code_Saturne features

ParaView visualization

ParaView can read Ensight files contained in the postprocessing folder.

For visualizing the part, the “Extract Block” filter must be applied and the “Fluid Domain” part must be selected.

Code_Saturne is a finite volume code, therefore, the information is on the cell centers. To interpolate the data to the nodes (for a smoother visualization), the “celldata2pointdata” filter must be used.

User subroutines

Code_Saturne subroutines

Overview

Code_Saturne provides great capabilities and user customization via FORTRAN subroutines.

- The running script runcase will compile all files in the SRC directory and link them with the existing libraries.
- Subroutines will have precedence to any settings on the xml files from the GUI.
- The user's subroutines are in the REFERENCE directory inside SRC, and some examples can be found in the EXAMPLES directory.
- For full description of subroutines see the user's guide.

Code_Saturne subroutines

Basic subroutines

- **cs_user_parameters.***: Set calculation parameters, physical and numerical.
- **cs_user_boundary_conditions.f90**: Boundary conditions definitions. All types of boundaries can be set, but pre-defined ones are: inlet, outlet, symmetry and walls (periodicity set on cs user mesh.c file).
- **cs_user_physical_properties.f90**: Definition of physical properties (variable ρ, μ, Cp etc.)
- **cs_user_initialization.f90**: Initialization of variables.
- **cs_user_extra_operations.***: User postprocessing (profiles, budgets …). Very useful and with plenty of examples.
- **cs_user_postprocess***: subroutines for the post-processing with ParaView (or Ensight) parts. User can create parts and assign variables to the new parts.
- **cs_user_source_terms.f90**: Source terms for each computed variable, including scalars (e.g. ΔP).

Code_Saturne subroutines

Structure

Important points to take into account:

- *Code_Saturne* is in double precision. Declare your variables as DP (and assign values as 1.d0).
- Most integer variables start with “i”, i.e. icodcl.
- Compilation not only of user subroutines but ANY f90 or c file in SRC (this could include *Code_Saturne* original source routines).
- If an input file is needed, place it in DATA and link it by editing the cs_user_scripts.py file.
- To check potential compilation errors before running, use the command “code_saturne compile” inside the SRC directory.

Code_Saturne subroutines

Main variables

Array dimensions (set by the kernel, not to be changed by the user)

- **ndim** : Number of dimensions.
- **ncel**, **ncelet**: Number of cells and cells plus ghost cells, per MPI task.
- **nfac**, **nfabor**: Number of internal and boundary faces.
- **nvar**: Number of variables (depends on case and the physics).
- **nfml**: Number of referenced families of entities (boundary faces, elements, ...).
- **nprfml**: Number of properties per referenced entity family.
- **nscal**: Number of transported scalars (not turbulence vars), i.e. the temperature and other scalars. Can be divided into two groups: **nscaus** user-defined scalars and **nscapp** scalars related to a “specific physics”. **nscal**=**nscaus**+**nscapp**.

Code_Saturne subroutines

Main variables

Geometrical arrays (set by the preprocessor, not be changed by the user):

- **xyzcen(ndim,ncelet)**: Coordinates of the cell centers.
- **cdgfac(ndim,nfac)**: Coordinates of the internal faces.
- **cdgfbo(ndim,nfabor)**: Coordinates of the boundary faces.
- **surfac(ndim,nfac)**: Vector normal to the internal face whose length is the area of the face.
- **surfbo(ndim,nfabor)**: Vector normal to the boundary face whose length is the area of the face.
- **volume(ncelet)**: Volume of each cell.
- **ifacel(2,nfac)**: Connectivity between faces and cells. Contains cells at each side of the face.
- **ifabor(nfabor)**: Connectivity between faces and cells at the boundaries (only one cell per face).

Code_Saturne subroutines

Boundary conditions

The boundary conditions can be set in cs_user_boundary_conditions.f90. Each face has to be assigned:

- a type of face **itypfb**.
- a type of boundary condition (Dirichlet, flux …).
- **icodcl** a value associated with that boundary condition **rcodcl**.

Possible face types are: **ientre** (inlet), **iparoi** (wall), **iparug** (rough wall), **isymet** (symmetry), **isolib** (outlet) and **iindef** (undefined).

Depending on the face type a value might have to be assign to **rcodcl** (Inlet values for the variables for example).

If **iindef** is used, all three values have to be assign for each variable.

Code_Saturne subroutines

Parallel considerations

Code_Saturne uses MPI to communicate between different domains.

All the MPI calls are done in C and they have a FORTRAN equivalent in *Code_Saturne*.

This needs to be taken into account when calculating anything that needs values on other processors (global maximum or minimum, total number of cells ...) .

Values at “ghost” cells are updated at the beginning of the iteration, but if you need to, you can update them by calling synsca and synvec (useful when calculating a gradient).

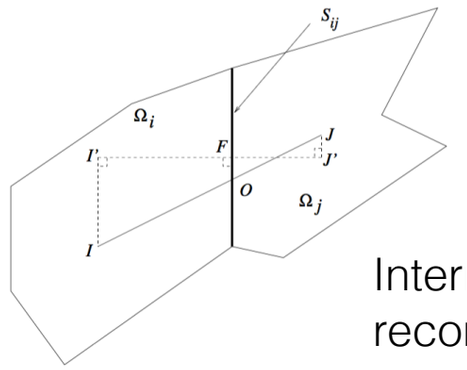
See cs_user_extra_operations.f90 for examples on several parallel subroutines.

Some examples…

Code_Saturne subroutines

Gradient reconstruction

Gradient reconstruction handles how to compute the cell gradient for vector or scalar fields. It is controlled by the **imrgra** parameter. Available methods are:



Internal faces gradient reconstruction.

0. Iterative reconstruction of non-orthogonalities.
1. Least squares based of first neighbours.
2. Least squares based on extended neighbours.
3. Least squared based on partial extended neighbours.
4. Iterative reconstruction with least squares initialization (first neighbours)
5. Iterative reconstruction with least squares initialization (extended neighbours)
6. Iterative reconstruction with least squares initialization (partial extended neighbours)

The recommended method for production is *imrgra=5*, set in the cs_user_parameters.f90.

Code_Saturne subroutines

2nd order time scheme

By default, *Code_Saturne* uses a first order time scheme for time integration, activating only the second order Crank-Nicholson scheme for LES.

When second order time scheme is activated, *Code_Saturne* assumes that the time step is constant (Note: second order time scheme is not available in RANS models).

In order to activate the second order time scheme, the variable **ischtp** has to be set in cs_user_parameters.f90.

- **ischtp** = 1: first order time scheme.
- **ischtp** = 2: second order Crank-Nicholson scheme.

Code_Saturne subroutines

Buffer zone

The buffer zone is a part of the domain in which the Reynolds number is decreased from 10 to 100 times that of the domain. For this reason, viscosity is increased 10 to 100 times that of the domain (**viscl0**).

Buffer zone is implemented in cs_user_physical_properties.f90.

The first thing is to set the length in which the buffer zone activates (xyzbufferzone).

ivivar = 1 is set to allow for variable viscosity.

Code_Saturne subroutines

Pressure outlet

The default outlet boundary condition corresponds to a Neumann condition to all transported variables.

The condition is changed so that a prescribed Dirichlet condition is applied for pressure at the outlet.

This is done in the cs_user_boundary_conditions.f90:

The faces at a given boundary are retrieved by

call getfbr('color',nlelt,lstelt)

being 'color' the designation of the boundary. A loop is then performed on the boundary elements nlelt.

itypfb(ifac) = isolib ! for outlet condition

for pressure (ipr) a Dirichlet condition is set

icodcl(ifac,ipr) = 1 !(1 for Dirichlet, 3 for Neumann)

rcondcl(ifac,ipr,1) = P ! (eg. 101300.d0)

rcondcl(ifac,ipr,2) = rinfin

rcondcl(ifac,ipr,3) = 0.d0

Code_Saturne subroutines

Forces computation

Code_Saturne computes the forces (in Newtons) on a boundary face and stores them in **iforbr**. Such forces are then accessed and stored (per component) in a binary file (coefficients.dat, which is later parsed by Matlab to retrieve the drag and lift coefficient as:

$$C_D = \frac{xfor(1)}{0.5\rho v^2 A} \qquad C_L = \frac{xfor(2)}{0.5\rho v^2 A}$$

Forces due to pressure are also computed as:

$$xpre = (P - P_\infty)\,\hat{S}$$

and stored at the same file for post-processing.

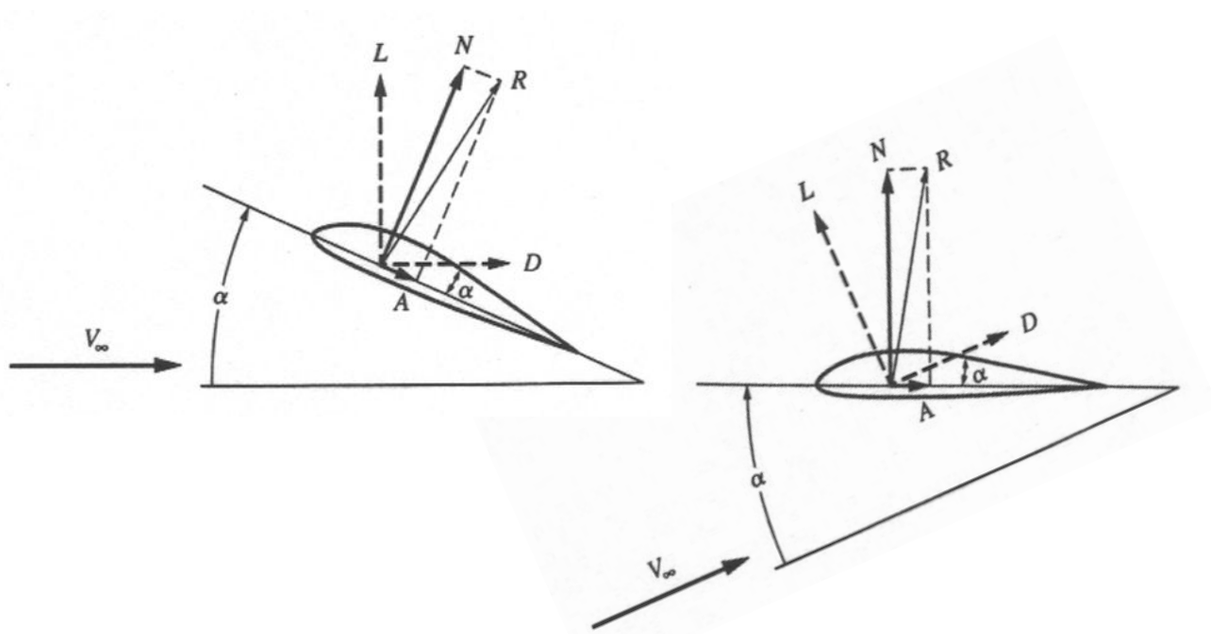
Code_Saturne subroutines

Forces computation

If the velocity is deflected a certain angle of attack instead of the geometry, *Code_Saturne* returns the normal (N) and axial (A) forces instead.

Lift and drag forces can be easily recovered in post-processing by means of:

$$D = N \sin\alpha + A \cos\alpha \qquad L = N \cos\alpha - A \sin\alpha$$



Conversion from Normal and Axial to Lift and Drag

Case setup walkthrough

Code_Saturne walkthrough

Overview

Generally, the following steps should be taken when running a case with *Code_Saturne*:

1. Create the STUDY and the CASE structure.
2. Copy the meshes to MESH folder.
3. Copy the subroutines to CASE/SRC folder.
4. Pop up the GUI and configure your simulation. Save the xml file under CASE/DATA. (if using GUI)
5. Configure any subroutines and test compilation.
6. Run or submit your case.
7. Grab the data under CASE/RESU and postprocess it.

Code_Saturne walkthrough

Creating the work directory

Code_Saturne needs specific directory structure. It is organised in STUDY/CASE hierarchy. To create the necessary structure use the command:

```
code_saturne create -s STUDY1 -c CASE1 -c CASE2
```

which creates a study named STUDY1 with cases CASE1 and CASE2. Cases can be added by using:

```
code_saturne create -c CASE3
```

More info by typing:

```
code_saturne create -h
```

Code_Saturne walkthrough

Using the GUI

To use the graphical interface issue the command

code_saturne gui

Create a new file.

Go through the folders and fill the necessary information for the case (physical properties, boundary conditions, numerical options, etc.).

Save an .xml file (by default in the data directory).

Not all modules are available on the GUI. Sometimes use of the FORTRAN files is required as complement.

If the GUI is not used, all necessary parameters have to be filled in the FORTRAN subroutines.

TIPS: when in the GUI, go to “Mesh quality criteria” and run “check mesh”. You can use the check_mesh.log to import the boundary conditions by clicking the button “Import from Preprocessor listing”.

Code_Saturne walkthrough

Run your case

To run your case, go under the CASE directory and issue

code_saturne run -p <xml file> -n <nprocs>

- -p (--param) must follow the name of the parameter file (generated by the GUI).
- -n must follow the number of processors to use for the run.

The calculation will run on a directory with the date under RESU.

The listing file contains all input parameters and information on the variables at each iteration.

Once the calculation is finished, all relevant files are copied to the RESU directory.

Code_Saturne walkthrough

In case of problems…

In case of problems, the following files can give an idea of what was the problem:

- compilation log: if the run stopped at compilation, then here the compilation error will be found.

- check the preprocessor log: if the run stopped when preprocessing, the preprocessor log will contain information where it stopped and why.

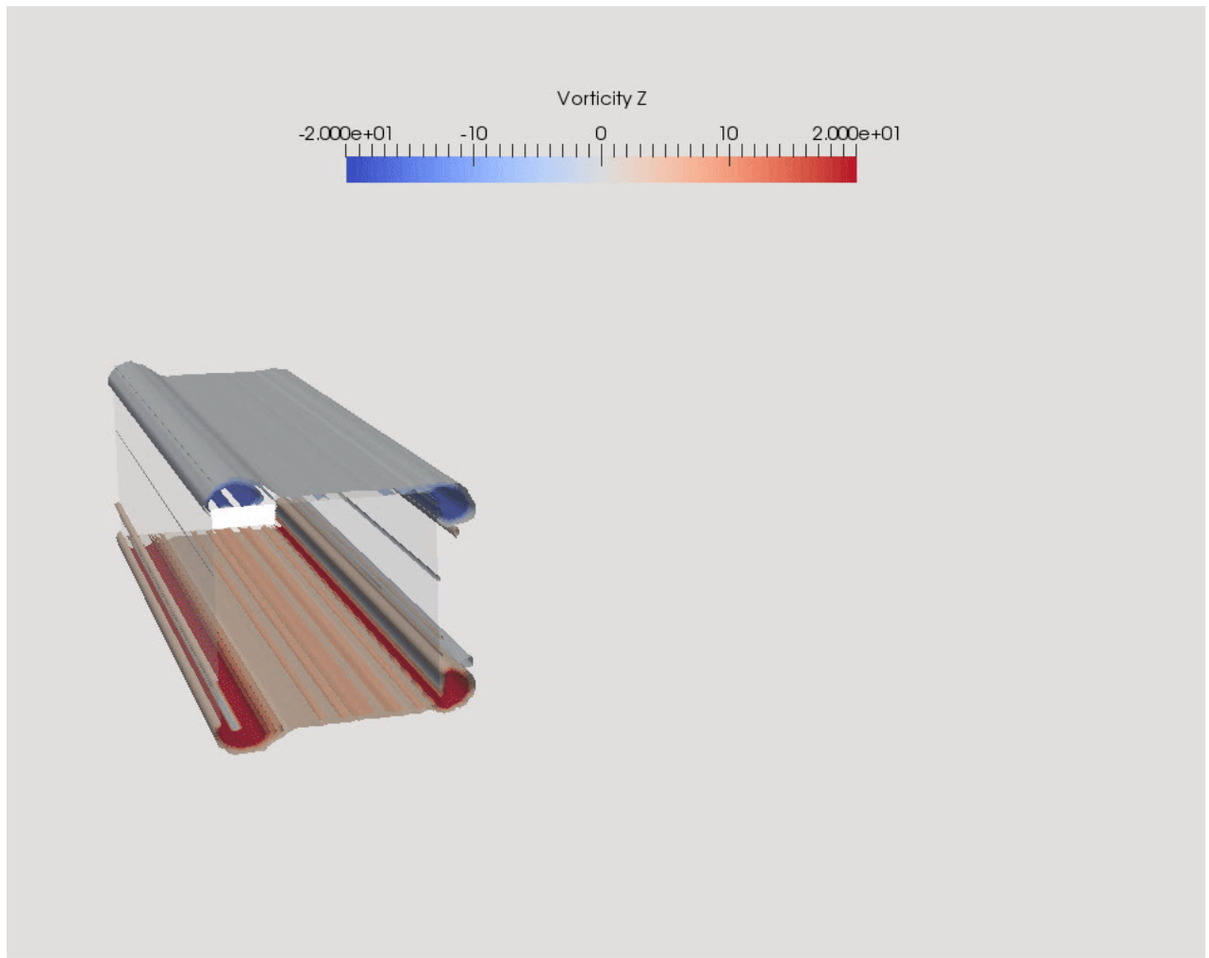
- listing file: if the run stopped while computing, the listing file will have information why it stopped.

Examples

Code_Saturne examples

Square cylinder – LES

LES case includes periodic boundary conditions for the front and back patches. The interior is split in 20 different planes, thus the mesh is fully 3D.

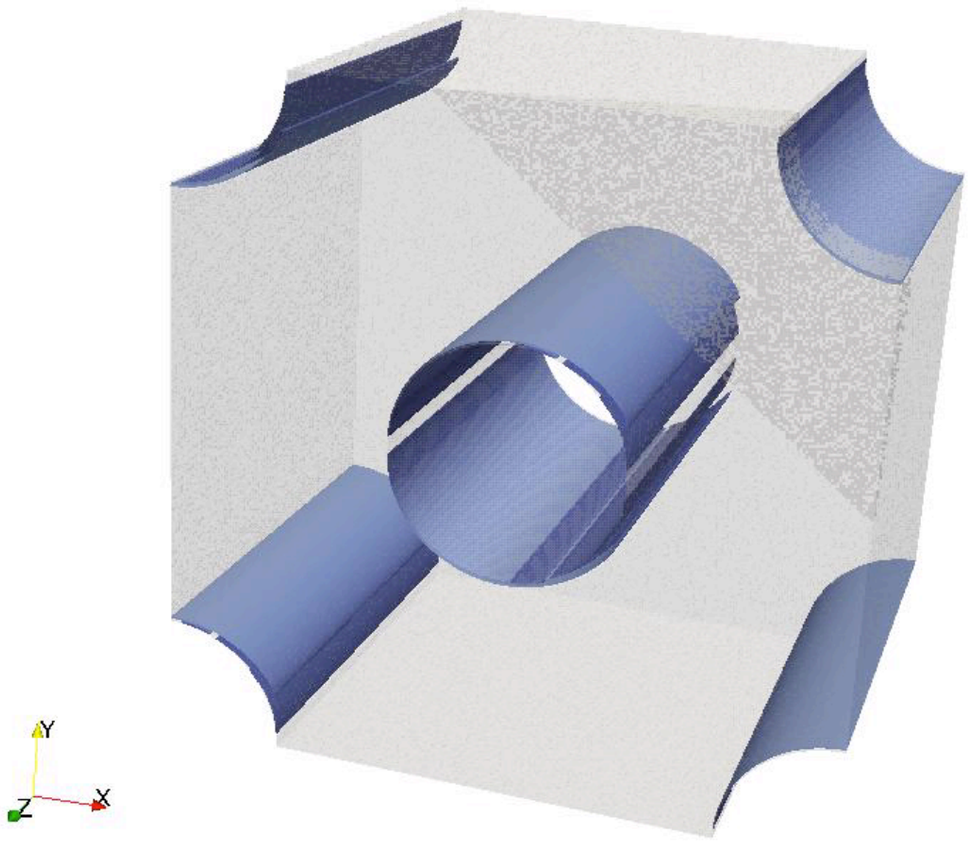


LES Smagorinsky isosurfaces animation.

Code_Saturne examples

Tube bundle – LES

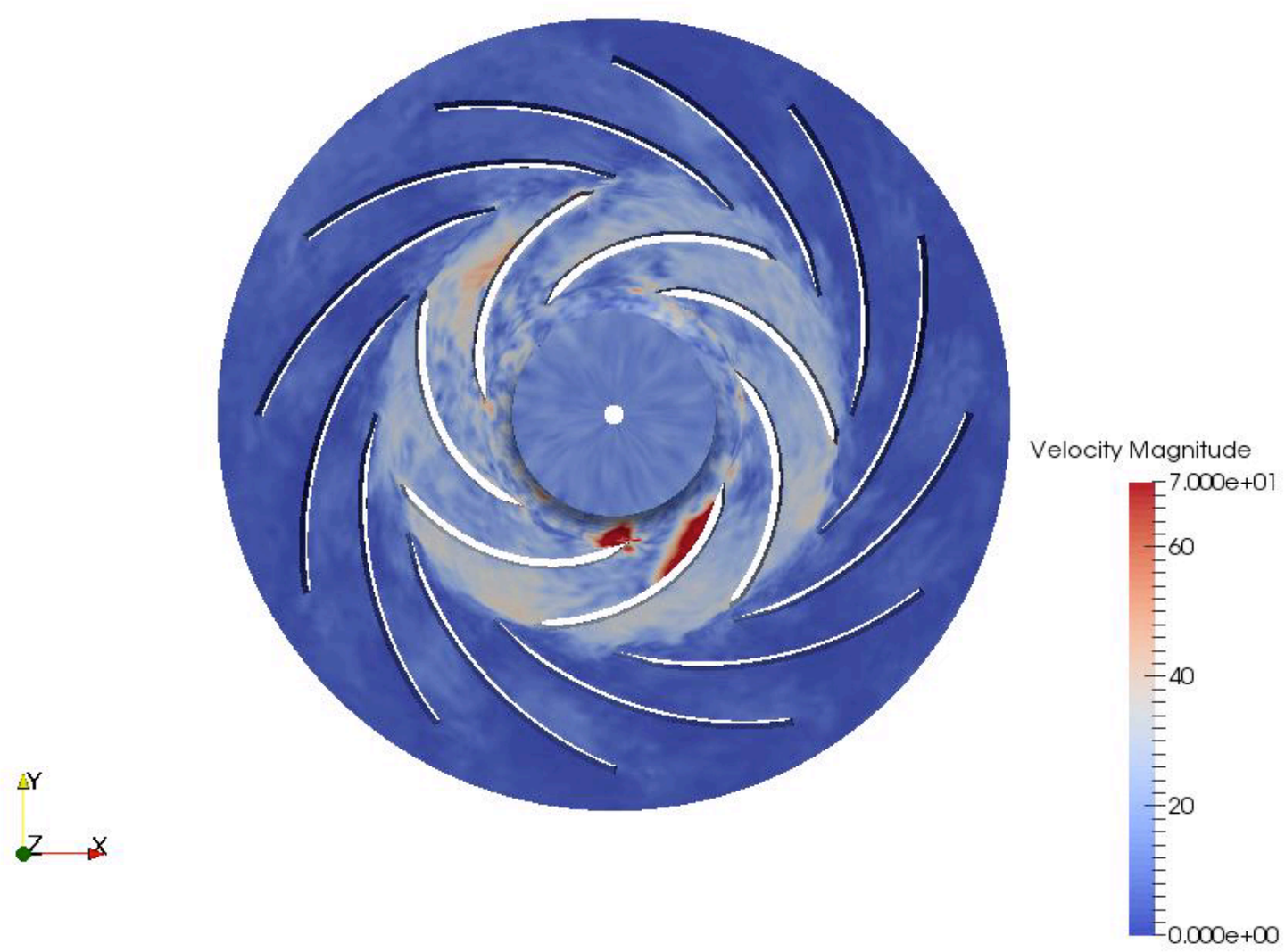
LES WALE simulation of a tube bundle with periodic boundary conditions. Simulation ran on a power8 cluster under 64 processors for 18 hours. Isocontours of vorticity are showed being colored by the magnitude of the velocity.



Code_Saturne examples

Centrifugal pump – radial compressor

Simulation of a centrifugal pump using the turbomachinery module of Code_Saturne. Simulation ran overnight on MareNostrum supercomputer under 32 cpu's.



Tutorials

Code_Saturne tutorials

Overview

This section features a number of tutorials to help the user familiarize with the *Code_Saturne* environment.

- Driven cavity.
- Channel flow with square cylinder.
- Square cylinder on a turbulent flow.

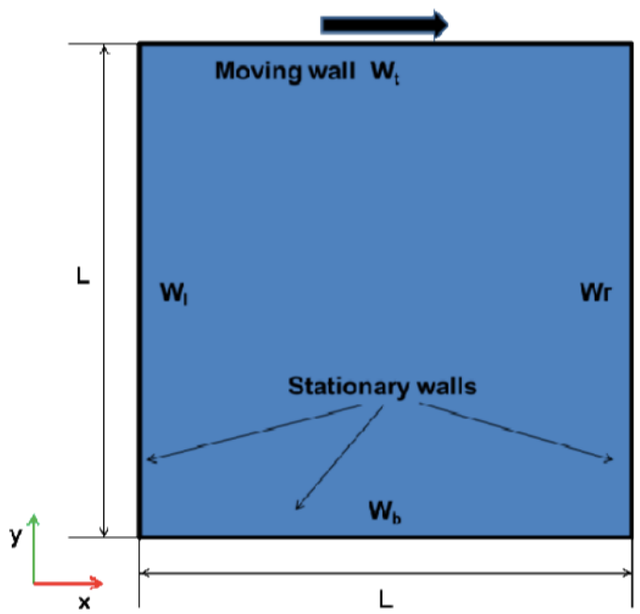
Apart from this tutorials, the reader is encouraged to check the Code_Saturne webpage for further tutorials and University of Manchester (http://cfd.mace.manchester.ac.uk/twiki/bin/view/Saturne/WebHome).

Code_Saturne tutorials

Driven cavity

This simple case is performed for validating the code against a well-known example. In this case, the shear driven cavity flow case has been set up following a tutorial [1].

The case is run for several structured meshes (50x50, 75x75, 100x100, 128x128) at Re = 1000 (laminar case) with the top wall sliding at 1.0 m/s. The case is run for 400 iterations at a time step of 0.1 (40 time units).



Description of the case. [1]

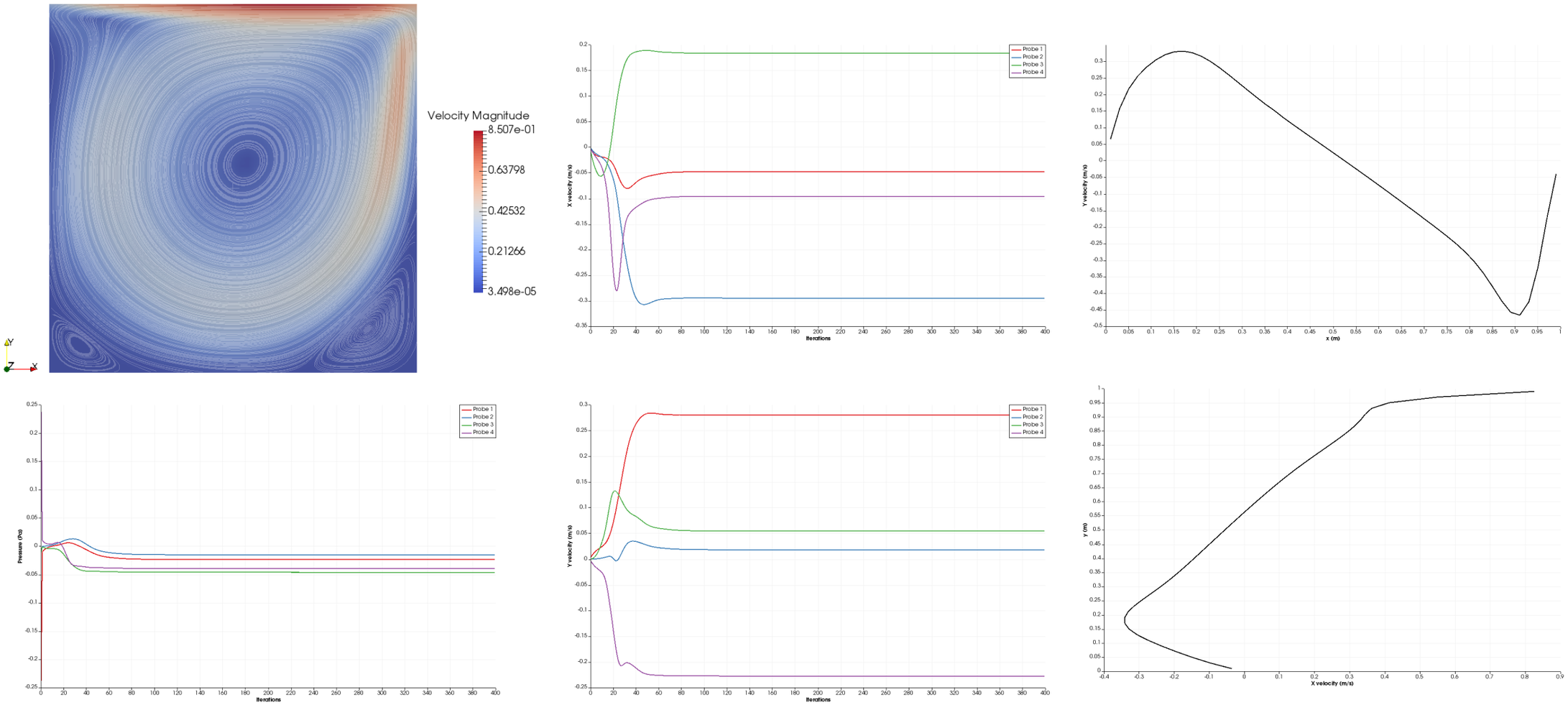
Probe points location. [1]

n	x	y	z
1	0.25	0.5	0
2	0.5	0.25	0
3	0.5	0.75	0
4	0.75	0.5	0

Code_Saturne tutorials

Driven cavity – Results

The steady case is the case presented in the tutorial [1]. The results are obtained using a structured hexa mesh (50x50). The obtained results are identical of that on the tutorial.



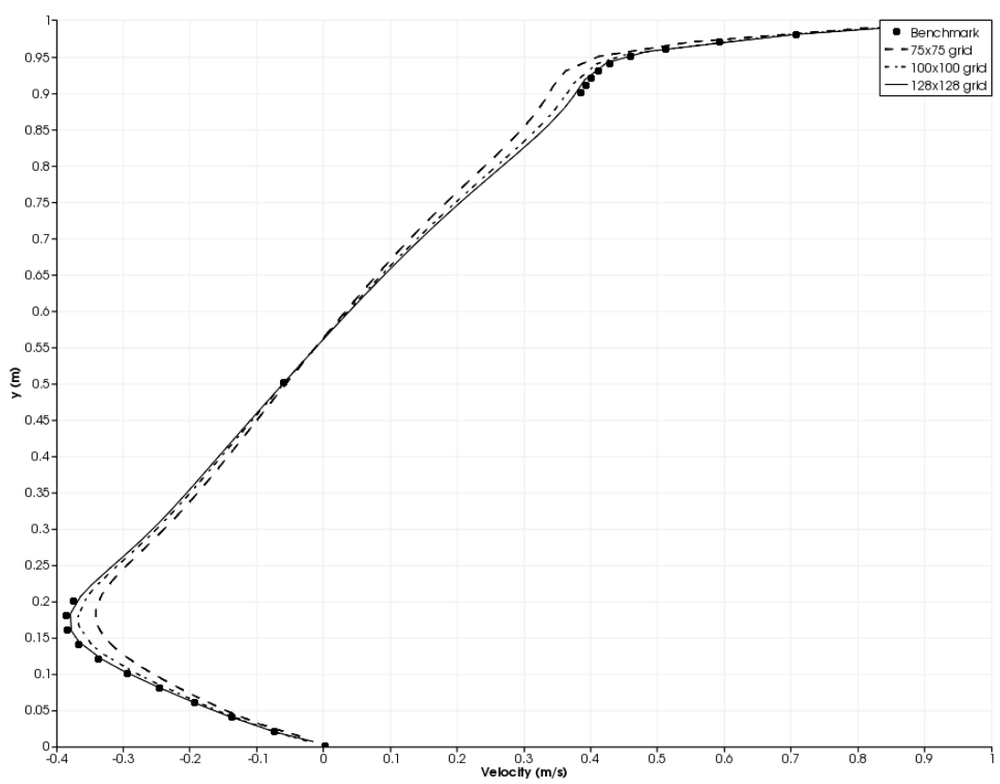
Driven cavity results for structured hexa mesh.

Code_Saturne tutorials

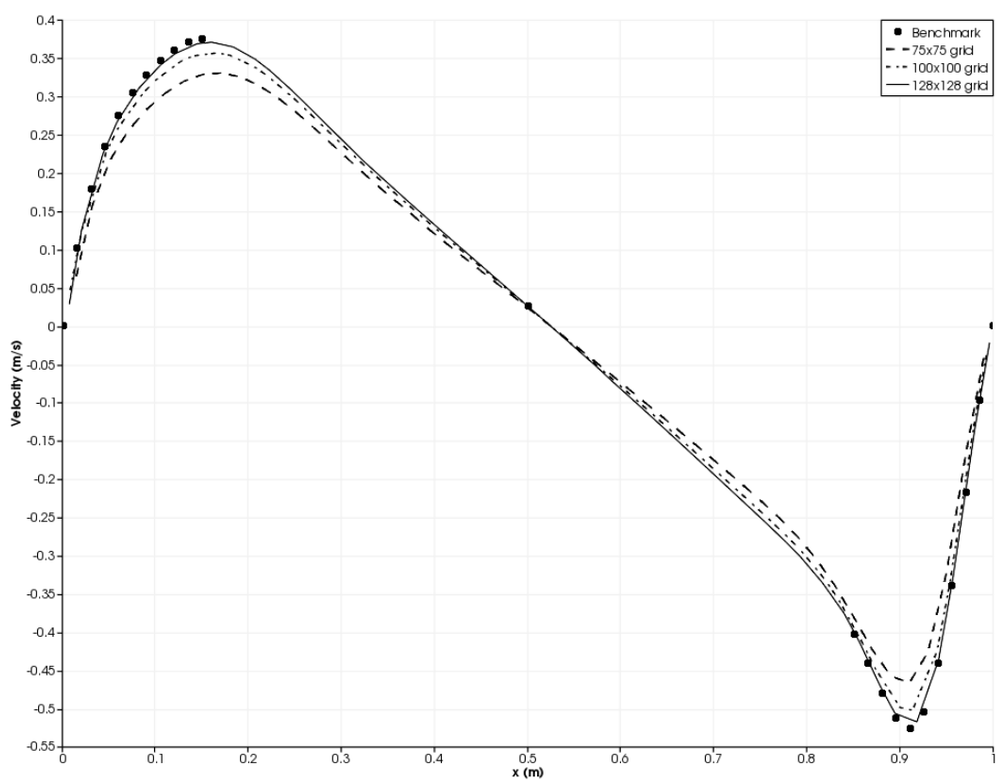
Driven cavity – Validation

The results have also been compared with a benchmark case [2]. Here U and V profiles at end of computation are shown for meshes of 75x75, 100x100 and 128x128.

The benchmark case is performed with a grid of 128x128.



U velocity profile at center line for different meshes.



V velocity profile at center line for different meshes.

Code_Saturne tutorials

Driven cavity – References

[1] EDF R&D. "*Code_Saturne* version 3.0 tutorial. Shear driven cavity flow" *Code_Saturne* documentation (2013).

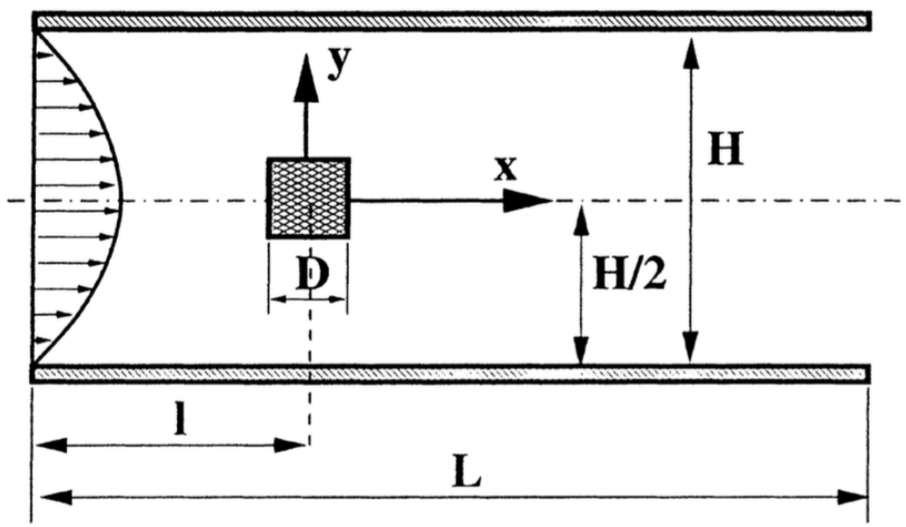
[2] Erturk, E., Corke, T. C., & Gökçöl, C. (2005). Numerical solutions of 2-D steady incompressible driven cavity flow at high Reynolds numbers. International Journal for Numerical Methods in Fluids, 48(7), 747-774.

Code_Saturne tutorials

Channel flow with square cylinder

A 2D flow is considered on a plane channel with an obstacle of size D inside. Blockage ratio B is taken as 1/8 and the total length of the channel set to L/D = 50. The inlet is situated at a distance l = L/4 from the center of the cylinder.

The following monitoring points and velocity profiles are considered:



Description of the geometry. [1]

Point	Location	x (m)	y (m)
1	1.5D	1.5	0
2	1.5D	1.5	-0.5
3	1.5D	1.5	-1
4	2.0D	2	0
5	2.0D	2	-0.5
6	2.0D	2	-1
7	4.0D	4	0
8	4.0D	4	-0.5
9	4.0D	4	-1

Location of monitoring points and profile lines

Profile	Location	Measure
Cylinder	x = 0	u, v
Near Wake	x = 4	u, v
Far Wake	x = 8	u, v
Symmetry	y = 0	u, v

Code_Saturne tutorials

Channel flow – computational domain

Mesh is comprised of uniform hexa structured cells. The square rod diameter D is 1 and the blockage ratio is 0.125, therefore, H is 8. Several meshes are done:

- Mesh 1: similar to FVM 500x80 [45800 cells]
- Mesh 2: similar to FVM 400x240 [147200 cells]

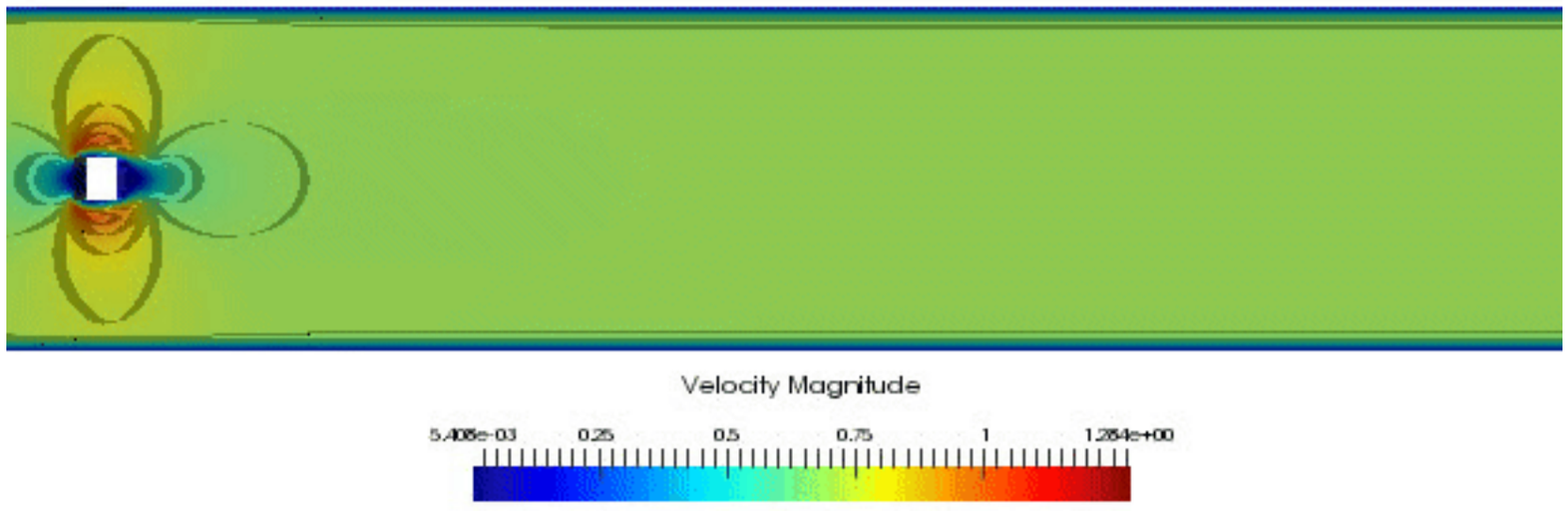
Method	Total number of CV	Grid type	Number of CV on cylinder	Smallest CV at cylinder/D	Maximum stretching factor
FVM	500×80	Equidistant	10	0.1	1
FVM	400×240	Non-equidistant	80	0.01	1.03
FVM	560×340	Non-equidistant	100	0.01	1.02

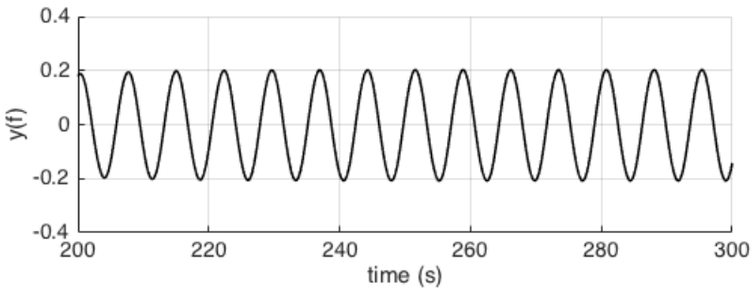
Meshes comparison. [1]

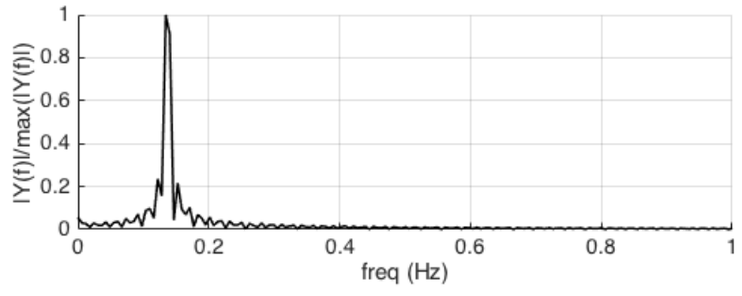
Code_Saturne tutorials

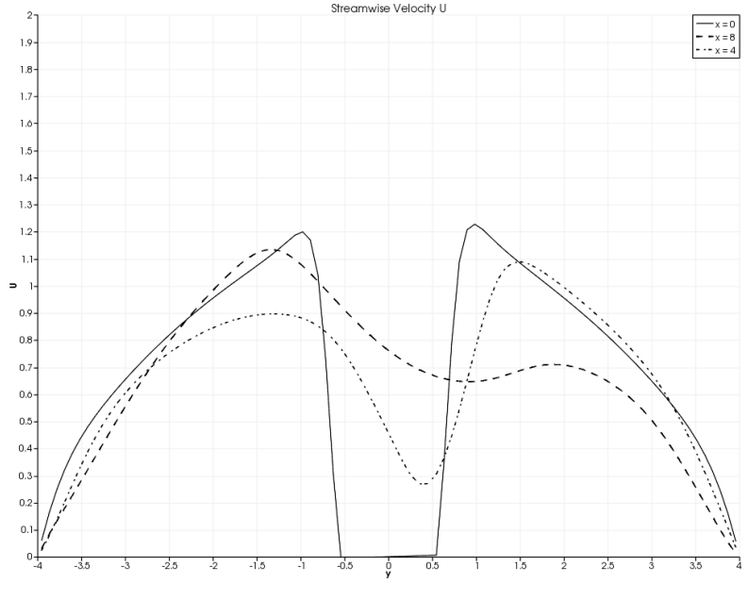
Channel flow – mesh 1 results

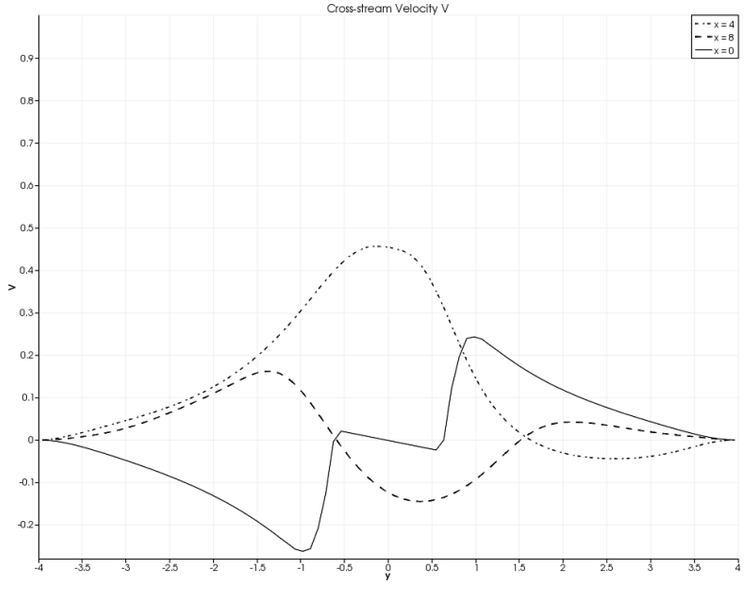
Results are obtained for Re = 100. The obtained Strouhal number is of 0.1343 and the C_D of 1.369.

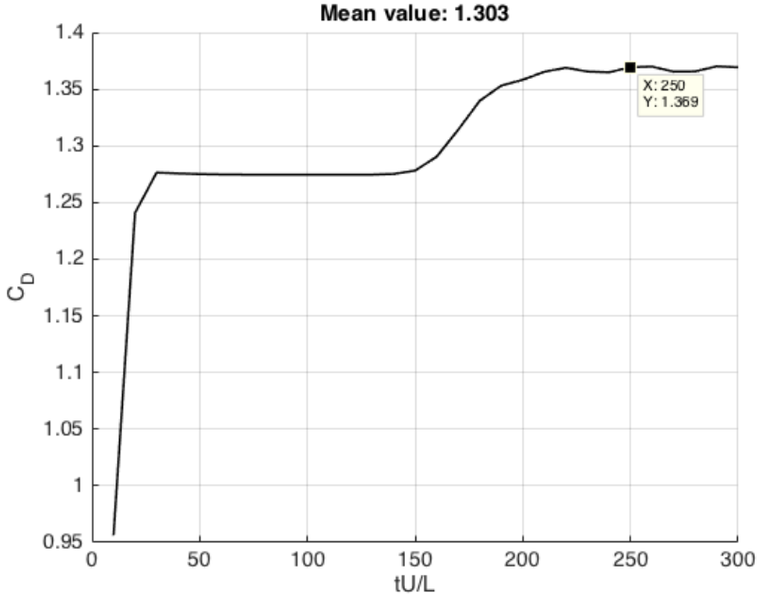








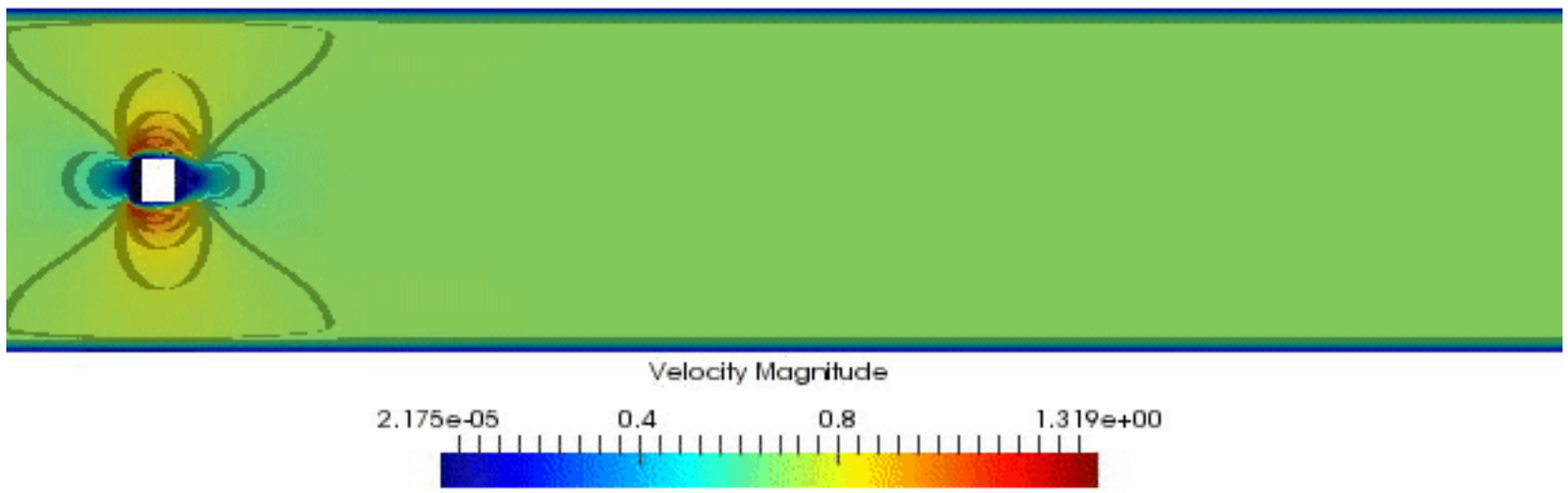


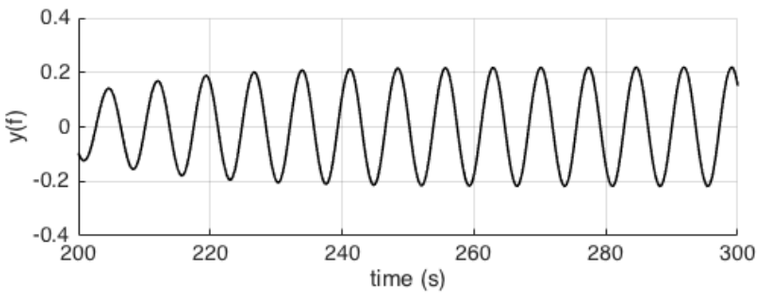


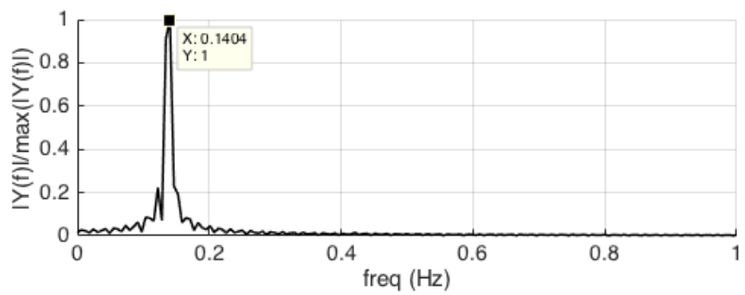
Code_Saturne tutorials

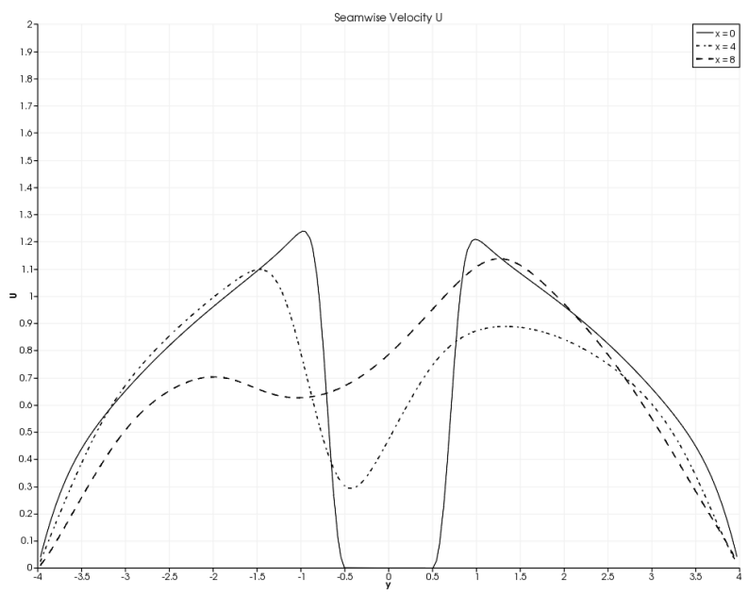
Channel flow – mesh 2 results

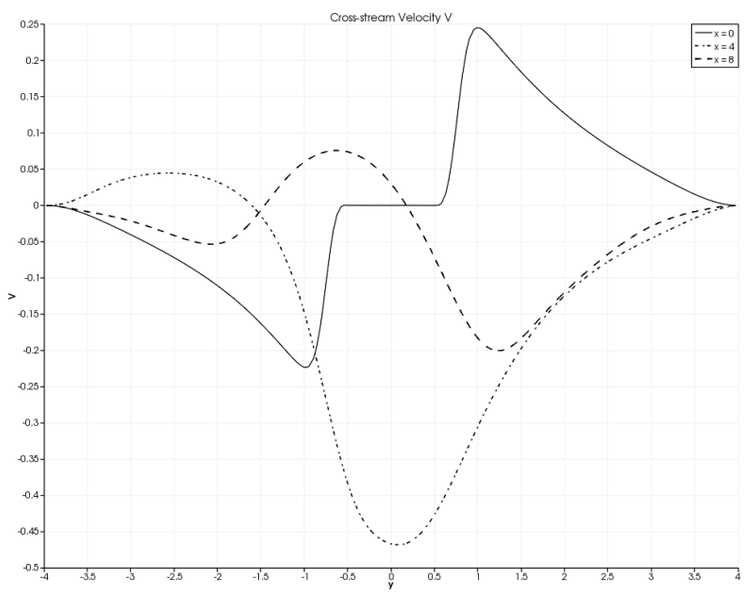
Results are obtained for Re = 100. The obtained Strouhal number is of 0.1404 and the C_D of 1.37.

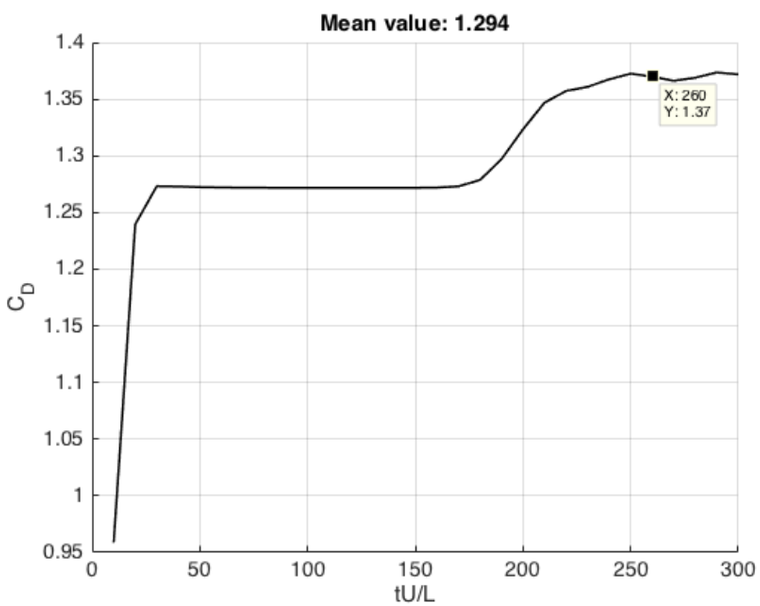








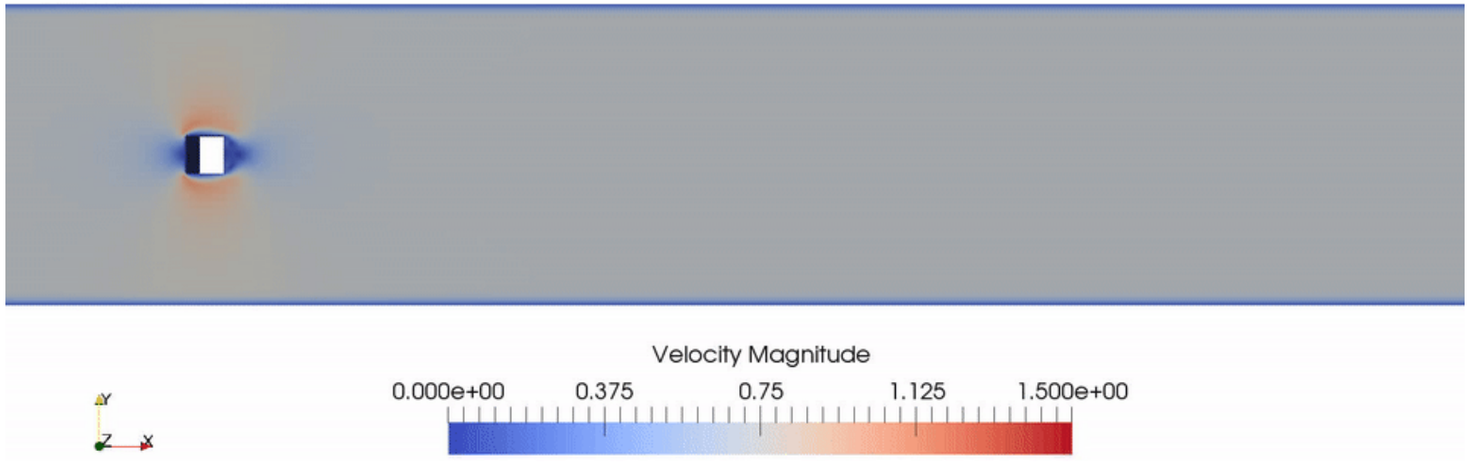


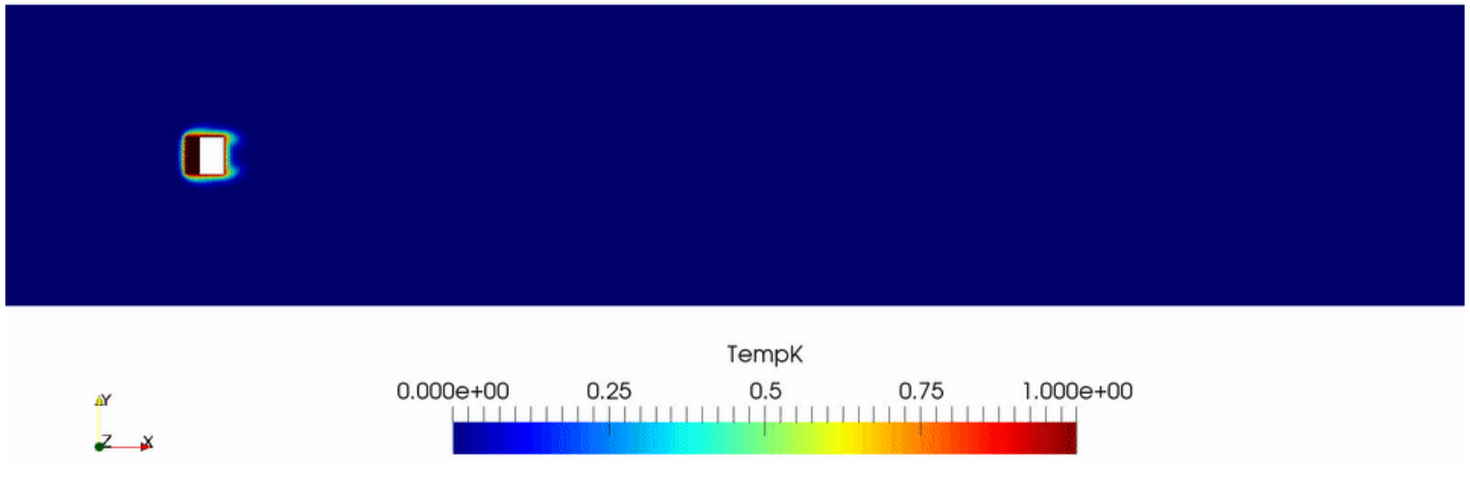


Code_Saturne tutorials

Channel flow – Passive scalar

Temperature is activated as a “passive scalar”, i.e., it is only being transported. Therefore, it actuates as a tracer for the fluid. This is done by activating the thermal module but setting the Rayleigh number to zero, i.e., density remains constant.

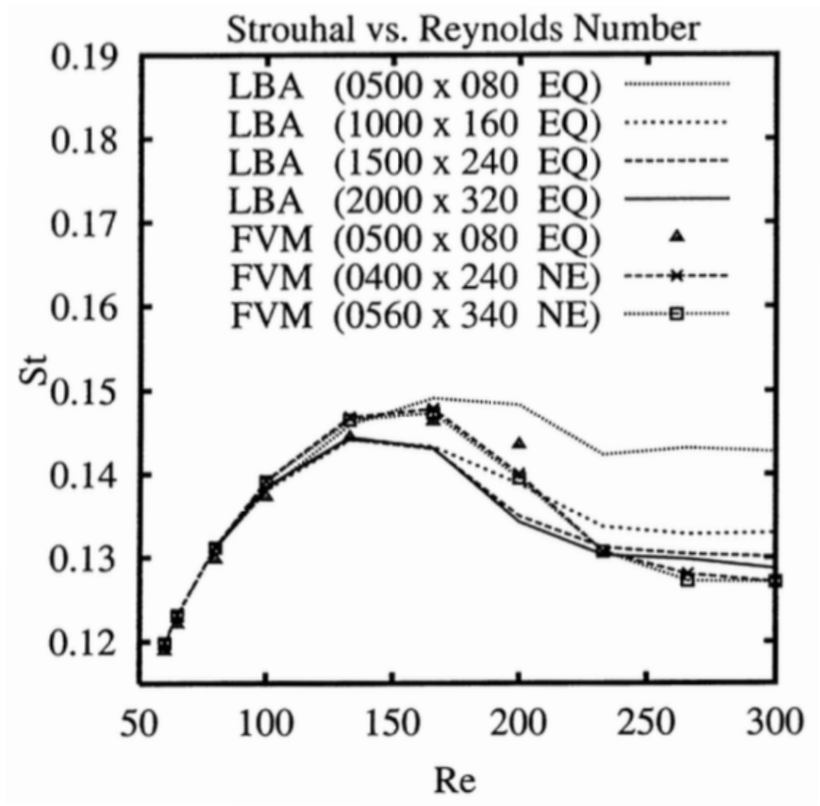




Code_Saturne tutorials

Channel flow – Results discussion

Obtained results for mesh 1 successfully follow the literature. Strouhal number for Re = 100 is near 0.14 and CD around 1.4 (for the first mesh) and around 1.35 for the other meshes.



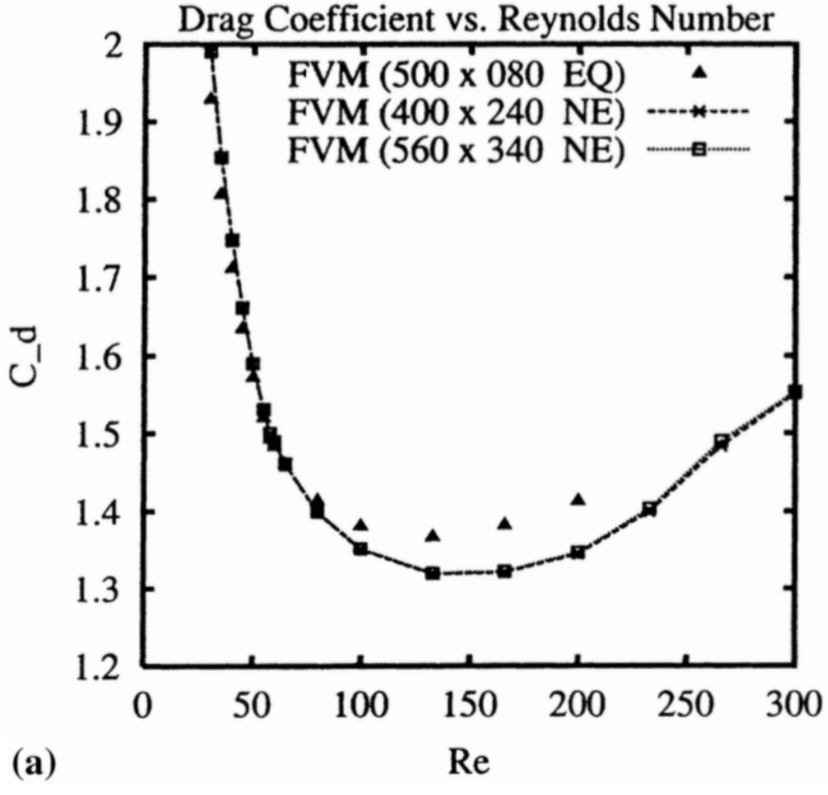
Strouhal number variation with Reynolds. [1]

Reynolds:

$$Re = \frac{\rho v_{max} D}{\mu}$$

Strouhal:

$$St = \frac{\omega d}{2\pi U_m} = \frac{f d}{U_m}$$



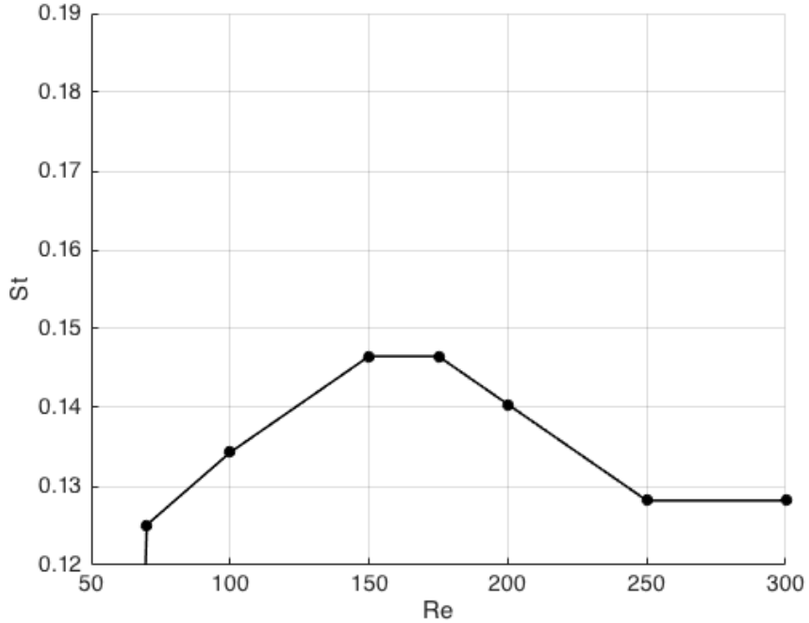
C_D variation with Reynolds. [1]

Code_Saturne tutorials

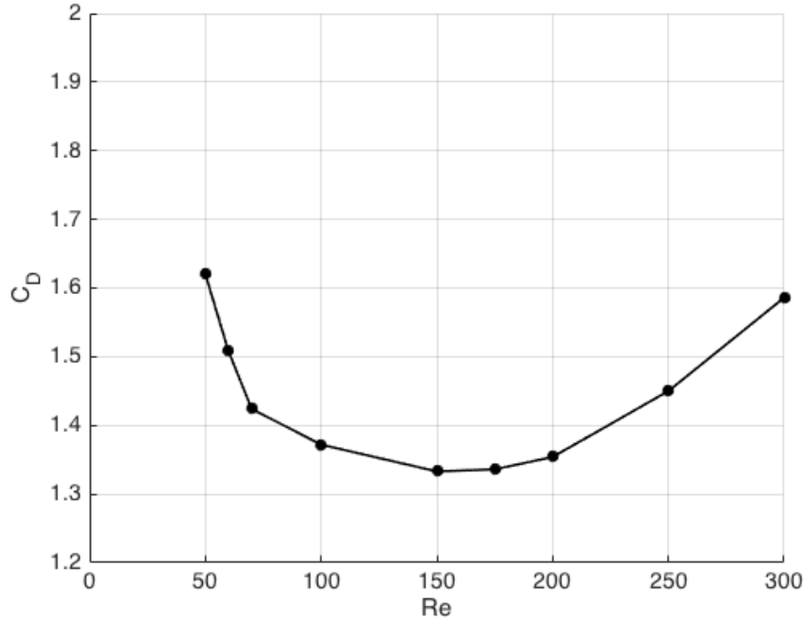
Channel flow – Results comparison

Results of Strouhal and drag coefficient are computed for a variety of Reynolds number and compared with literature.

Re	St	C_D
50	0.0000	1.620
60	0.0000	1.508
70	0.1251	1.424
100	0.1343	1.372
150	0.1465	1.333
175	0.1465	1.336
200	0.1404	1.354
250	0.1282	1.450
300	0.1282	1.586



Strouhal number variation with Reynolds.



C_D variation with Reynolds.

Code_Saturne tutorials

Channel flow – References

[1] Breuer, M., et al. "Accurate computations of the laminar flow past a square cylinder based on two different methods: lattice-Boltzmann and finite-volume." International Journal of Heat and Fluid Flow 21.2 (2000): 186-196.

Code_Saturne tutorials

Square cylinder on a turbulent flow

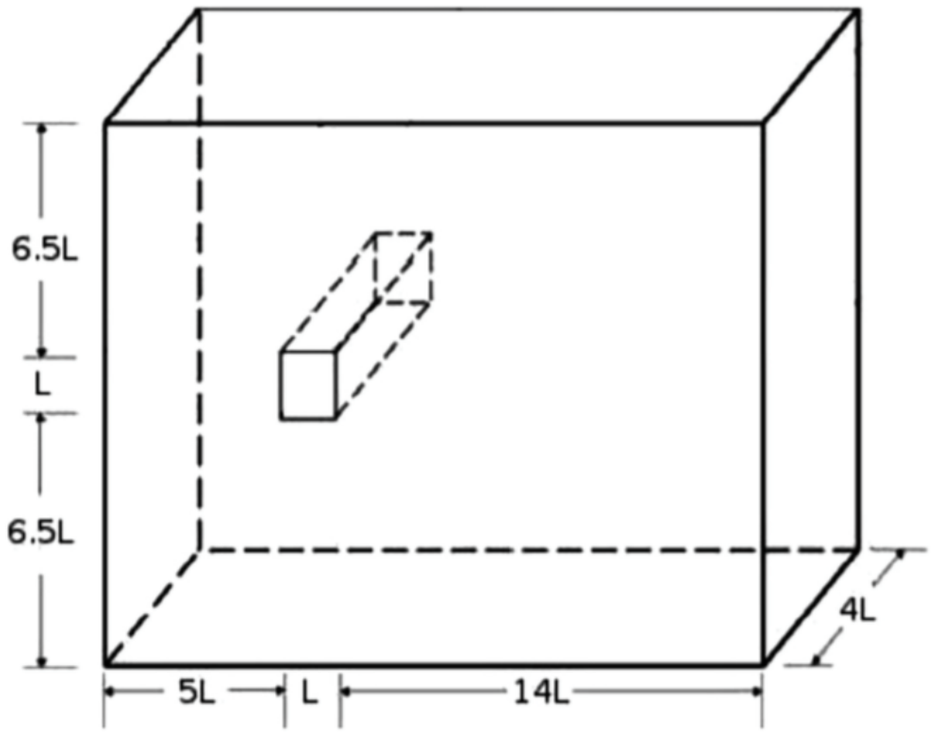
The case has been run for a number of two different meshes and turbulence models at Re = 22000 [1]. Buffer zone is considered to start at 13L and simulations are performed until 100 TU.

Meshes:

- Unstructured fully tetra mesh.
- Unstructured fully hexa mesh.

Turbulence models:

- URANS, k-omega SST.
- URANS, Spalart-Allmaras.
- LES, Smagorinsky.



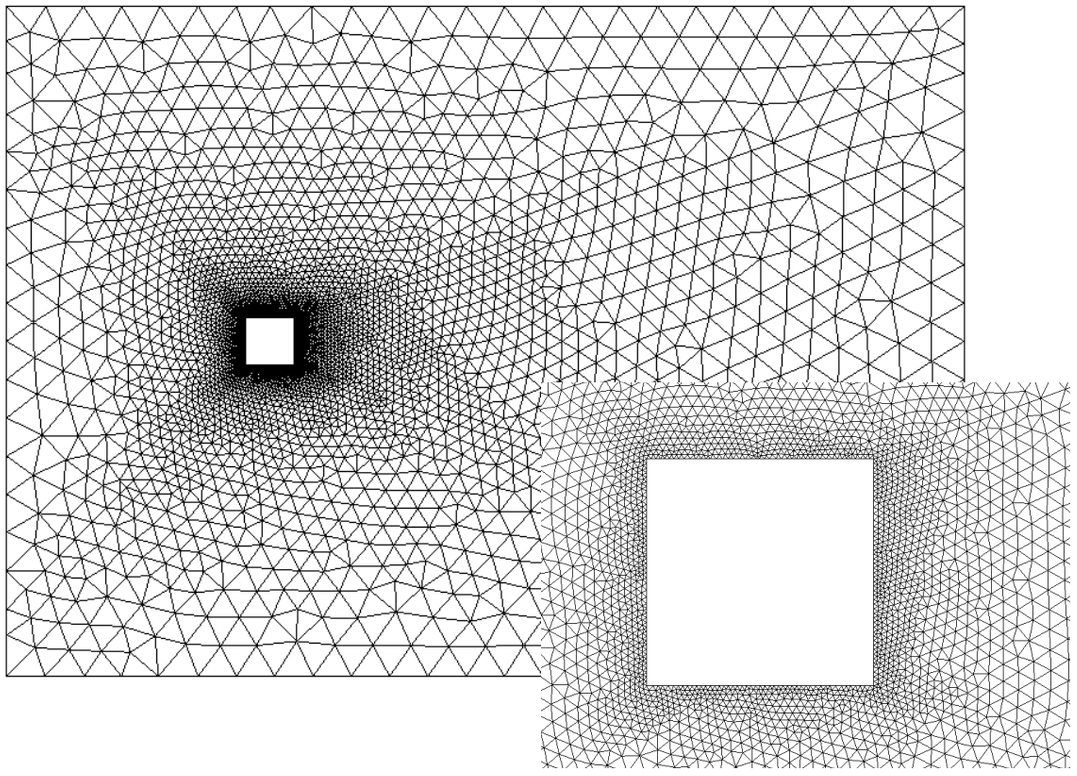
Overview of the computational domain. [1]

Code_Saturne tutorials

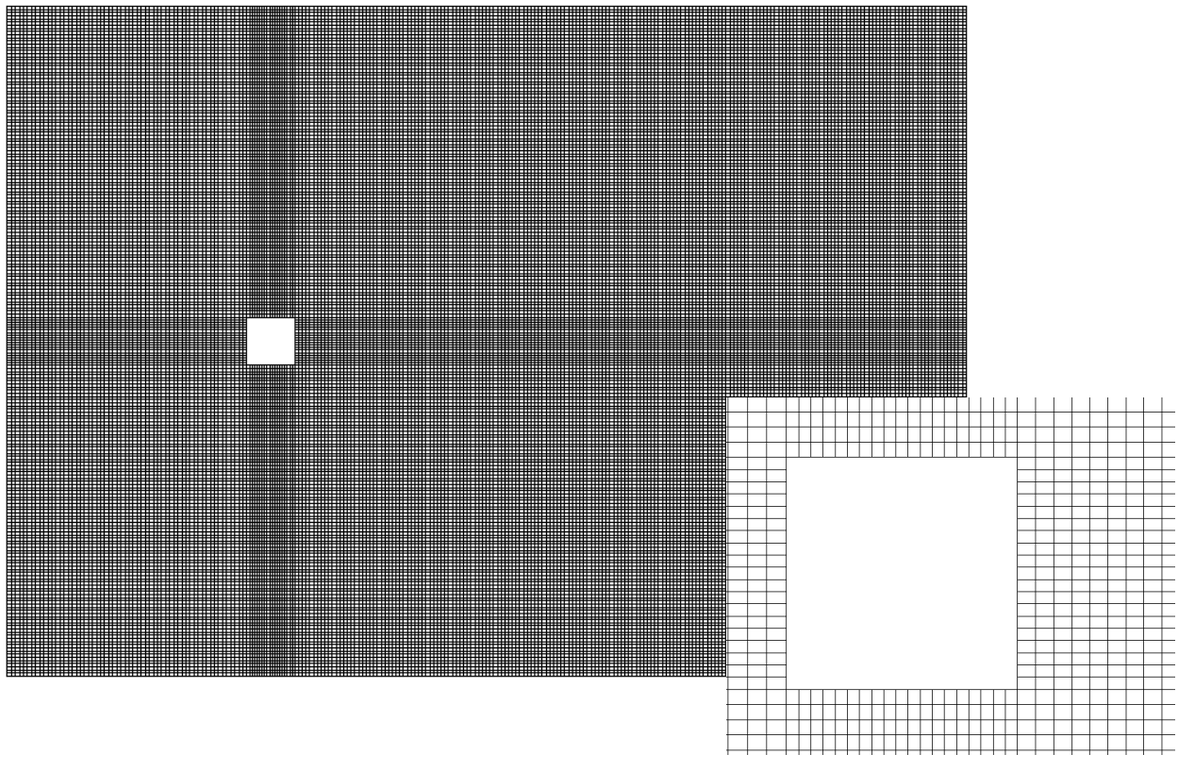
Square cylinder – mesh

The computational domain is modeled in ICEM as 2D domain (extruded). Two meshes are considered for comparing:

- Unstructured fully tetra. [8259 cells]
- Structured hexa. [55408 cells]



Unstructured tetra mesh detail.

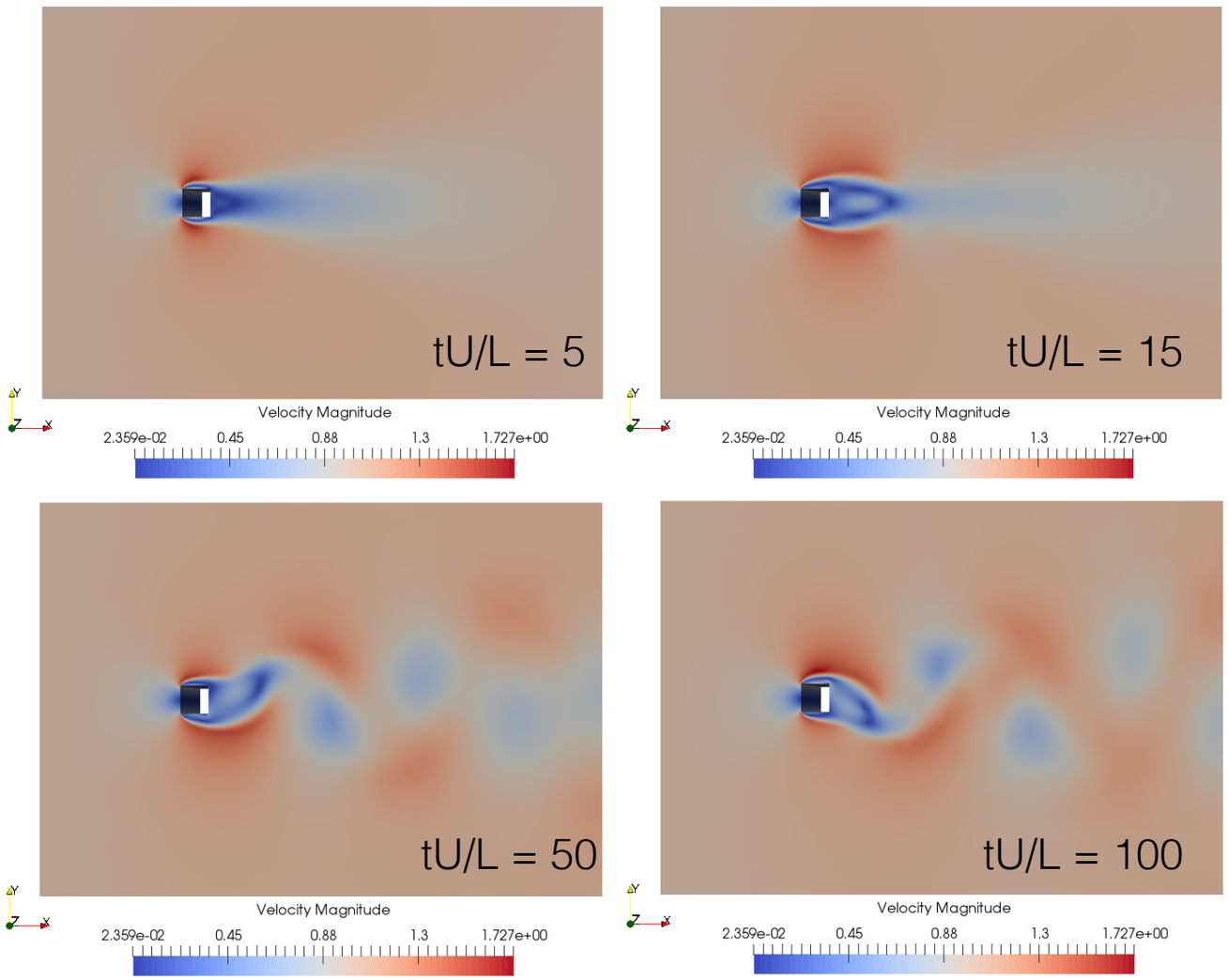


Structured hexa mesh detail.

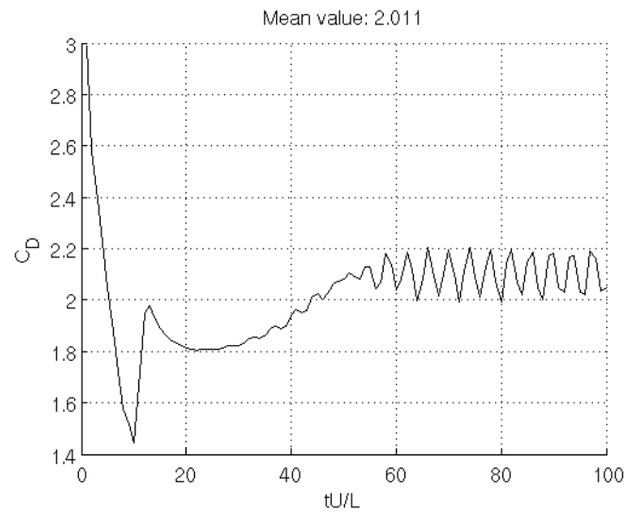
Code_Saturne tutorials

Square cylinder – Spalart-Allmaras

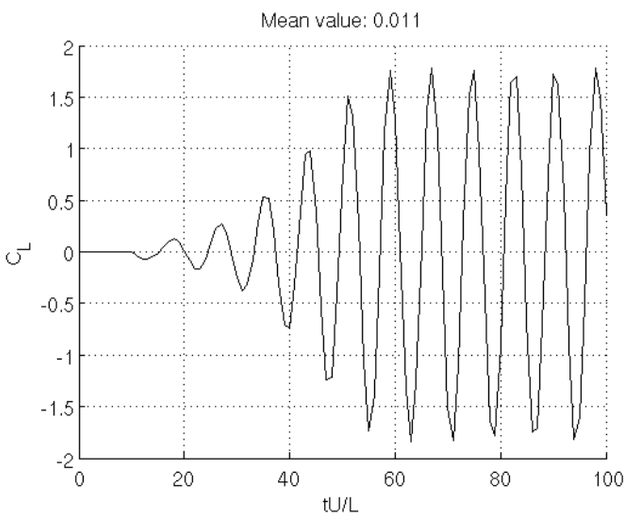
Spalart-Allmaras model seems to be quite unstable in *Code_Saturne*. While it provides accurate results for this particular case, in further studied cases diverges.



RANS Spalart-Allmaras flow at several timesteps.



RANS Spalart-Allmaras C_D variation.

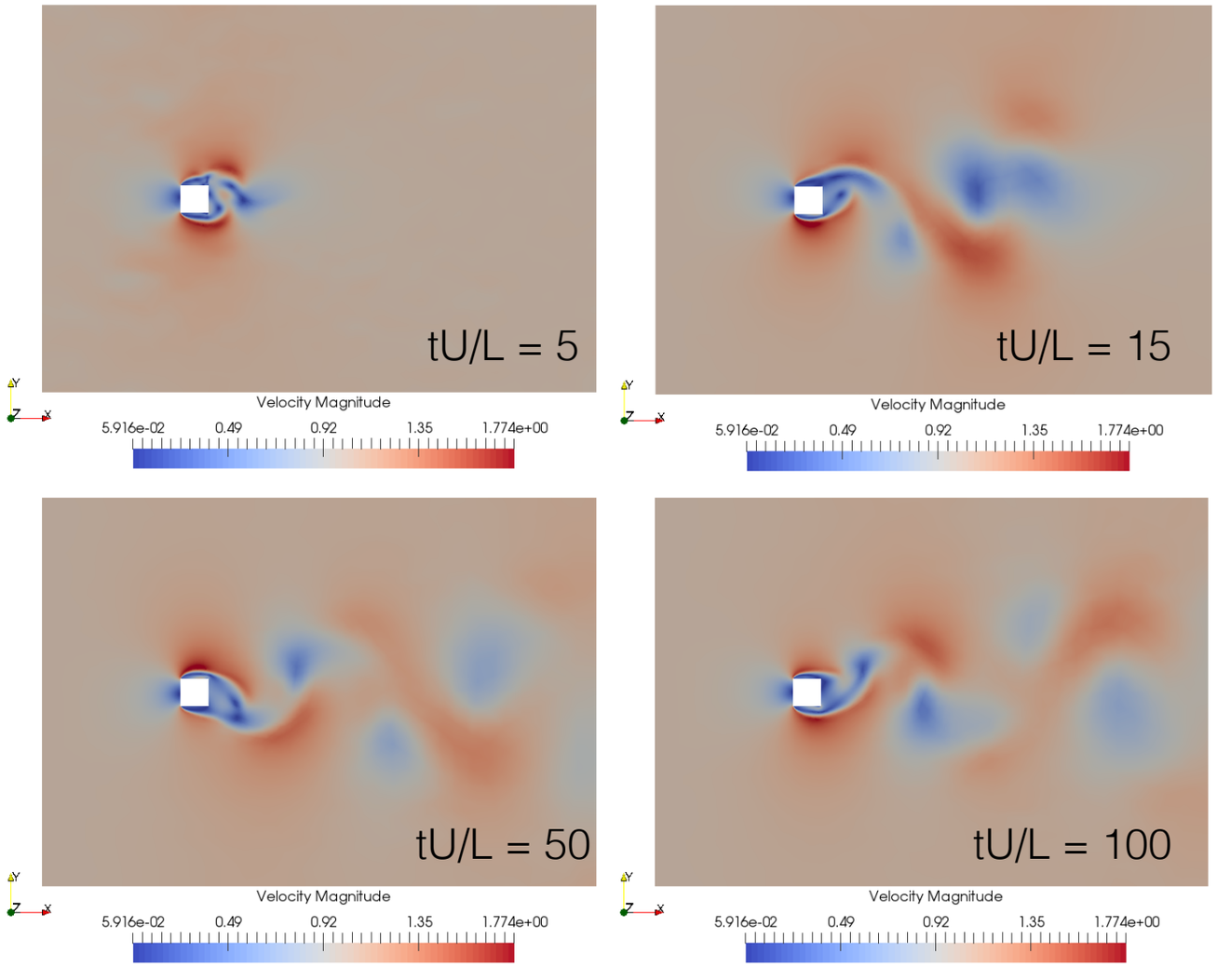


RANS Spalart-Allmaras C_L variation.

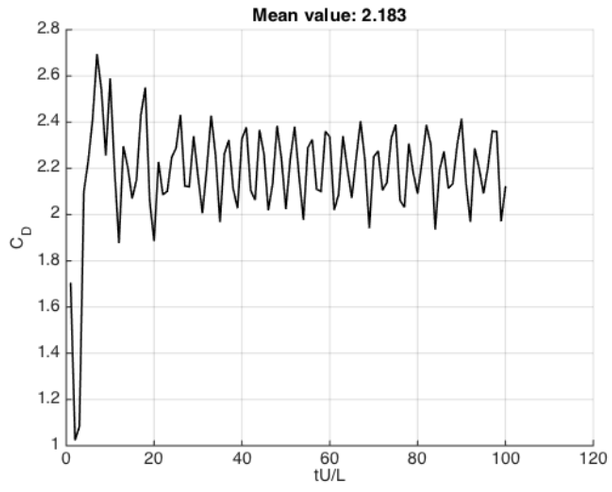
Code_Saturne tutorials

Square cylinder – k-omega SST

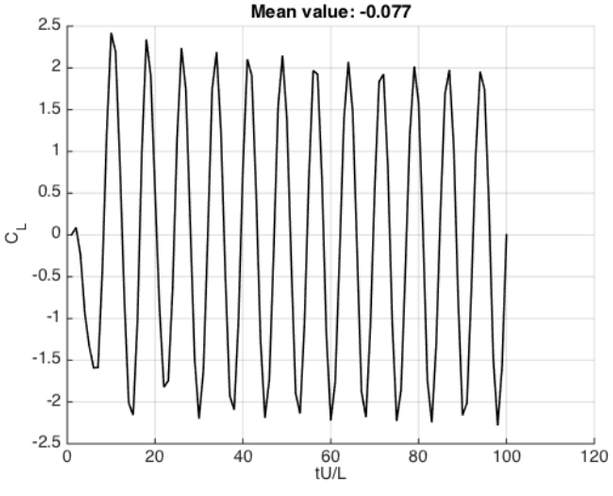
k-omega SST model is recommended for production runs in *Code_Saturne*. It presents more stability than Spalart-Allmaras, therefore, it becomes the model used for further computations.



RANS k-omega SST flow at several timesteps.



RANS k-omega SST C_D variation.

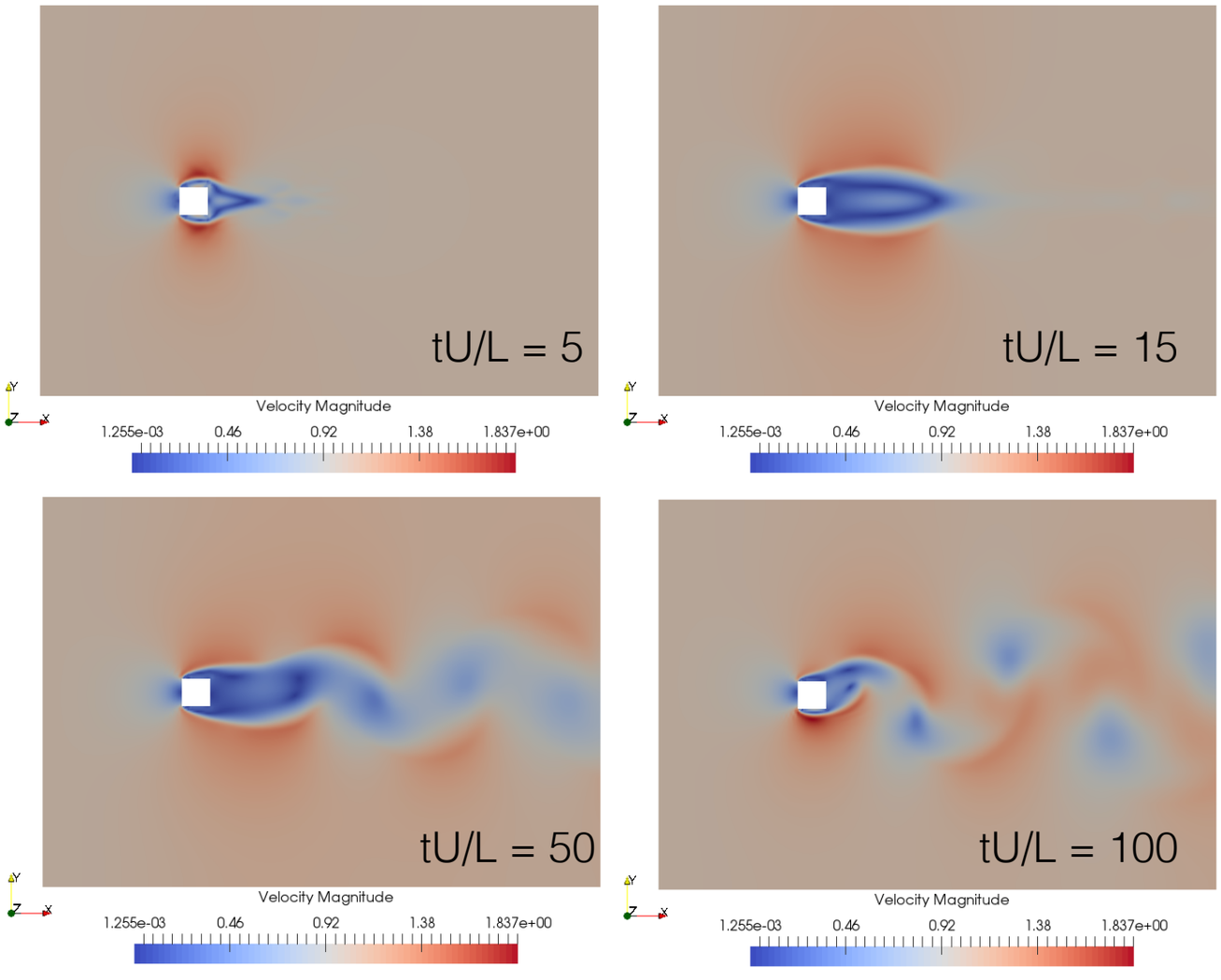


RANS k-omega SST C_L variation.

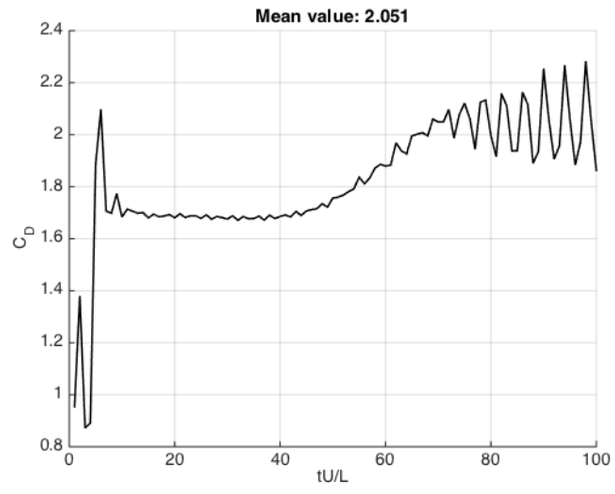
Code_Saturne tutorials

Square cylinder – k-omega SST (hexa)

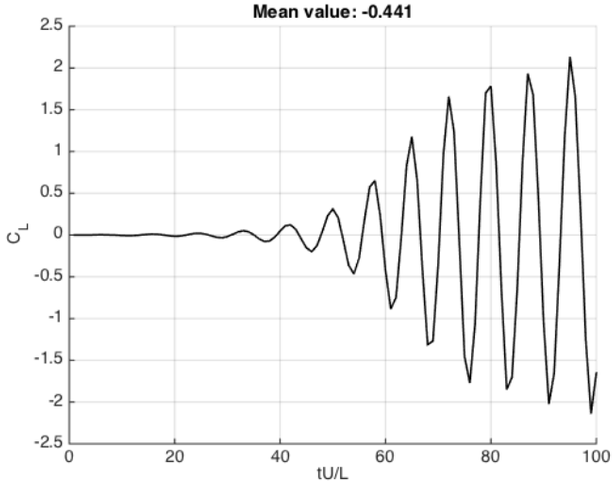
Hexa mesh is run using the k-omega SST. Although shedding appears, it does in a later timestep. However, the C_D and C_L results agree with the previous simulations.



RANS Spalart-Allmaras flow at several timesteps.



RANS Spalart-Allmaras C_D variation.

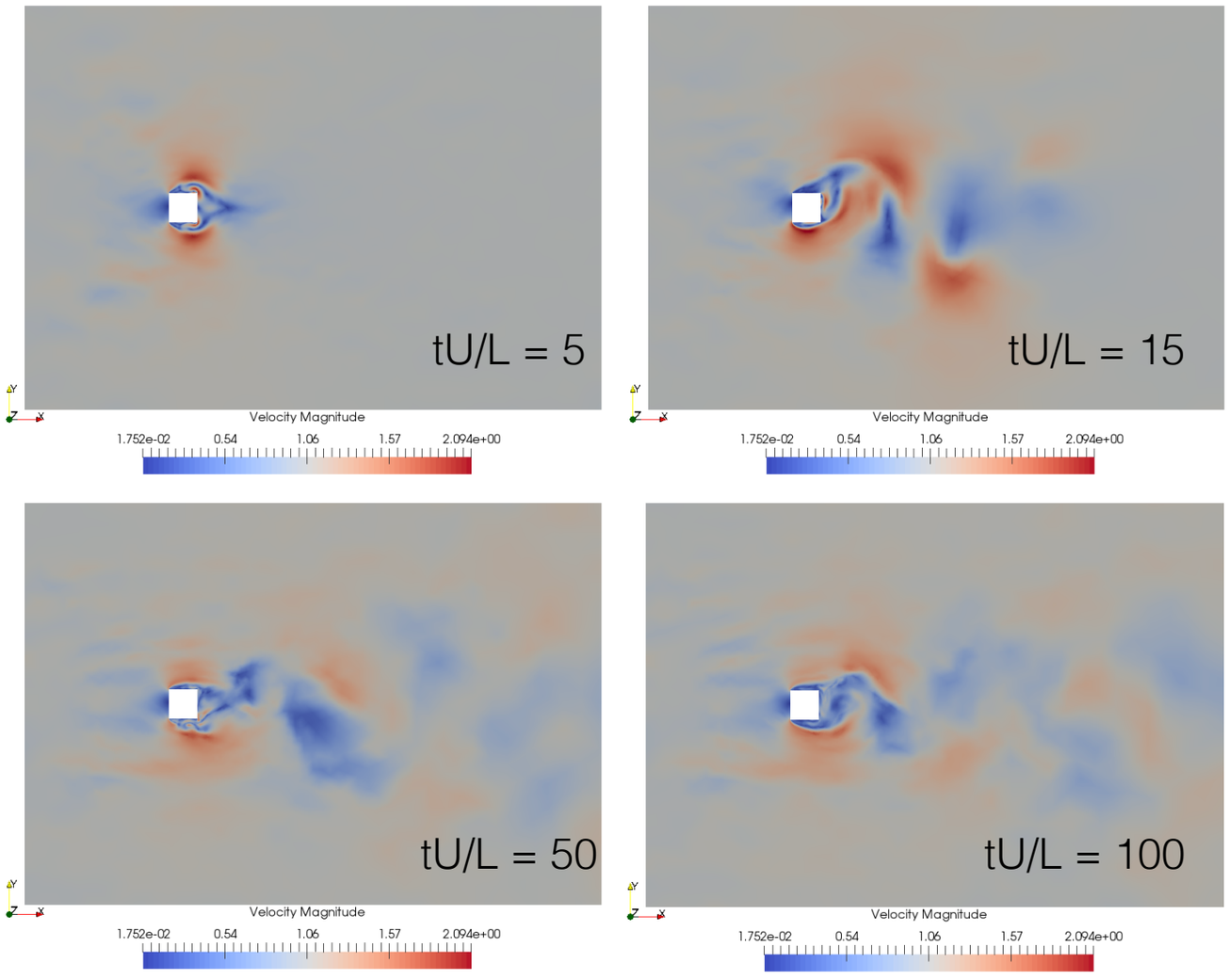


RANS Spalart-Allmaras C_L variation.

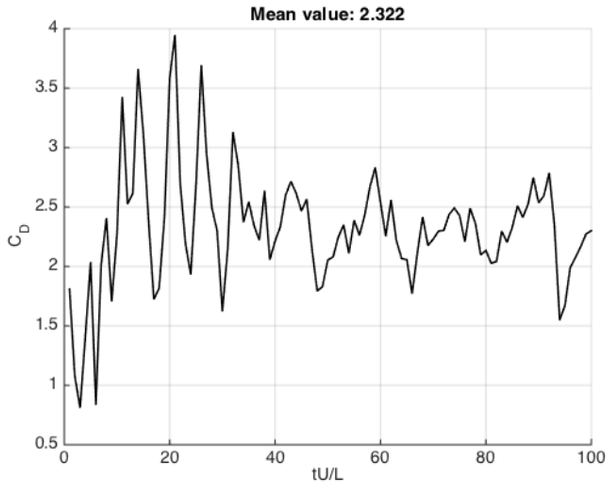
Code_Saturne tutorials

Square cylinder – LES-Smagorinsky

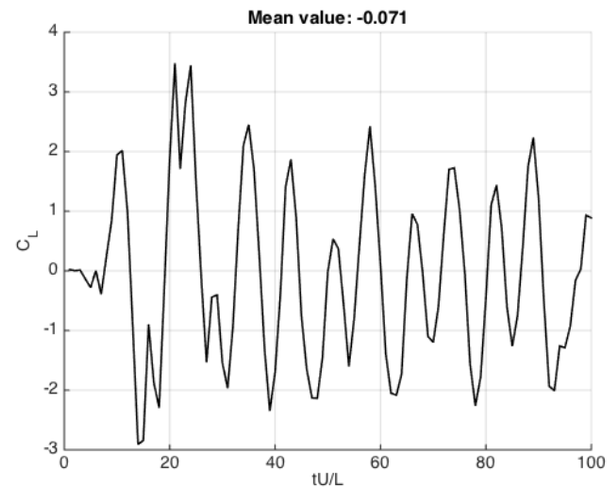
A LES run is performed using the Smagorinsky model and the unstructured tetra mesh with 20 planes.



LES Smagorinsky flow at several timesteps.



LES Smagorinsky C_D variation.



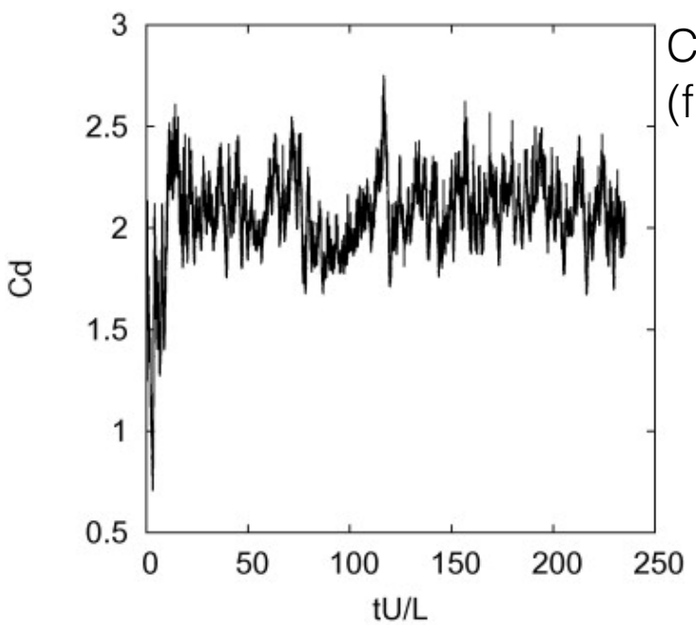
LES Smagorinsky C_L variation.

Code_Saturne tutorials

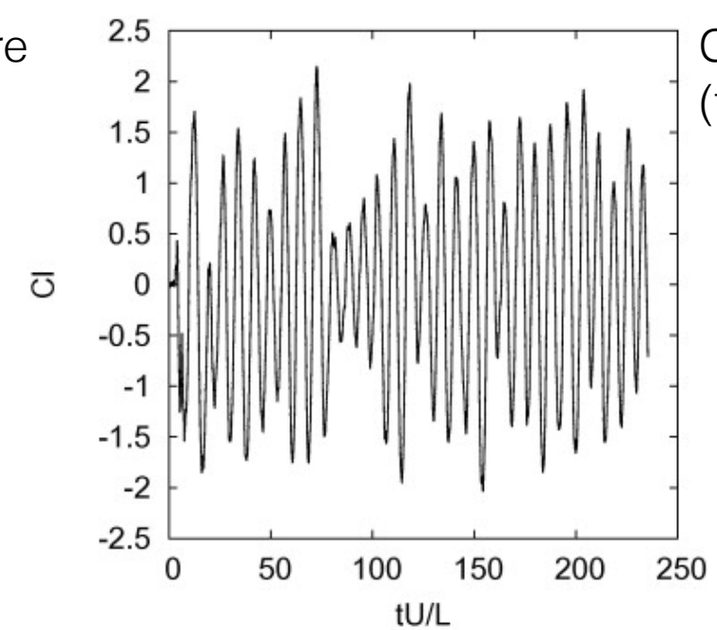
Square cylinder – Results discussion

Obtained results are compared with those on current literature [1] and are summarized in the following table:

	RANS Spalart-Allmaras	RANS k-omega SST	RANS k-omega SST hexa	LES Smagorisnky	Antepara et. al. (LES)	Sohankar (LES)	Verstappen (DNS)	Lyn-Luo (Exp)
C_D mean	2.011	2.183	2.051	2.322	2.024	2.03-2.32	2.1	2.1-2.2
C_L mean	0.011	-0.077	-0.441	-0.071	0.016	-	0.005	-



C_D variation in literature (from [1]).



C_L variation in literature (from [1]).

Code_Saturne tutorials

Square cylinder – References

[1] Antepara, O., et al. "Parallel adaptive mesh refinement for large-eddy simulations of turbulent flows." Computers & Fluids 110 (2015): 48-61.