

# Online Training – Advanced session

## February 2022

---

### Introduction to the FVM method: Standard practices in general CFD with applications to OpenFOAM

# Copyright and disclaimer

This offering is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPENFOAM® and OpenCFD® trademarks.

© 2014-2022 Wolf Dynamics.

All rights reserved. Unauthorized use, distribution or duplication is prohibited.

Contains proprietary and confidential information of Wolf Dynamics.

Wolf Dynamics makes no warranty, express or implied, about the completeness, accuracy, reliability, suitability, or usefulness of the information disclosed in this training material. This training material is intended to provide general information only. Any reliance the final user place on this training material is therefore strictly at his/her own risk. Under no circumstances and under no legal theory shall Wolf Dynamics be liable for any loss, damage or injury, arising directly or indirectly from the use or misuse of the information contained in this training material.

All trademarks are property of their owners.

Revision 1-2022

**JG**






# Before we begin

## On the training material

- **This training is based on OpenFOAM 9.**
- In the USB key/downloaded files you will find all the training material (tutorials, slides, and lectures notes).
- You can extract the training material wherever you want. From now on, this directory will become:
  - **\$TM**  
(abbreviation of **T**rainin**M**aterial)
- To uncompress the tutorials go to the directory where you copied the training material (**\$TM**) and then type in the terminal,
  - `$> tar -zxvf file_name.tar.gz`
- In the case directory of every single tutorial, you will find a few scripts with the extension `.sh`, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on.
- These scripts can be used to run the case automatically by typing in the terminal, for example,
  - `$> sh run_all.sh`
- These scripts are human-readable, and we highly recommend you open them, get familiar with the steps, and type the commands in the terminal. In this way, you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, run the cases automatically using these scripts.
- In the case directory, you will also find the `README.FIRST` file. In this file, you will find some additional comments.

# Conventions used

## The following typographical conventions are used in this training material

- Text in `Courier new` font indicates Linux commands that should be typed literally by the user in the terminal.
- Text in **`Courier new bold`** font indicates directories.
- Text in *`Courier new italic`* font indicates human readable files or ascii files.
- Text in **Arial bold font** indicates program elements such as variables, function names, classes, statements and so on. It also indicates environment variables, and keywords. They also highlight important information.
- Text in [Arial underline in blue](#) font indicates URLs and email addresses.
- This icon  indicates a warning or a caution.
- This icon  indicates a tip, suggestion, or a general note.
- This icon  indicates a folder or directory.
- This icon  indicates a human readable file (ascii file).
- This icon  indicates that the figure is an animation (animated gif).
- These characters `$>` indicate that a Linux command should be typed literally by the user in the terminal.



# Conventions used

The following typographical conventions are used in this training material

- Large code listing, ascii files listing, and screen outputs can be written in a square box, as follows:

```
1  #include <iostream>
2  using namespace std;
3
4  // main() is where program execution begins. It is the main function.
5  // Every program in c++ must have this main function declared
6
7  int main ()
8  {
9      cout << "Hello world";           //prints Hello world
10     return 0;                         //returns nothing
11 }
```

- To improve readability, the text might be colored.
- The font can be `Courier new` or **Arial bold**.
- And when required, the line number will be shown.

# Roadmap

1. **CFD and Multiphysics simulations**
2. **Important concepts to remember**
3. **The Finite Volume Method: An overview**
4. **Navier-Stokes equations and pressure-velocity coupling**
5. **On the CFL number**
6. **Unsteady and steady simulations**
7. **Understanding the residuals**
8. **Boundary conditions and initial conditions**
9. **The FVM in OpenFOAM: some implementation details and computational pointers**
10. **Best standard practices – General guidelines**
11. **Final remarks**

# Roadmap

1. **CFD and Multiphysics simulations**
2. Important concepts to remember
3. The Finite Volume Method: An overview
4. Navier-Stokes equations and pressure-velocity coupling
5. On the CFL number
6. Unsteady and steady simulations
7. Understanding the residuals
8. Boundary conditions and initial conditions
9. The FVM in OpenFOAM: some implementation details and computational pointers
10. Best standard practices – General guidelines
11. Final remarks

# CFD and Multiphysics simulations

## What is CFD?

- Computational Fluid Dynamics (CFD), is the science of predicting fluid flow, heat and mass transfer, chemical reactions, and related phenomena by using numerical methods and computers.
- To predict these phenomena, CFD finds the approximate numerical solution of the governing equations (conservation of mass, momentum, energy, and additional transport equations and models).
- CFD is an ensemble of,
  - Numerical methods.
  - Computer science.
  - Fluid dynamics.
  - Scientific visualization.
  - Engineering applications.
  - And most recently machine learning is making its way.

# CFD and Multiphysics simulations

## Multiphysics simulations

- Multiphysics simulations (MS) are computer simulations that involve physical models or phenomena that can be coupled together.
- MS consists in finding the approximate numerical solution of the governing equations (often PDEs).
- The physics involved can be fluid flow, heat transfer, mass transfer, stress/deformation, structural dynamics, chemical kinetics, pharmacokinetics, biochemistry, electrostatics, electromagnetics, fire dynamics, aero-acoustics, combustion, chemical reactions, finance, astronomy, and others, coupled in any combination.
- These disciplines can be solved in multiple dimensions (from 1D to 3D in steady or unsteady formulation), ranging from the continuum level to the molecular level.
- I like to see CFD as a subset of Multiphysics simulations.
- Multiphysics simulations can include the following computational disciplines:
  - Computational fluid dynamics → CFD
  - Computational structural dynamics → CSD
  - Computational heat transfer → CHT
  - Computational electromagnetics → CEM
  - Computational aero-acoustics → CAA
  - Magneto hydrodynamics → MHD
  - Fluid structure interaction → FSI
  - Discrete particle methods → DPM
  - And many more ...

# CFD and Multiphysics simulations

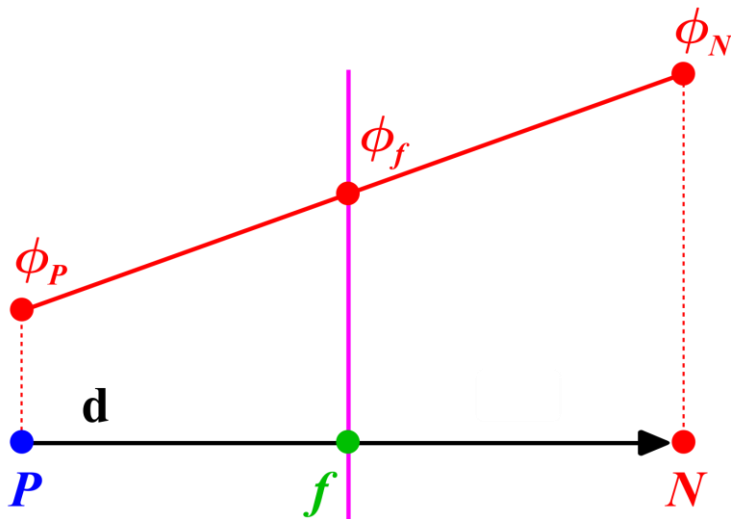
- In CFD/Multiphysics simulations there are many discretization approaches, just to name a few:
  - Finite Difference Method → FDM
  - Finite Element Method – Galerkin → G-FEM
  - Finite Element Method – Discontinuous Galerkin → DG-FEM
  - Finite Volume Method → FVM
  - Immersed Boundary Method → IBM
  - Lattice Boltzmann Method → LBM
  - Spectral Element Methods → SEM
  - Boundary Element Method → BEM
- Each method will find the approximate numerical solution of the governing equations
- The main difference among all methods is the way how they arrive to the system of discrete algebraic equations.
- Hereafter, we are going to address the FVM method.
- Most of the commercial Multiphysics frameworks and CFD solvers are based on the FVM.
- Also, many open-source frameworks are based on the FVM.
- The popularity of the FVM relies on the fact that can be used with arbitrary control volumes, it is easy to implement, and it enforces conservation in every single cell of the mesh (thus in the whole domain).
- OpenFOAM, SU2, code Saturn, CFX, FLUENT, Star-CCM, NUMECA, and CFD-ACE+ are all based on the FVM.

# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- 2. Important concepts to remember**
- ~~3. The Finite Volume Method: An overview~~
- ~~4. Navier-Stokes equations and pressure-velocity coupling~~
- ~~5. On the CFL number~~
- ~~6. Unsteady and steady simulations~~
- ~~7. Understanding the residuals~~
- ~~8. Boundary conditions and initial conditions~~
- ~~9. The FVM in OpenFOAM: some implementation details and computational pointers~~
- ~~10. Best standard practices – General guidelines~~
- ~~11. Final remarks~~

# Important concepts to remember

- Before starting this discussion, remember the following concepts as they will answer many questions later.
- Let us recall linear interpolation.
- In reference to the figure below, to find the value of the quantity  $\phi$  in  $f$ , using the known values of  $\phi$  in  $P$  and  $N$ , we can proceed as follows,



$$\phi_f = f_x \phi_P + (1 - f_x) \phi_N$$

$$f_x = \frac{fN}{PN} = \frac{|\mathbf{x}_f - \mathbf{x}_N|}{|\mathbf{d}|}$$



# Important concepts to remember

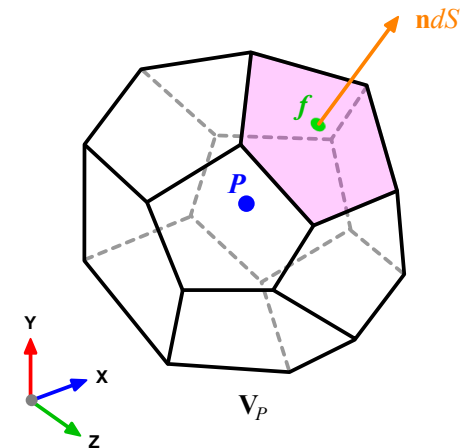
- Before starting this discussion, remember the following concepts as they will answer many questions later.
- Let us recall the Gauss theorem (also known as Divergence theorem or Ostrogradsky theorem),



$$\int_V \nabla \cdot \mathbf{a} dV = \oint_{\partial V} d\mathbf{S} \cdot \mathbf{a}$$

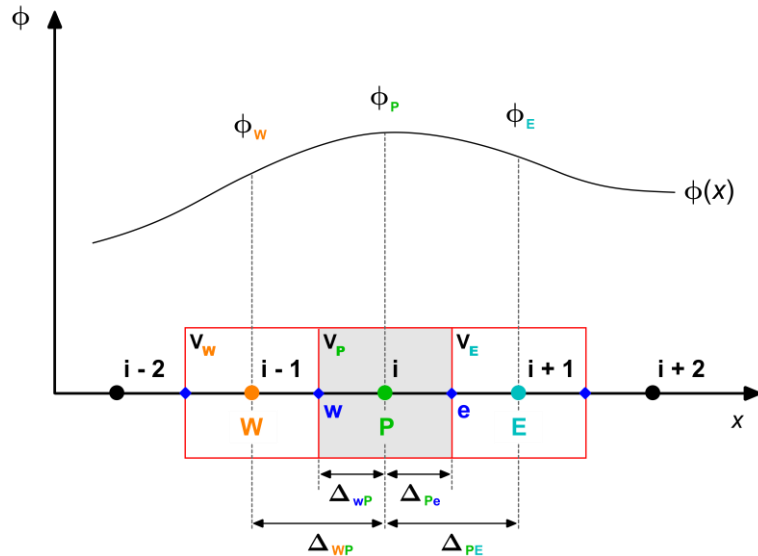
where  $\partial V_P$  is a closed surface bounding the control volume  $V_P$  and  $d\mathbf{S}$  represents an infinitesimal surface element with associated normal  $\mathbf{n}$  pointing outwards of the surface  $\partial V_P$ , and  $\mathbf{n}dS = d\mathbf{S}$

- The Gauss or Divergence theorem simply states that the outward flux of a vector field through a closed surface is equal to the volume integral of the divergence over the region inside the surface.



# Important concepts to remember

- Before starting this discussion, remember the following concepts as they will answer many questions later.
- Let us recall Taylor series expansions (TSE), they are used to define our profile assumptions, to reconstruct cell centered variables to face center variables, to compute derivatives, to determine truncation errors and so on.



- According to TSE, any continuous differentiable function can be expressed as an infinite sum of terms that are calculated from the values of the function derivatives at a single point.

$$\phi(x+\Delta x) = \phi(x) + \Delta x \left( \frac{\partial \phi}{\partial x} \right)_x + \frac{(\Delta x)^2}{2!} \left( \frac{\partial^2 \phi}{\partial x^2} \right)_x + \frac{(\Delta x)^3}{3!} \left( \frac{\partial^3 \phi}{\partial x^3} \right)_x + \mathcal{HOT},$$

- For example, using TSE the node center **E** in the figure can be approximated as,

$$\phi_E = \phi_P + \Delta_{PE} \left( \frac{\partial \phi}{\partial x} \right)_P + \mathcal{HOT}.$$

- And the face center **e** can be approximated as,

$$\phi_e = \phi_P + \Delta_{Pe} \left( \frac{\partial \phi}{\partial x} \right)_P + \mathcal{HOT}.$$

# Important concepts to remember

- During this discussion, we will use the general transport equation to explain the fundamentals of the finite volume method.

$$\underbrace{\int_{V_P} \frac{\partial \rho \phi}{\partial t} dV}_{\text{Temporal derivative}} + \underbrace{\int_{V_P} \nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{Convective term}} - \underbrace{\int_{V_P} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{Diffusion term}} = \underbrace{\int_{V_P} S_\phi(\phi) dV}_{\text{Source term}}$$

- But have in mind that starting from the general transport equation we can write down the Navier-Stokes equations (NSE). For example, by setting the variables to,

$$\begin{aligned}\phi &= 1 \\ \Gamma_\phi &= 0 \\ S_\phi &= 0\end{aligned}$$

- We can obtain the continuity equation,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

# Important concepts to remember

- During this discussion, we will use the general transport equation to explain the fundamentals of the finite volume method.

$$\underbrace{\int_{V_P} \frac{\partial \rho \phi}{\partial t} dV}_{\text{Temporal derivative}} + \underbrace{\int_{V_P} \nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{Convective term}} - \underbrace{\int_{V_P} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{Diffusion term}} = \underbrace{\int_{V_P} S_\phi(\phi) dV}_{\text{Source term}}$$

- But have in mind that starting from the general transport equation we can write down the Navier-Stokes equations (NSE). For example, by setting the variables to,

$\phi = u$	$\phi = v$	$\phi = w$
$\Gamma_\phi = \mu$	$\Gamma_\phi = \mu$	$\Gamma_\phi = \mu$
$S_\phi = S_u - \frac{\partial p}{\partial x}$	$S_\phi = S_v - \frac{\partial p}{\partial y}$	$S_\phi = S_w - \frac{\partial p}{\partial z}$

- We can obtain the momentum equations,

$$\frac{\partial \rho u}{\partial t} + \nabla \cdot (\rho \mathbf{u} u) = \nabla \cdot (\mu \nabla u) - \frac{\partial p}{\partial x} + S_u \qquad \frac{\partial \rho v}{\partial t} + \nabla \cdot (\rho \mathbf{u} v) = \nabla \cdot (\mu \nabla v) - \frac{\partial p}{\partial y} + S_v \qquad \frac{\partial \rho w}{\partial t} + \nabla \cdot (\rho \mathbf{u} w) = \nabla \cdot (\mu \nabla w) - \frac{\partial p}{\partial z} + S_w$$

# Important concepts to remember

- During this discussion, we will use the general transport equation to explain the fundamentals of the finite volume method.

$$\underbrace{\int_{V_P} \frac{\partial \rho \phi}{\partial t} dV}_{\text{Temporal derivative}} + \underbrace{\int_{V_P} \nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{Convective term}} - \underbrace{\int_{V_P} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{Diffusion term}} = \underbrace{\int_{V_P} S_\phi(\phi) dV}_{\text{Source term}}$$

- But have in mind that starting from the general transport equation we can write down the Navier-Stokes equations (NSE). For example, by setting the variables to,

$$\begin{aligned}\phi &= h \\ \Gamma_\phi &= k/C_p \\ S_\phi &= S_h\end{aligned}$$

- We can obtain the energy equation,

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot (\rho \mathbf{u} h) = \nabla \cdot \left( \frac{k}{C_p} \nabla T \right) + S_h$$

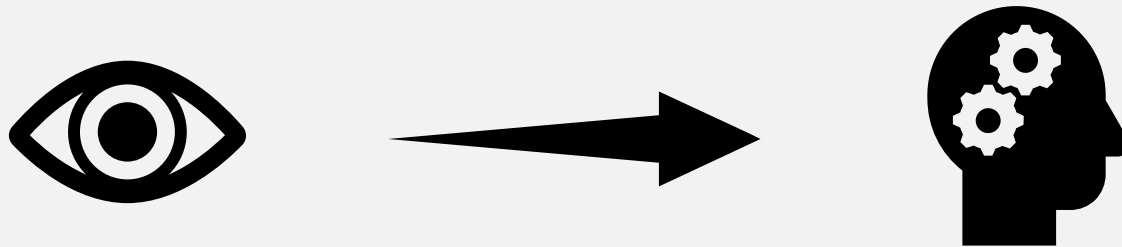
# Important concepts to remember

- Contrary to commercial CFD solvers, in OpenFOAM there are no default values.
- It is up to the user to find those values.
- However, following good standard practices and knowing a little bit the theory is a very good starting point.
- Our goal is to give you the best standard practices and default values (ours) to be used with OpenFOAM.

# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- ~~2. Important concepts to remember~~
- 3. The Finite Volume Method: An overview**
- ~~4. Navier-Stokes equations and pressure-velocity coupling~~
- ~~5. On the CFL number~~
- ~~6. Unsteady and steady simulations~~
- ~~7. Understanding the residuals~~
- ~~8. Boundary conditions and initial conditions~~
- ~~9. The FVM in OpenFOAM: some implementation details and computational pointers~~
- ~~10. Best standard practices – General guidelines~~
- ~~11. Final remarks~~

- In this training, we will focus our eyes to train our brain.





# The Finite Volume Method: An overview

**General transport equation and profile assumptions**

# The Finite Volume Method: An overview

- Let us use the general transport equation as the starting point to explain the FVM,

$$\underbrace{\int_{V_P} \frac{\partial \rho \phi}{\partial t} dV}_{\text{Temporal derivative}} + \underbrace{\int_{V_P} \nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{Convective term}} - \underbrace{\int_{V_P} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{Diffusion term}} = \underbrace{\int_{V_P} S_\phi(\phi) dV}_{\text{Source term}}$$

- We want to solve the general transport equation for the transported quantity  $\phi$  in a given domain, with given boundary conditions BC and initial conditions IC.
- This is a second order equation. For good accuracy, it is necessary that the order of the discretization is equal or higher than the order of the equation that is being discretized.
- By the way, starting from this equation we can write down the Navier-Stokes equations (NSE). So everything we are going to address also applies to the NSE.

# The Finite Volume Method: An overview

- Let us use the general transport equation as the starting point to explain the FVM,

$$\underbrace{\int_{V_P} \frac{\partial \rho \phi}{\partial t} dV}_{\text{Temporal derivative}} + \underbrace{\int_{V_P} \nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{Convective term}} - \underbrace{\int_{V_P} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{Diffusion term}} = \underbrace{\int_{V_P} S_\phi(\phi) dV}_{\text{Source term}}$$

- Hereafter we are going to assume that the discretization practice is at least second order accurate in space and time.
- As consequence of the previous requirement, all dependent variables are assumed to vary linearly around a point  $P$  in space and instant  $t$  in time,

$$\phi(\mathbf{x}) = \phi_P + (\mathbf{x} - \mathbf{x}_P) \cdot (\nabla \phi)_P \quad \text{where} \quad \phi_P = \phi(\mathbf{x}_P)$$

$$\phi(t + \delta t) = \phi^t + \delta t \left( \frac{\partial \phi}{\partial t} \right)^t \quad \text{where} \quad \phi^t = \phi(t)$$

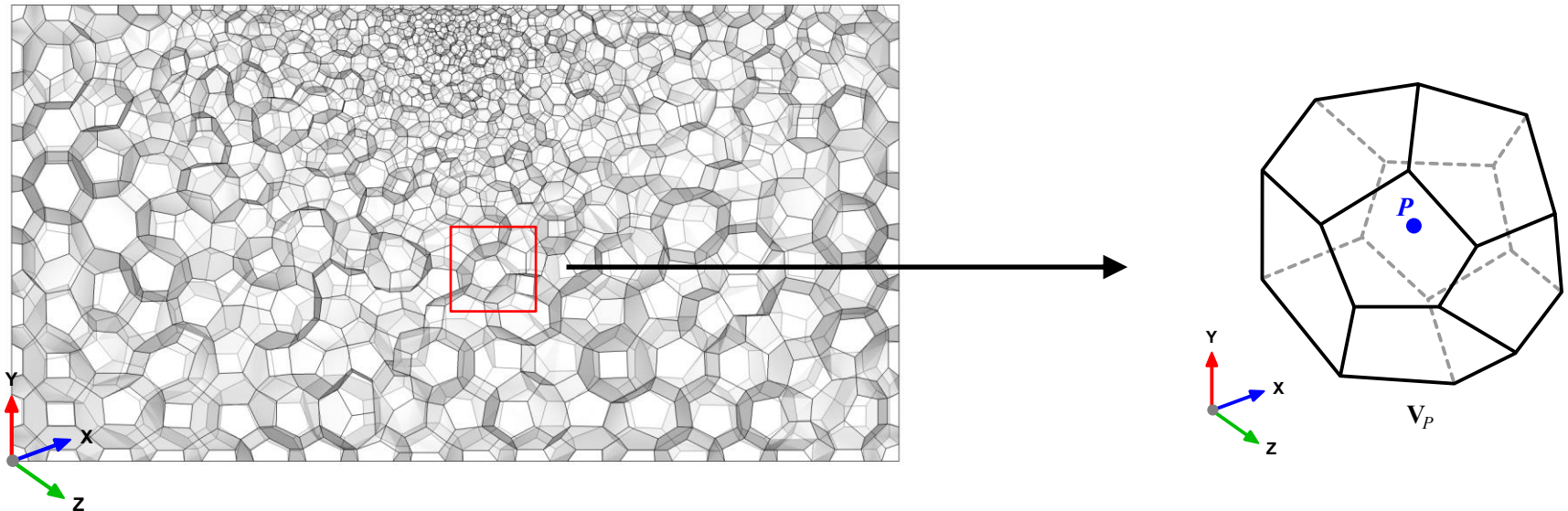
Profile assumptions using Taylor expansions around point  $P$  (in space) and point  $t$  (in time)

# The Finite Volume Method: An overview

**Mesh data, geometrical information,  
and variable arrangement**

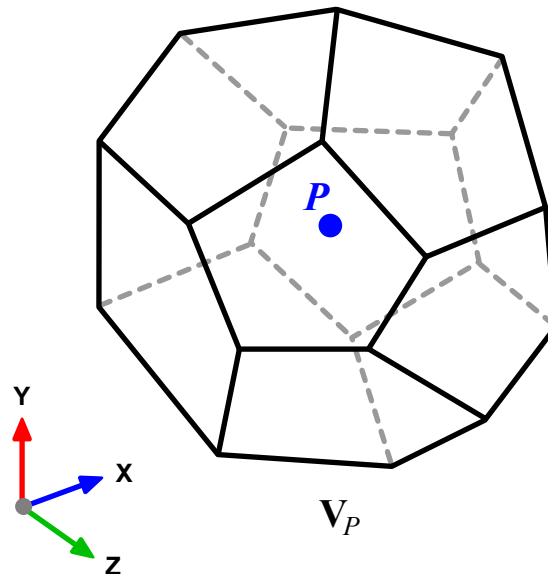
# The Finite Volume Method: An overview

- Let us divide the solution domain into a finite number of arbitrary control volumes or cells, such as the one illustrated below.
- Inside each control volume the solution is sought.
- We know all the geometrical information of all cells. That is, cell centers, face centers, cells neighbors, face connectivity, cells volume, faces area, vectors connecting cells centers, and so on.
- Let us see in detail all the required geometrical information.



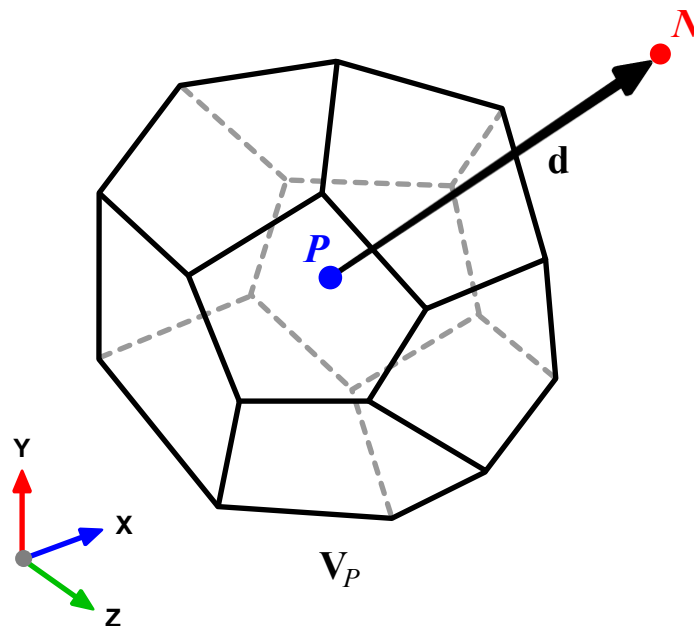
# The Finite Volume Method: An overview

- The control volume  $V_P$  has a volume  $V$  and is constructed around point  $P$ , which is the centroid of the control volume. Therefore, the notation  $V_P$ .
- The volume  $V$  of all control volumes is known.
- The control volumes can be of any shape (e.g., tetrahedrons, hexes, prisms, pyramids, dodecahedrons, and so on).
- The only requirement is that the elements need to be convex and the faces that made up the control volume need to be planar.



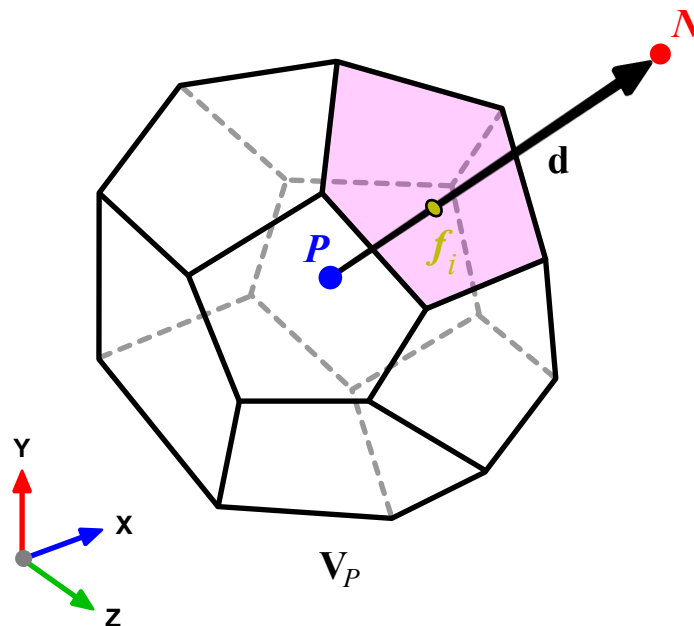
# The Finite Volume Method: An overview

- We also know the neighbors of the control volume or  $V_N$  (following our notation).
- A face of  $V_P$  can have more than one neighbor (non-conformal mesh)
- At this point, we know all the connectivity information, that is:  $P$  location, neighbors  $N$ 's of  $P$ , faces connectivity, vertices location, and so on.
- Note that the volume of the control volumes needs to be higher than zero.



# The Finite Volume Method: An overview

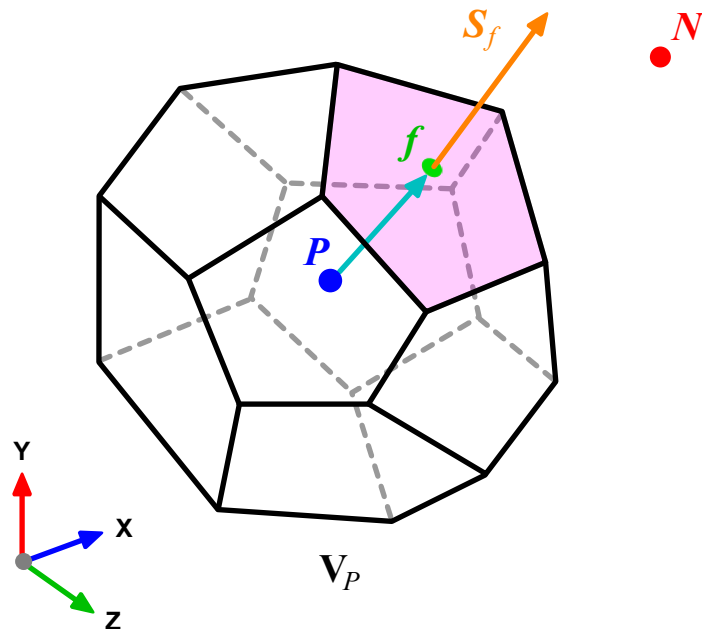
- The vector from the centroid  $P$  of  $V_P$  to the centroid  $N$  of  $V_N$  is named  $\mathbf{d}$ .
- The location where the vector  $\mathbf{d}$  intersects a face is  $f_i$ .
- We know this information for all control volumes and all faces.
- We also know which control volumes are internal and which control volumes lie on the boundaries.





# The Finite Volume Method: An overview

- The control volume faces are labeled  $f$ , which also denotes the face center.
- The face area vector  $S_f$  point outwards from the control volume, is located at the face centroid, is normal to the face, and has a magnitude equal to the area of the face.
- The vector from the centroid  $P$  to the face center  $f$  is named  $Pf$ .
- Note that the vectors  $S_f$  and  $Pf$  not necessarily are aligned.
- Same applies with the vector  $d$  (vector connecting  $P$  and  $N$ ).



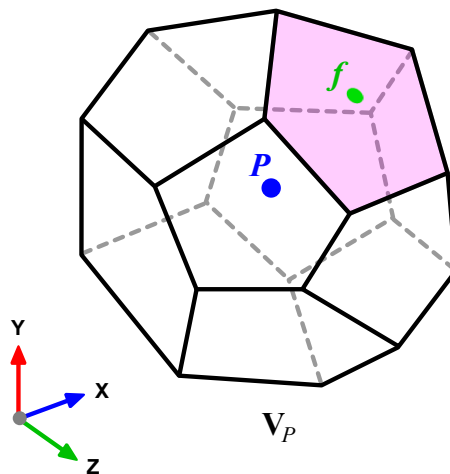
# The Finite Volume Method: An overview

- In the control volume illustrated, the centroid  $P$  is given by

$$\int_{V_P} (\mathbf{x} - \mathbf{x}_P) dV = 0$$

- In the same way, the centroid of face  $f$  is given by

$$\int_{S_f} (\mathbf{x} - \mathbf{x}_P) dS = 0$$



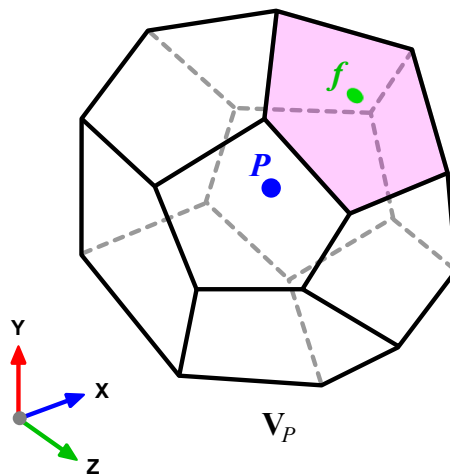
Second order approximations

# The Finite Volume Method: An overview

- Finally, we assume that the values of all variables are computed and stored in the centroid of the control volume  $V_P$  and that they are represented by a piecewise constant profile (the mean value),

$$\phi_P = \bar{\phi} = \frac{1}{V_P} \int_{V_P} \phi(\mathbf{x}) dV$$

- This is known as the collocated arrangement. Specifically, cell centered collocated arrangement.
- This is what is called in literature variable arrangement and mean value assumptions.



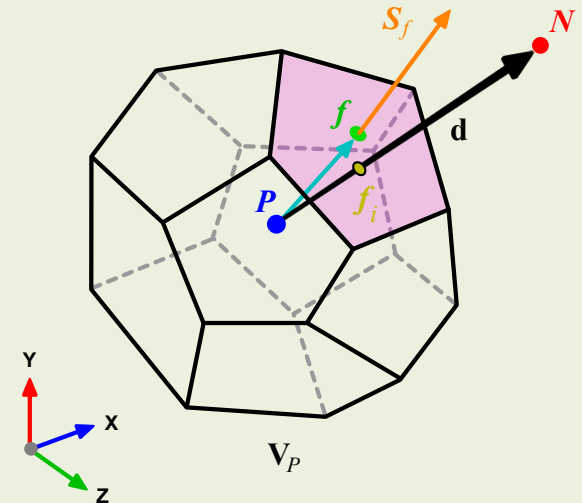
Second order approximations

# The Finite Volume Method: An overview

- Putting all together, it is a lot geometrical information that we need to track.
- A lot of overhead goes into the data book-keeping.
- At the end of the day, the FVM simply consist in conservation of the transported quantities and interpolating information from cell centers to face centers.

## Summary:

- The control volume  $V_P$  has a volume  $V$  and is constructed around point  $P$ , which is the centroid of the control volume. Therefore, the notation  $V_P$ .
- The vector from the centroid  $P$  of  $V_P$  to the centroid  $N$  of  $V_N$  is named  $\mathbf{d}$ .
- We also know all neighbors  $V_N$  of the control volume  $V_P$
- The control volume faces are labeled  $f$ , which also denotes the face center.
- The location where the vector  $\mathbf{d}$  intersects a face is  $f_i$ .
- The face area vector  $\mathbf{S}_f$  point outwards from the control volume, is located at the face centroid, is normal to the face and has a magnitude equal to the area of the face.
- The vector from the centroid  $P$  to the face center  $f$  is named  $Pf$ .



# The Finite Volume Method: An overview

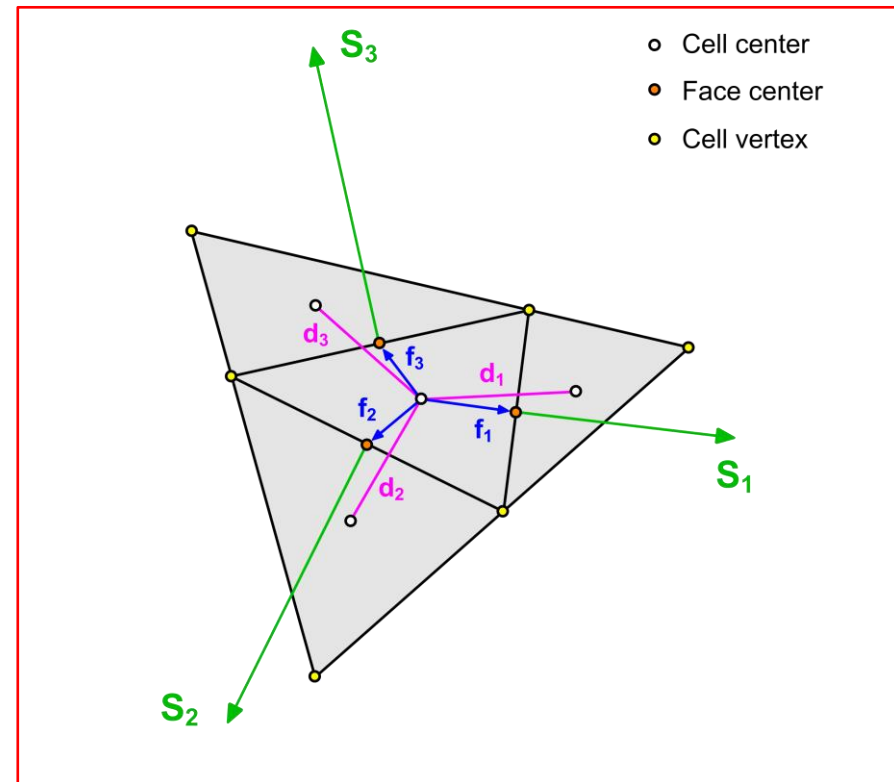
- Have in mind that there are different FVM formulations based on the variable arrangement (e.g., cell centered, node/vertex based).
- Hereafter we will address the cell centered collocated arrangement, which is the one implemented in OpenFOAM and many commercial CFD software (e.g., Ansys Fluent and StarCCM+).
- Remember, for good accuracy we want a method that is at least second order accurate (as the equations we are solving are second order).
- All the previous approximations are at least second order accurate.
- So far, we have talked about geometric requirements of the FVM.
- Let us address interpolation from cell center to face center and computation of the face fluxes.
- But before moving on, let us mention something about one of the elephants in the room, mesh quality.

# The Finite Volume Method: An overview

- In CFD, the mesh is everything.
- As we will see later, the matrix coefficients of the discretized system of algebraic equations depends on the geometry quantities shown in the figure.
- Specifically, on the dot product of  $\mathbf{S}$  (vector normal to face passing by the face center) and  $\mathbf{d}$  (vector connecting two cell centers).
- This dependence on the dot product  $\mathbf{S} \cdot \mathbf{d}$  is due to the fact that the coefficients contain the following term,

$$\frac{\mathbf{S} \cdot \mathbf{S}}{\mathbf{S} \cdot \mathbf{d}}$$

- For perfect cells (orthogonal meshes), the dot product is equal to one (there is no deviation between the vectors  $\mathbf{S}$  and  $\mathbf{d}$ ).
- The more a cell deviates from its perfect shape, the smaller the dot product becomes, and this results in large values of the matrix coefficients which increases the system stiffness.
- For very bad quality cells (e.g., very skew cells or cells with zero volume), this vector product may become zero, producing an undefined system (throwing a division by zero error).



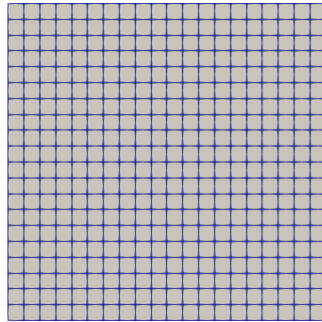
In the figure:

- $\mathbf{S}$  is the vector normal to face and anchored at the face center
- $\mathbf{d}$  is the vector connecting two cell centers.
- $\mathbf{f}$  is the vector from the cell center to the face center.
- If all these vectors are aligned, we are in the presence of a perfect mesh. In practice, this does not happen very often.

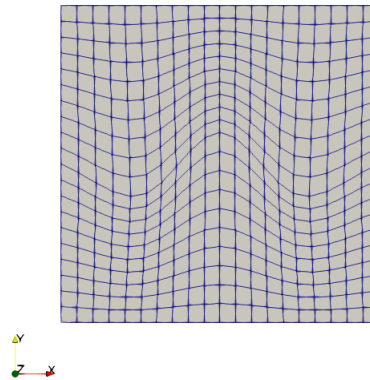
# The Finite Volume Method: An overview

- Different meshes and their respective matrix of coefficients.
- The quality of all meshes is excellent; however, the matrix of coefficients is different in all cases.

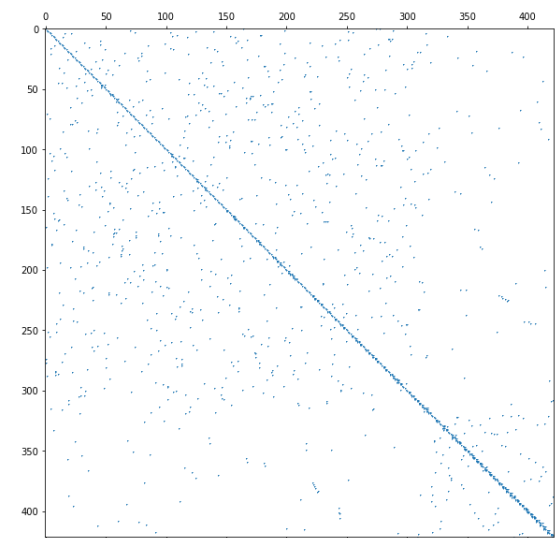
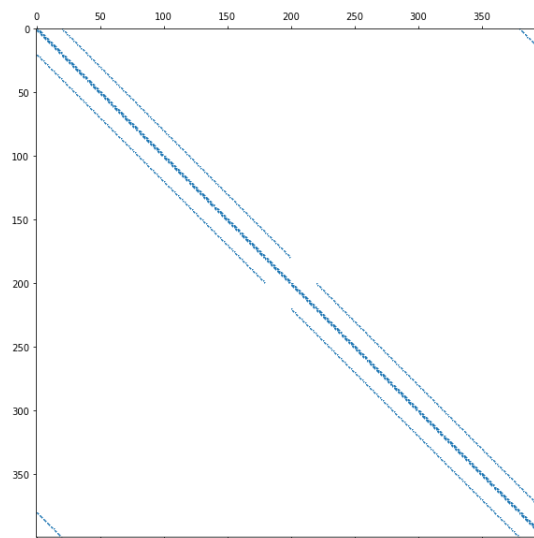
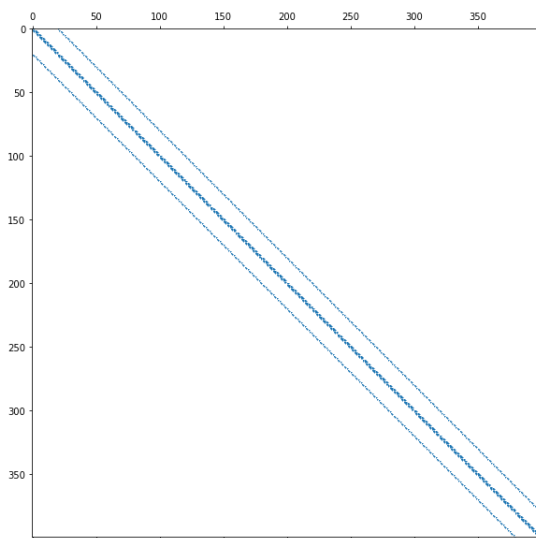
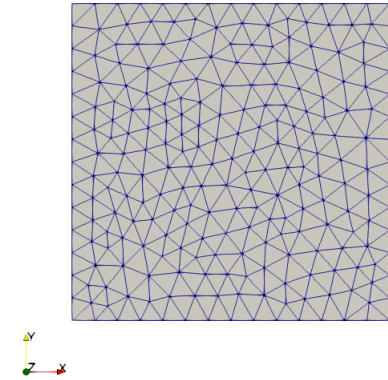
Orthogonal mesh (perfect mesh)



Non-orthogonal mesh



Unstructured triangular mesh



# The Finite Volume Method: An overview

**Gauss theorem and face fluxes computation**



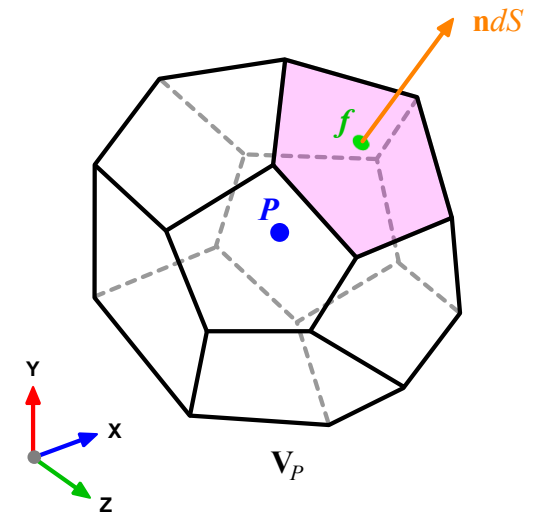
# The Finite Volume Method: An overview

- Let us recall the Gauss or Divergence theorem,

$$\int_V \nabla \cdot \mathbf{a} dV = \oint_{\partial V} d\mathbf{S} \cdot \mathbf{a}$$

where  $\partial V_P$  is a closed surface bounding the control volume  $V_P$  and  $dS$  represents an infinitesimal surface element with associated normal  $\mathbf{n}$  pointing outwards of the surface  $\partial V_P$ , and  $\mathbf{n}dS = d\mathbf{S}$

- The Gauss or Divergence theorem simply states that the outward flux of a vector field through a closed surface is equal to the volume integral of the divergence over the region inside the surface.
- This theorem is fundamental in the FVM, it is used to convert the volume integrals appearing in the governing equations into surface integrals.



# The Finite Volume Method: An overview

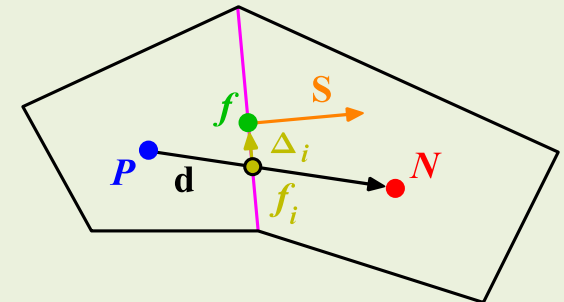
- Let us use the Gauss theorem to convert the volume integrals into surface integrals,

$$\underbrace{\int_{V_P} \frac{\partial \rho \phi}{\partial t} dV}_{\text{Temporal derivative}} + \underbrace{\int_{V_P} \nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{Convective term}} - \underbrace{\int_{V_P} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{Diffusion term}} = \underbrace{\int_{V_P} S_\phi(\phi) dV}_{\text{Source term}}$$

$$\int_V \nabla \cdot \mathbf{a} dV = \oint_{\partial V} d\mathbf{S} \cdot \mathbf{a}$$

$$\frac{\partial}{\partial t} \int_{V_P} (\rho \phi) dV + \oint_{\partial V_P} \underbrace{d\mathbf{S} \cdot (\rho \mathbf{u} \phi)}_{\text{convective flux}} - \oint_{\partial V_P} \underbrace{d\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi)}_{\text{diffusive flux}} = \int_{V_P} S_\phi(\phi) dV$$

- At this point the problem reduces to interpolating somehow the cell centered values (known quantities) to the face centers.
- Any deviation when interpolating the cell centered values to the face centers ( $D_i$ ) is a source of error.



# The Finite Volume Method: An overview

**Convective, diffusive, gradients and source terms approximations**

# The Finite Volume Method: An overview

- Integrating in space each term of the general transport equation and by using Gauss theorem, yields to the following discrete equations for each term

## Convective term:

$$\int_{V_P} \underbrace{\nabla \cdot (\rho \mathbf{u} \phi)}_{\text{convective term}} dV = \oint_{\partial V_P} \underbrace{d\mathbf{S} \cdot (\rho \mathbf{u} \phi)}_{\text{convective flux}} = \sum_f \int_f d\mathbf{S} \cdot (\rho \mathbf{u} \phi)_f \approx \sum_f \underbrace{\mathbf{S}_f \cdot (\overline{\rho \mathbf{u} \phi})_f}_{\text{mid point rule}} = \sum_f \mathbf{S}_f \cdot (\rho \mathbf{u} \phi)_f$$

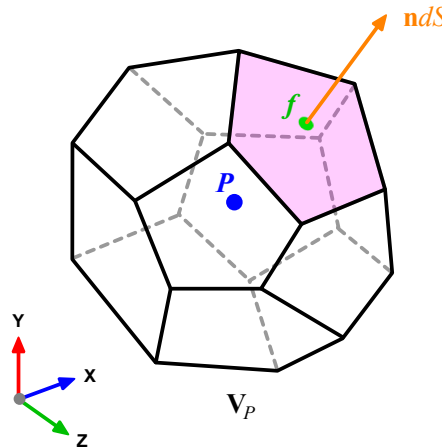
↓

By using Gauss theorem we convert volume integrals into surface integrals

where we have approximated the integrant by means of the mid point rule, which is second order accurate

## Gauss theorem:

$$\int_V \nabla \cdot \mathbf{a} dV = \oint_{\partial V} d\mathbf{S} \cdot \mathbf{a}$$



# The Finite Volume Method: An overview

- Integrating in space each term of the general transport equation and by using Gauss theorem, yields to the following discrete equations for each term

## Diffusive term:

$$\underbrace{\int_{V_P} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{diffusion term}} = \underbrace{\oint_{\partial V_P} d\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi)}_{\text{diffusive flux}} = \sum_f \int_f d\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi)_f \approx \underbrace{\sum_f \mathbf{S}_f \cdot (\overline{\rho \Gamma_\phi \nabla \phi})_f}_{\text{approximation}} = \sum_f \mathbf{S}_f \cdot (\rho \Gamma_\phi \nabla \phi)_f$$

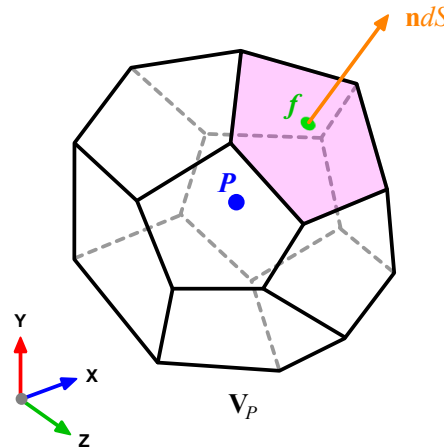
↓

By using Gauss theorem we convert volume integrals into surface integrals

where we have approximated the integrant by means of the mid point rule, which is second order accurate

## Gauss theorem:

$$\int_V \nabla \cdot \mathbf{a} dV = \oint_{\partial V} d\mathbf{S} \cdot \mathbf{a}$$



# The Finite Volume Method: An overview

- Integrating in space each term of the general transport equation and by using Gauss theorem, yields to the following discrete equations for each term

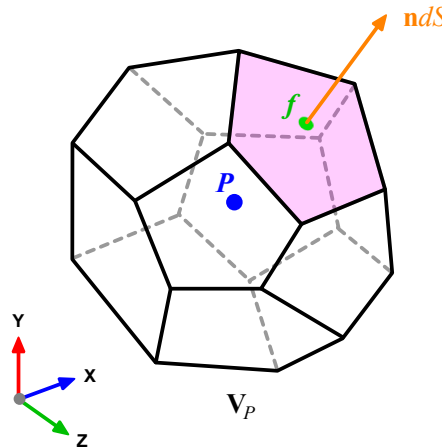
## Gradient term:

$$(\nabla \phi)_P = \frac{1}{V_P} \sum_f (\mathbf{S}_f \phi_f)$$

where we have approximated the centroid gradients by using the Gauss theorem.  
This method is second order accurate and is known as Gauss cell-based.

## Gauss theorem:

$$\int_V \nabla \cdot \mathbf{a} dV = \oint_{\partial V} d\mathbf{S} \cdot \mathbf{a}$$



## Note:

- There are more methods for gradients computation, e.g., least squares, node-based reconstruction, and so on.
- As there is some algebra involved, we do not provide the demonstration.

# The Finite Volume Method: An overview

- Integrating in space each term of the general transport equation and by using Gauss theorem, yields to the following discrete equations for each term

## Source term:

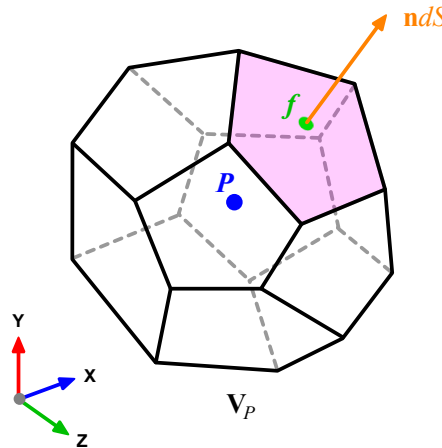
$$\int_{V_P} S_\phi(\phi) dV = S_c V_P + S_p V_P \phi_P$$

This approximation is exact if  $S_\phi$  is either constant or varies linearly within the control volume; otherwise is second order accurate.

$S_c$  is the constant part of the source term and  $S_p$  is the non-linear part

## Gauss theorem:

$$\int_V \nabla \cdot \mathbf{a} dV = \oint_{\partial V} d\mathbf{S} \cdot \mathbf{a}$$



# The Finite Volume Method: An overview

- Integrating in space each term of the general transport equation and by using Gauss theorem, yields to the following discrete equations for each term

## Convective term:

$$\int_{V_P} \underbrace{\nabla \cdot (\rho \mathbf{u} \phi)}_{\text{convective term}} dV = \oint_{\partial V_P} \underbrace{d\mathbf{S} \cdot (\rho \mathbf{u} \phi)}_{\text{convective flux}} = \sum_f \int_f d\mathbf{S} \cdot (\rho \mathbf{u} \phi)_f \approx \sum_f \mathbf{S}_f \cdot (\overline{\rho \mathbf{u} \phi})_f = \sum_f \mathbf{S}_f \cdot (\rho \mathbf{u} \phi)_f$$

## Diffusive term:

$$\int_{V_P} \underbrace{\nabla \cdot (\rho \Gamma_\phi \nabla \phi)}_{\text{diffusion term}} dV = \oint_{\partial V_P} \underbrace{d\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi)}_{\text{diffusive flux}} = \sum_f \int_f d\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi)_f \approx \sum_f \mathbf{S}_f \cdot (\overline{\rho \Gamma_\phi \nabla \phi})_f = \sum_f \mathbf{S}_f \cdot (\rho \Gamma_\phi \nabla \phi)_f$$

## Source term:

$$\int_{V_P} S_\phi(\phi) dV = S_c V_P + S_p V_P \phi_P$$

## Gradient term:

$$(\nabla \phi)_P = \frac{1}{V_P} \sum_f (\mathbf{S}_f \phi_f)$$



# The Finite Volume Method: An overview

- Using the previous equations to evaluate the general transport equation over all the control volumes, we obtain the following semi-discrete equation

$$\underbrace{\int_{V_P} \frac{\partial \rho \phi}{\partial t} dV}_{\text{temporal derivative}} + \sum_f \underbrace{\mathbf{S}_f \cdot (\rho \mathbf{u} \phi)_f}_{\text{convective flux}} - \sum_f \underbrace{\mathbf{S}_f \cdot (\rho \Gamma_\phi \nabla \phi)_f}_{\text{diffusive flux}} = \underbrace{(S_c V_P + S_p V_P \phi_P)}_{\text{source term}}$$

where  $\mathbf{S} \cdot (\rho \mathbf{u} \phi) = F^C$  is the convective flux

and  $\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi) = F^D$  is the diffusive flux.

- And recall that all variables are computed and stored at the centroid of the control volumes.
- The face values appearing in the convective and diffusive fluxes have to be computed by some form of interpolation from the centroid values of the control volumes at both sides of face  $f$ .

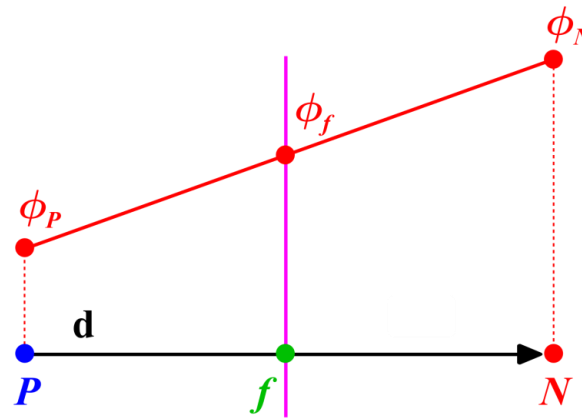
# The Finite Volume Method: An overview

**Interpolation of the convective fluxes**

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes

- By looking the figure below, the face values appearing in the convective flux can be computed as follows,



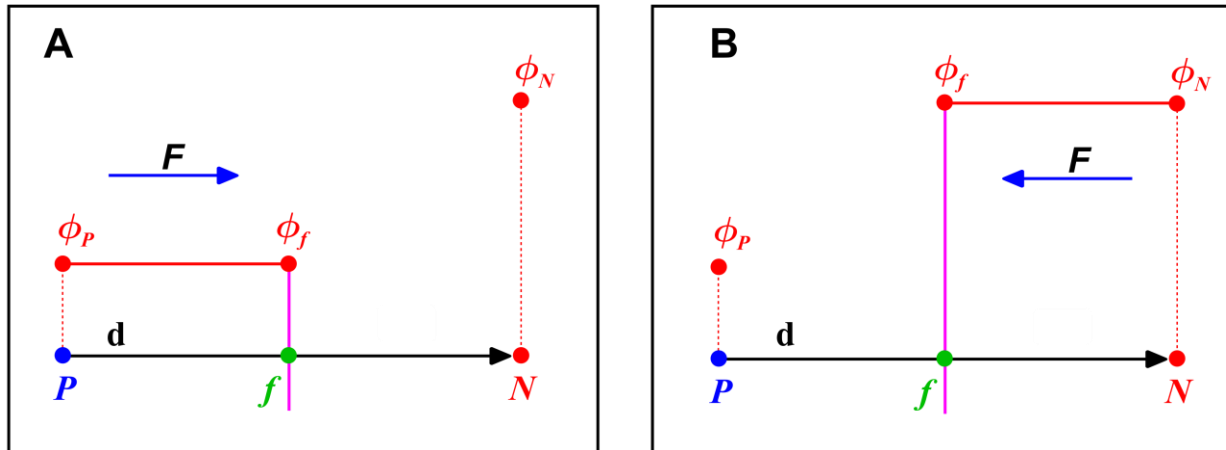
$$\phi_f = f_x \phi_P + (1 - f_x) \phi_N \qquad f_x = \frac{fN}{PN} = \frac{|\mathbf{x}_f - \mathbf{x}_N|}{|\mathbf{d}|}$$

- This type of interpolation scheme is known as linear interpolation or central differencing, and it is second order accurate.
- However, it may generate oscillatory solutions (unbounded solutions).

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes

- By looking the figure below, the face values appearing in the convective flux can be computed as follows,



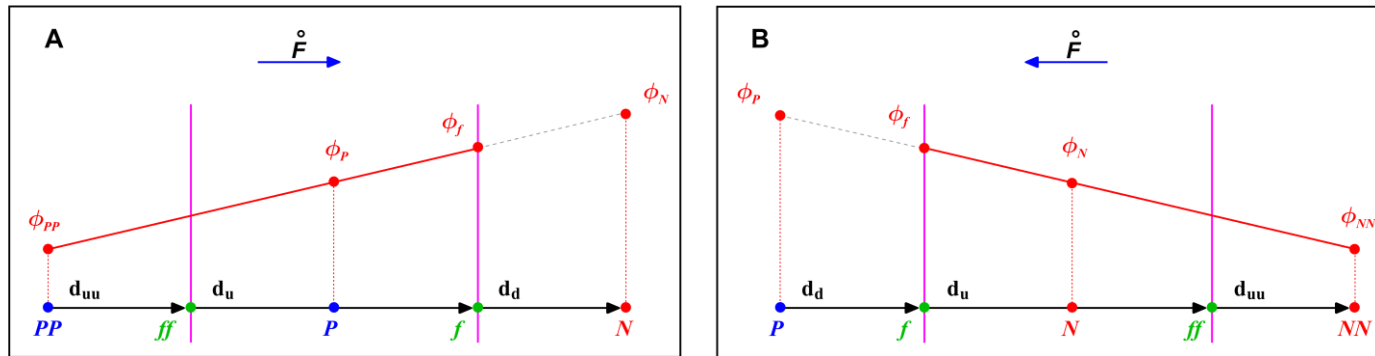
$$\phi_f = \begin{cases} \phi_f = \phi_P & \text{for } \dot{\bar{F}} \geq 0, \\ \phi_f = \phi_N & \text{for } \dot{\bar{F}} < 0. \end{cases}$$

- This type of interpolation scheme is known as upwind differencing, and it is first order accurate.
- This scheme is bounded (non-oscillatory) and diffusive.

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes

- By looking the figure below, the face values appearing in the convective flux can be computed as follows,



$$\phi_f = \begin{cases} \phi_P + \frac{1}{2}(\phi_P - \phi_{PP}) = \frac{3}{2}\phi_P - \frac{1}{2}\phi_{PP} & \text{for } \dot{F} \geq 0, \\ \phi_N + \frac{1}{2}(\phi_N - \phi_{NN}) = \frac{3}{2}\phi_N - \frac{1}{2}\phi_{NN} & \text{for } \dot{F} < 0. \end{cases}$$

- This type of interpolation scheme is known as second order upwind differencing (SOU), linear upwind differencing (LUD) or Beam-Warming (BW), and it is second order accurate.
- For highly convective flows or in the presence of strong gradients, this scheme is oscillatory (unbounded).

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes

- To prevent oscillations in the SOU, we add a limiter function  $\psi(r)$ , often referred to as flux or gradient (slope) limiter.

$$\phi_f = \begin{cases} \phi_P + \frac{1}{2}\psi_P^-(\phi_P - \phi_{PP}) & \text{for } \dot{F} \geq 0, \\ \phi_N - \frac{1}{2}\psi_P^+(\phi_{NN} - \phi_N) & \text{for } \dot{F} < 0. \end{cases}$$

- When the limiter detects strong gradients or changes in slope, it switches locally to low resolution (upwind).
- The concept of the limiter function  $\psi(r)$  is based on monitoring the ratio of successive gradients, e.g.,

$$r_P^- = \frac{\phi_N - \phi_P}{\phi_P - \phi_{PP}} \quad \text{and} \quad r_P^+ = \frac{\phi_P - \phi_N}{\phi_N - \phi_{NN}}$$

- By adding a well-designed limiter function  $\psi(r)$ , we get a high resolution (second order accurate), and bounded scheme. This is a TVD scheme.

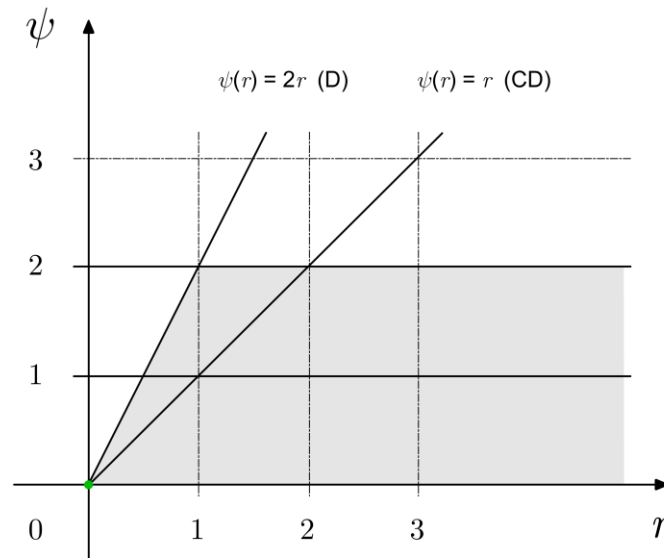
# The Finite Volume Method: An overview

## **TVD Schemes**

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – TVD schemes

- A TVD scheme, is a scheme that does not create new local undershoots and/or overshoots in the solution or amplify existing extremes.
- In CFD we want stable, non-oscillatory, bounded, high order (HO) schemes. We want high resolution schemes (HR), in other words, TVD schemes and at least second order accurate.
- The Sweby diagram (Sweby, 1984), gives the necessary and sufficient conditions for a scheme to be TVD. In the figure, the grey area represents the admissible TVD region. However, not all limiter functions are second order.



$$\phi_f = \begin{cases} \phi_P + \frac{1}{2}\psi_P^-(\phi_P - \phi_{PP}) & \text{for } \overset{\circ}{F} \geq 0, \\ \phi_N - \frac{1}{2}\psi_P^+(\phi_{NN} - \phi_N) & \text{for } \overset{\circ}{F} < 0. \end{cases}$$

$$\psi(r) = 2$$

$$\psi(r) = 1 \text{ (SOU)}$$

$$\psi(r) = 0 \text{ (UD)}$$

UD = upwind  
SOU = second order upwind  
CD = central differencing  
D = downwind

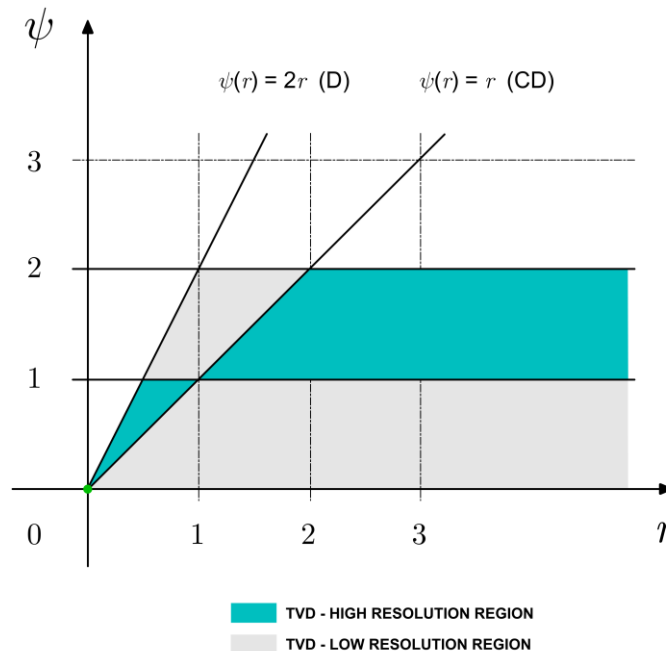
TVD REGION



# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – TVD schemes

- The choice of the limiter function  $\psi(r)$  dictates the order of the scheme and its boundedness.
- High-resolution schemes fall in the blue area and low-resolution schemes fall in the grey area.
- The development of high-resolution schemes is one of the most remarkable achievements of the history of CFD.

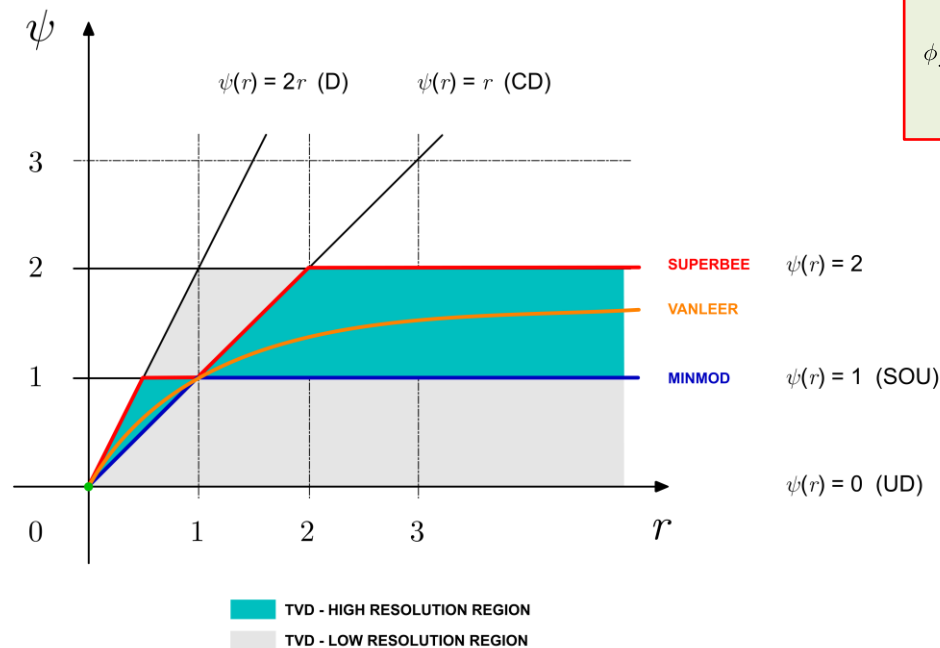


$$\phi_f = \begin{cases} \phi_P + \frac{1}{2}\psi_P^-(\phi_P - \phi_{PP}) & \text{for } \overset{\circ}{F} \geq 0, \\ \phi_N - \frac{1}{2}\psi_P^+(\phi_{NN} - \phi_N) & \text{for } \overset{\circ}{F} < 0. \end{cases}$$

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – TVD schemes

- The drawback of the limiters is that they reduce the accuracy of the scheme **locally** to first order (low-resolution scheme), when  $r \leq 0$  (sharp gradient, opposite slopes or zero gradient). However, this is justified when it serves to suppress oscillations.
- The various limiters have different switching characteristics and are selected according to the particular problem and solution scheme.
- No particular limiter has been found to work well for all problems, and a particular choice is usually made on a trial-and-error basis.



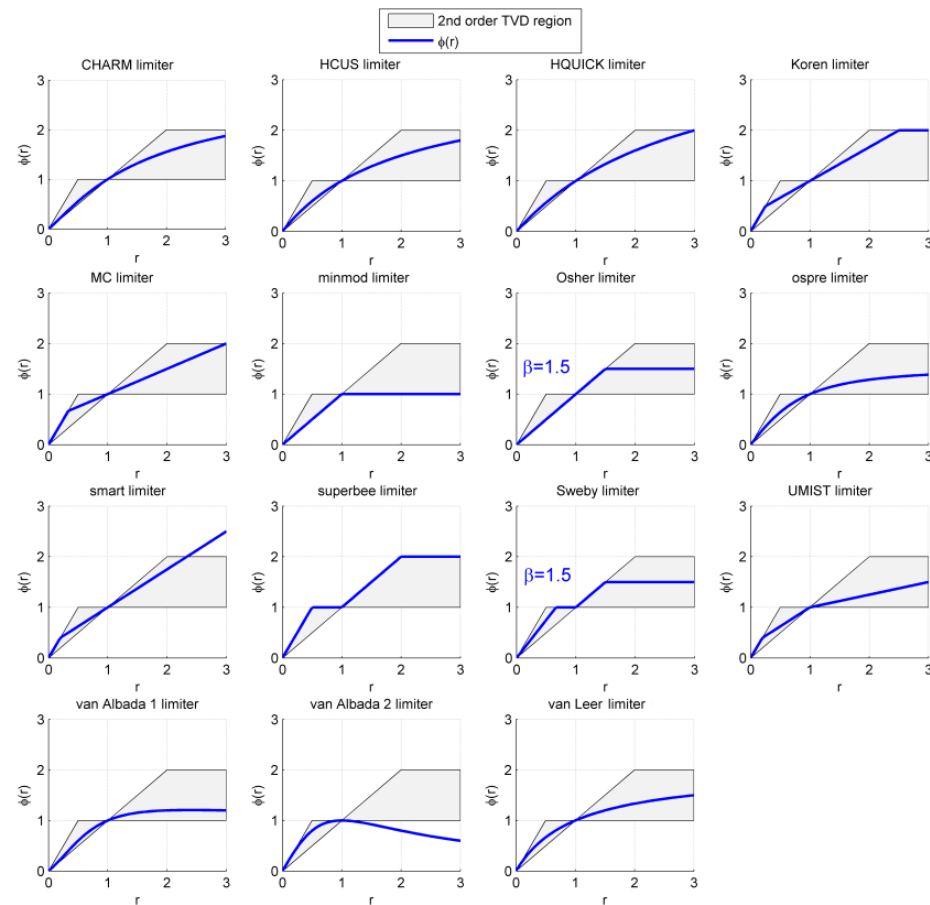
$$\phi_f = \begin{cases} \phi_P + \frac{1}{2}\psi_P^-(\phi_P - \phi_{PP}) & \text{for } \bar{F} \geq 0, \\ \phi_N - \frac{1}{2}\psi_P^+(\phi_{NN} - \phi_N) & \text{for } \bar{F} < 0. \end{cases}$$

UD = upwind  
SOU = second order upwind  
CD = central differencing  
D = downwind

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – TVD schemes

- Sweby diagram and TVD limiters.
- The fact that some limiters are non differentiable, and some others are differentiable can have an influence on the solution behavior (accuracy and convergence rate), specially when dealing with steady simulations.



**Limiter functions overlaid onto second-order TVD region**

<https://en.wikipedia.org/wiki/File:LimiterPlots1.png>

This work is licensed under a Creative Commons License (CC BY-SA 3.0)

**minmod** (by Roe, 1986)

$$\phi(r) = \max[0, \min(1, r)]$$

**superbee** (by Roe, 1986)

$$\phi(r) = \max[0, \min(2r, 1), \min(r, 2)]$$

**van Leer** (by van Leer, 1974)

$$\phi(r) = \frac{r + |r|}{1 + |r|}$$

Differentiable limiter  
(except at  $r = 0$ )

**venkatakrishnan** (by Venkatakrishnan. 1993)

$$\phi(r) = \frac{r^2 + 2r}{r^2 + r + 2}$$

Differentiable limiter

Non differentiable limiters

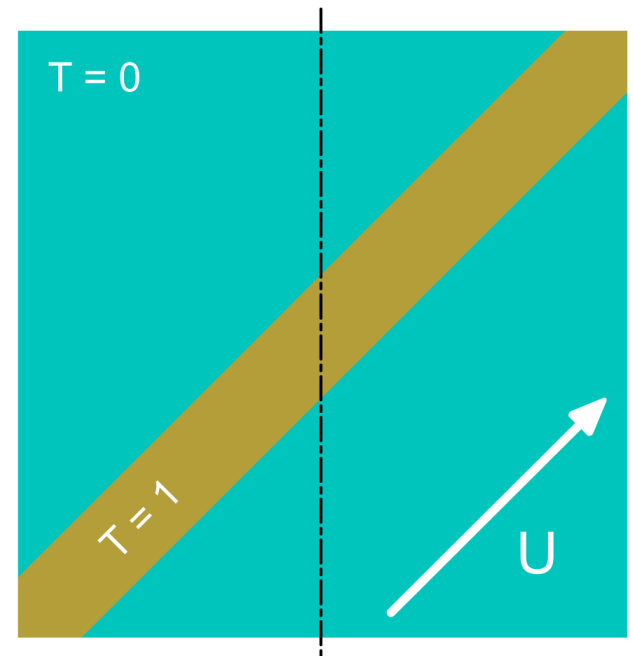
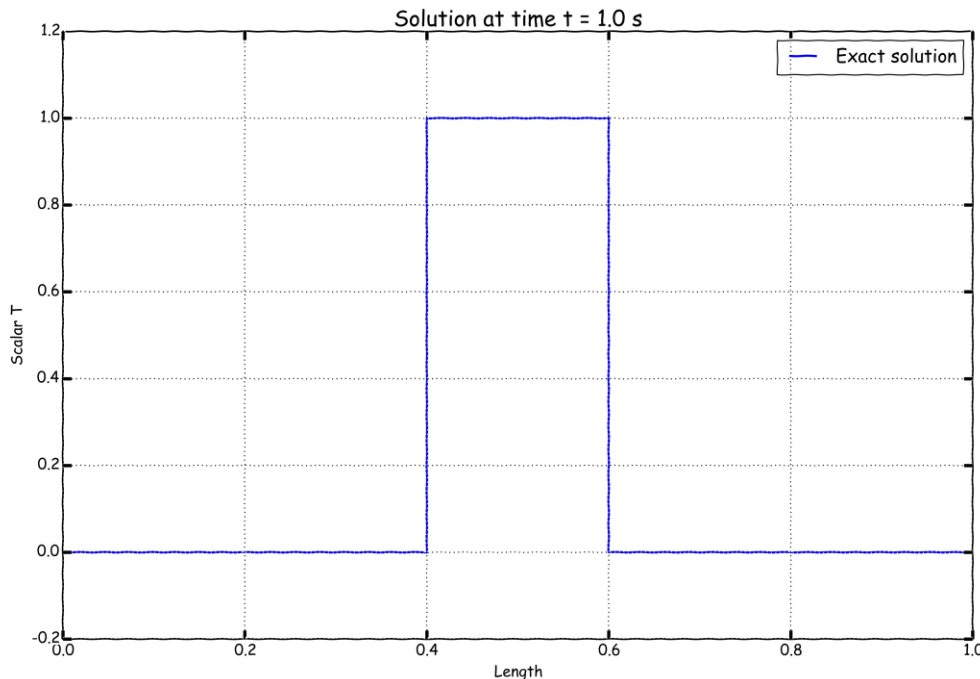
# The Finite Volume Method: An overview

**TVD schemes in action – A numerical schemes  
killer test case**

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – TVD schemes

- Let us see how the superbee, minmod and vanleer TVD schemes behave in a numerical schemes killer test case:
  - The oblique double step profile in a uniform vector field (pure convection).
- By the way, this problem has an exact solution.

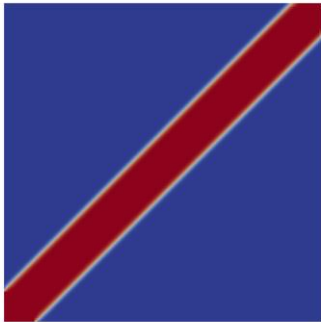


# The Finite Volume Method: An overview

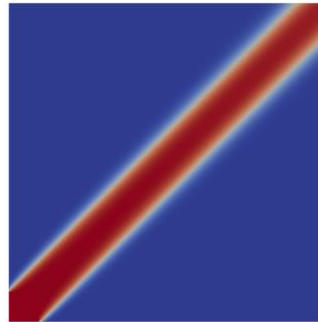
## Interpolation of the convective fluxes – Linear and non-linear limiter functions

- Comparison of non-linear limiter functions.
- All the following TVD schemes are second order accurate. However, the Minmod is a little bit more dissipative.

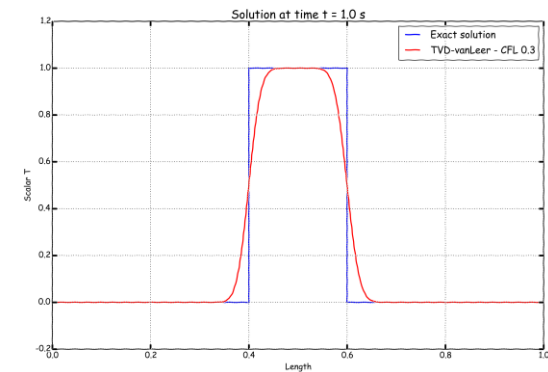
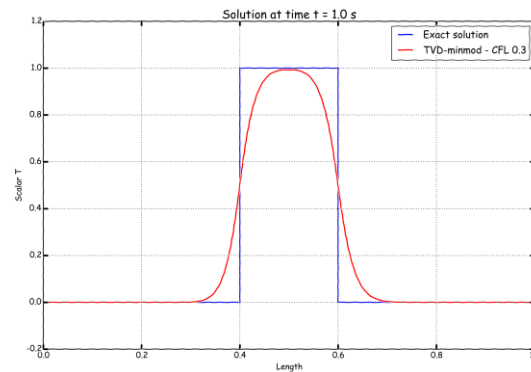
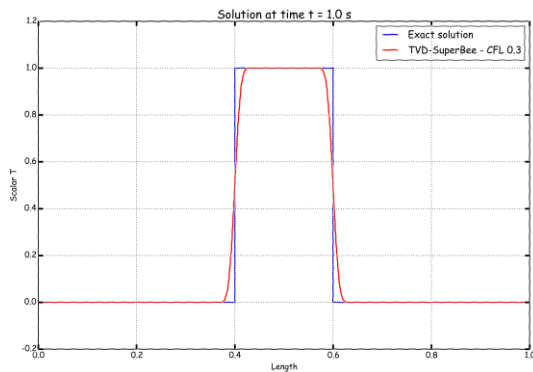
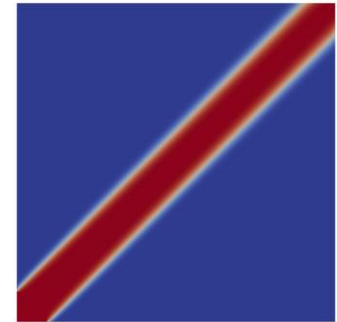
**SuperBee - Compressive**



**Minmod - Diffusive**



**vanLeer - Smooth**

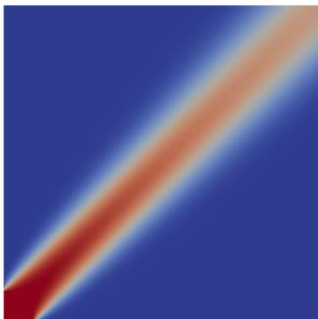


# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Linear and non-linear limiter functions

- Comparison of linear limiters (upwind and linear upwind) and non-linear limiters (SuperBee).
- Recall that the linear upwind method is 2<sup>nd</sup> order, and that the upwind method is 1<sup>st</sup> order.
- The upwind method is extremely stable and non-oscillatory. However, it is highly diffusive.
- On the other side, the linear upwind method is accurate but oscillatory in the presence of strong gradients.
- Remember, TVD methods switch locally to upwind when they detect strong gradients.

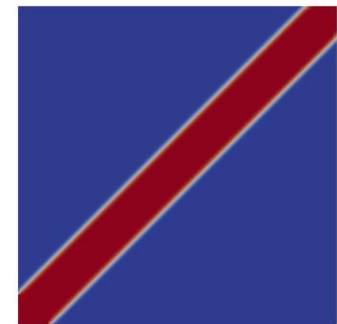
**Upwind – 1<sup>st</sup> order**



**Linear Upwind – 2<sup>nd</sup> order**



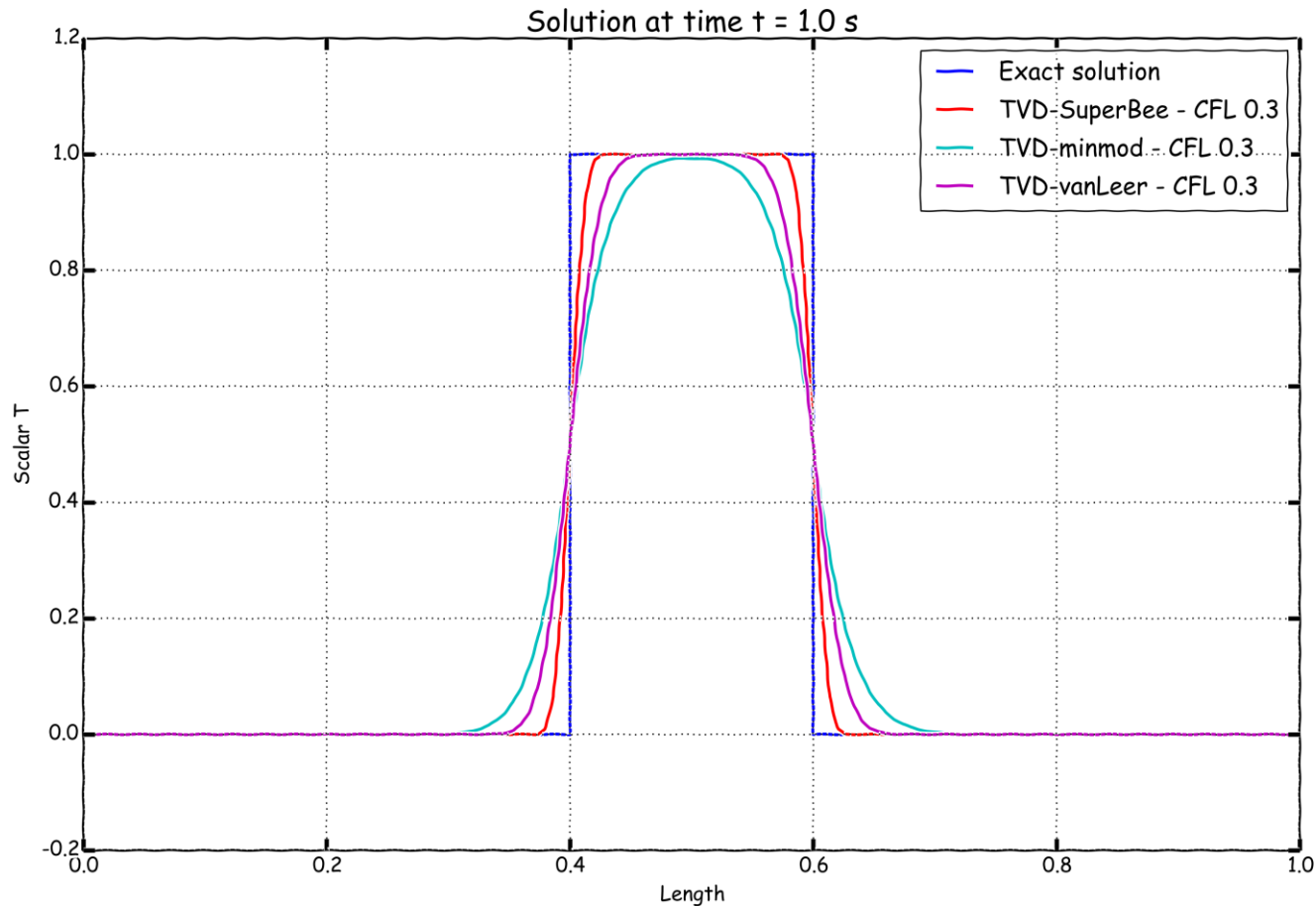
**SuperBee – TVD**



# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Linear and non-linear limiter functions

- Let us see how the non-linear limiter functions compare.

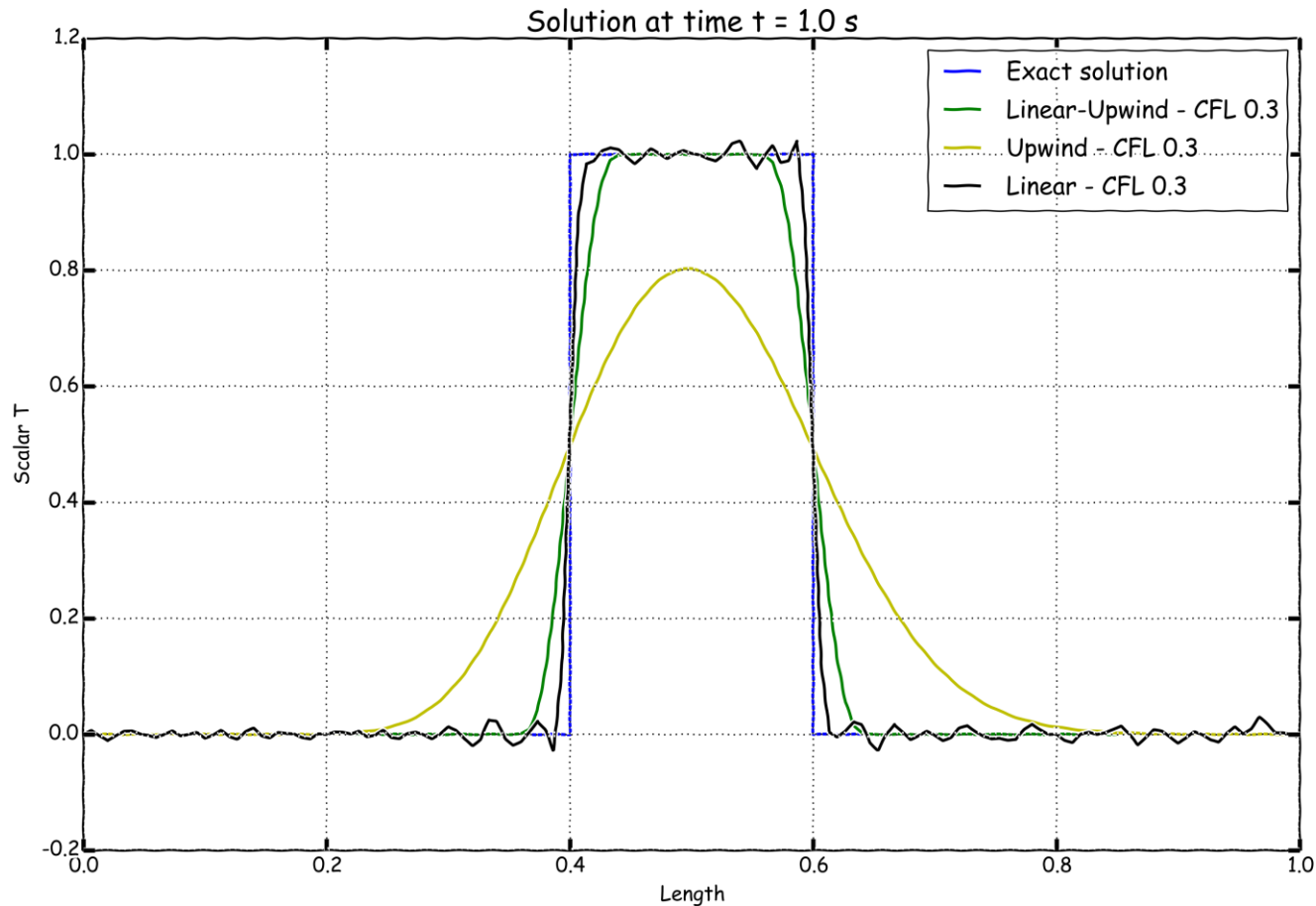




# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Linear and non-linear limiter functions

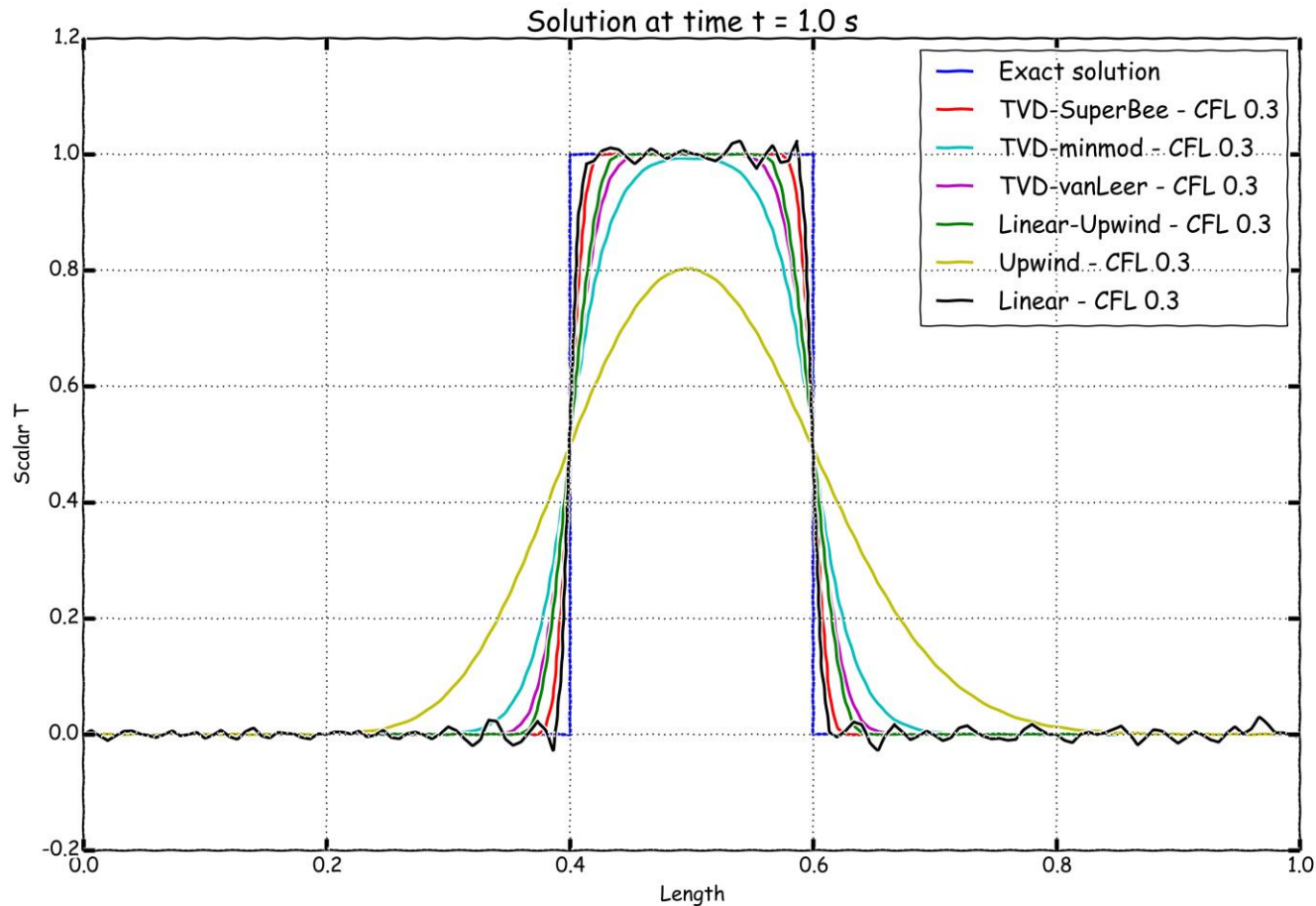
- Let us see how the linear limiter functions compare.



# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Linear and non-linear limiter functions

- Let us see how the linear and non-linear limiter functions compare.



# The Finite Volume Method: An overview

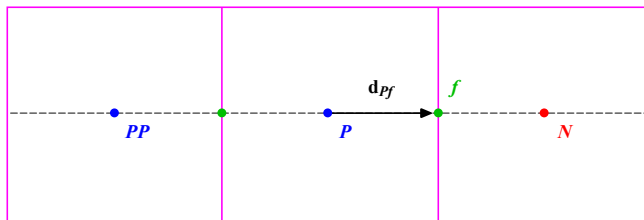
**Interpolation of the convective fluxes –  
Unstructured meshes**

# The Finite Volume Method: An overview

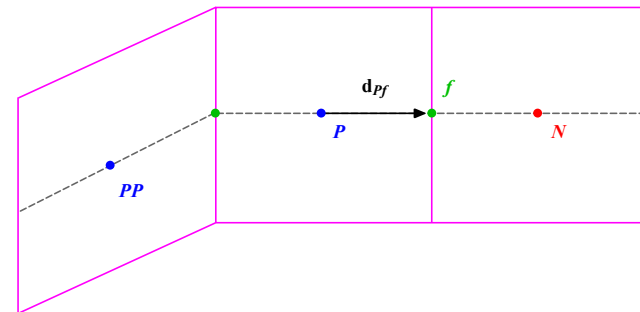
## Interpolation of the convective fluxes – Unstructured meshes

- All the high-order (HO) and high-resolution (HR) schemes we have seen so far, assume line structure (figure A). In other words, they are formulated in structured meshes (orthogonal meshes).
- In orthogonal meshes, the cell centers  $PP$ ,  $P$ , and  $N$  are all aligned (colinear). Therefore, constructing numerical stencils is relative straightforward.
- In unstructured meshes or when the cell centers are not colinear, the use of the previous schemes is not straightforward as the cell center  $PP$  is not aligned with the vector connecting cells  $P$  and  $N$  (figure B).
- High-order and high-resolution schemes for unstructured meshes are an area of active research and new ideas continue to emerge.

A



B



# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Unstructured meshes

- In unstructured meshes, the face centered values are computed by using the following flux-limited scheme,

$$\phi_f = \phi_U + \frac{1}{2}\psi(r_f^\pm)(\phi_D - \phi_U)$$

- Notice that the formulation of the flux-limited scheme is the same as the one used when the cell centers are colinear.
- The only difference is the way how the ratio of successive gradients  $r_f^\pm$  is computed.
- One way to compute  $r_f^\pm$  in unstructured meshes is by using the formulation presented in reference [1],

$$r_f^\pm = \frac{(2\nabla\phi_U \cdot \mathbf{d}_{UD})}{\phi_D - \phi_U} - 1$$

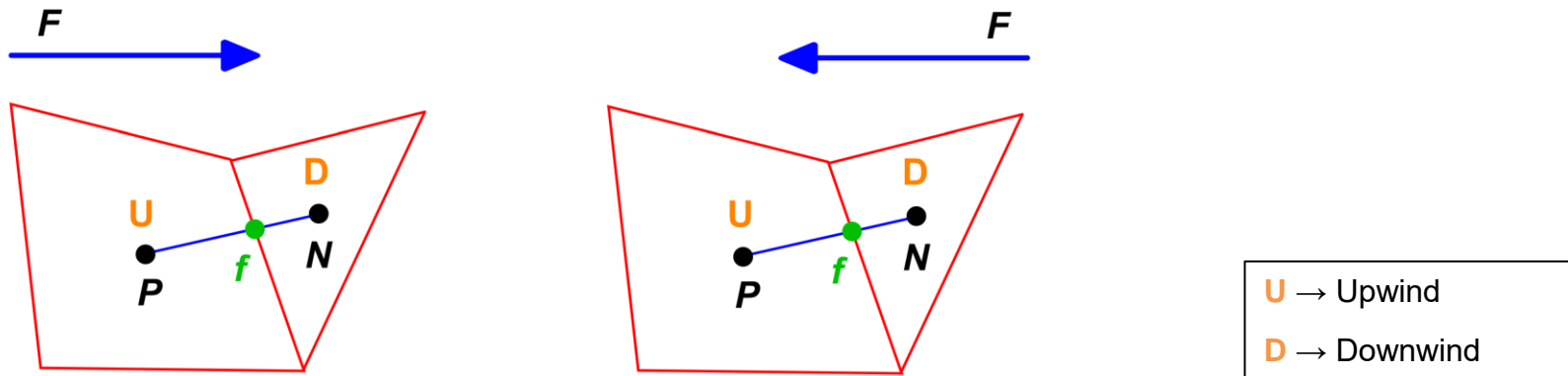
### Reference:

[1] Darwish, M. S., Moukalled, F., “TVD schemes for unstructured grids”

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Unstructured meshes

- The sub-index notation used in the ratio of successive gradients  $r_f^\pm$  [1] and in the general flux-limited scheme for unstructured meshes, is illustrated in the figure below.



- Where **D** stands for downwind cell center and **U** for upwind cell center.
- Notice that we can use the same relations regardless of the flow direction.
- We only need to be sure to use the right indices.

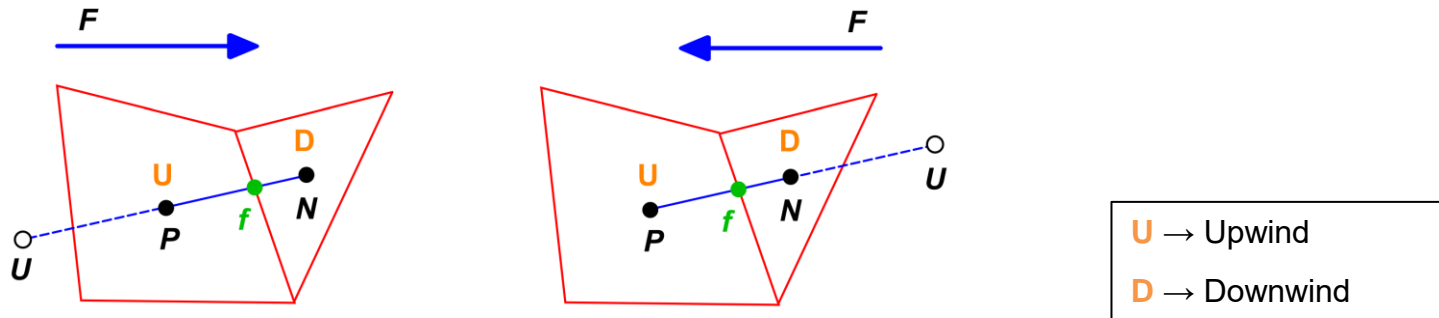
### Reference:

[1] Darwish, M. S., Moukalled, F., “TVD schemes for unstructured grids”

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Unstructured meshes

- The ratio of successive gradients  $r_f^\pm$  [1], can be derived as follows.



- First, let us add a virtual node  $U$  upstream of  $P$  (for  $F > 0$ ), in such a way that the cell center  $P$  splits the vector  $UN$  in half (a similar reasoning can be used for  $F < 0$ ).
- Recall that when the cell centers are colinear, the ratio of successive gradients  $r_f$  can be computed as follows (for  $F > 0$ ),

$$r_f = \frac{\phi_P - \phi_U}{\phi_N - \phi_P}$$

### Reference:

[1] Darwish, M. S., Moukalled, F., "TVD schemes for unstructured grids"

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Unstructured meshes

- Let us add and subtract  $\phi_N$  to  $r_f$ , such that,

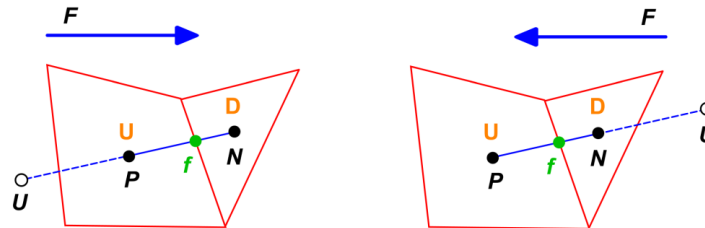
$$r_f = \frac{\phi_N + (\phi_P - \phi_U) - \phi_N}{\phi_N - \phi_P} = \frac{(\phi_N - \phi_U) - (\phi_N - \phi_P)}{\phi_N - \phi_P}$$

- By using Taylor expansions, the term  $(\phi_N - \phi_U)$  can be approximated as follows,

$$(\phi_N - \phi_U) = \nabla\phi_P \cdot \mathbf{d}_{UN} = 2\nabla\phi_P \cdot \mathbf{d}_{PN}$$

- And after some algebra, we obtain the following expression for  $r_f$

$$r_f = \frac{(2\nabla\phi_P \cdot \mathbf{d}_{PN})}{\phi_N - \phi_P} - 1$$



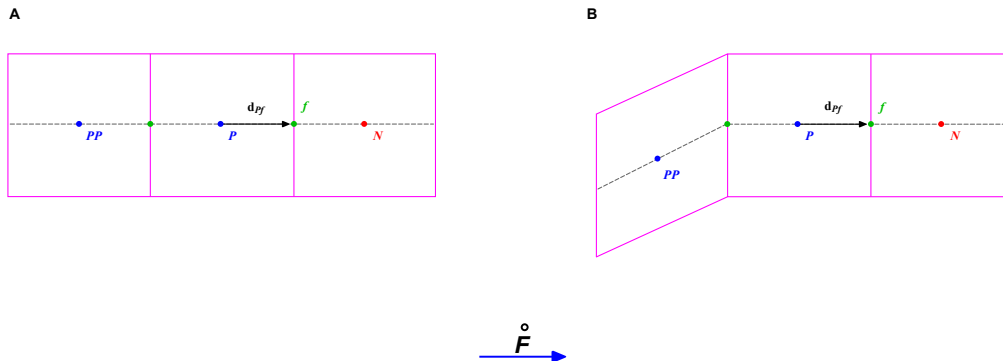
U → Upwind  
D → Downwind



# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Unstructured meshes

- There are many ways to compute the ratio of successive gradients in unstructured meshes.
- We just presented one formulation [1].
- This is an area of active research where new ideas continue to emerge.
- Thanks to the treatment presented for unstructured meshes, we ended with a compact numerical stencil very friendly for unstructured CFD solvers.
- At this point, you should have realized why computing accurate gradients is so important in CFD.
- It is also possible to use the wide stencil used for collinear cells centers, but the reconstruction of the value at the cell PP can be very cumbersome.



$$r_f = \frac{\phi_P - \phi_{PP}}{\phi_N - \phi_P}$$

For structured meshes or colinear cell centers – Wide stencil

$$r_f = \frac{(2\nabla\phi_P \cdot \mathbf{d}_{PN})}{\phi_N - \phi_P} - 1$$

For unstructured meshes – Compact stencil

### Reference:

[1] Darwish, M. S., Moukalled, F., “TVD schemes for unstructured grids”

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Unstructured meshes

- To summarize:
  - A simple way to interpolate the convective fluxes in unstructured meshes is by redefining HO and HR schemes in terms of gradients at the control volume center  $\mathbf{P}$  and the face center  $\mathbf{f}$ .
  - In unstructured meshes, the face centered values are computed using the same flux-limited scheme as for structured meshes,

$$\phi_f = \phi_U + \frac{1}{2}\psi(r_f^{\pm})(\phi_D - \phi_U)$$

- The main difference is the way how the ratio of successive gradients  $r_f^{\pm}$  is computed.
- The limiter function  $\psi(r)$  remains the same as for structured meshes.
- That is, the Sweby diagram is the same.

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Unstructured meshes

- To summarize:
  - For example, using the gradient at the cell center  $\mathbf{P}$  and the face center  $\mathbf{f}$ , we can compute the face values as follows (upwind bias formulations),

Upwind  $\rightarrow \phi_f = \phi_P$

$$\psi(r) = 0$$

Central difference  $\rightarrow \phi_f = \phi_P + \nabla \phi_f \cdot \mathbf{d}_{Pf}$

$$\psi(r) = 1$$

Second order upwind differencing  $\rightarrow \phi_f = \phi_P + (2\nabla \phi_P - \nabla \phi_f) \cdot \mathbf{d}_{Pf}$

$$\psi(r) = r$$

QUICK  $\rightarrow \phi_f = \phi_P + 1/2 (\nabla_P + \nabla \phi_f) \cdot \mathbf{d}_{Pf}$

$$\psi(r) = \frac{3+r}{4}$$

- Notice that in this new formulation the cell PP does not appear anymore, we are using compact stencils.
- The problem now turns in the accurate evaluation of the gradients at the cell and face centers.
- So, as long as the computation of the gradients is second order accurate, it does not matter the way they are computed.
- For example, the gradients at the cell centers can be computed using the Gauss method, and then interpolated to the face centers.
- At this point, we are only missing the reconstruction of the cell center gradients at the face centers, this is explained latter.

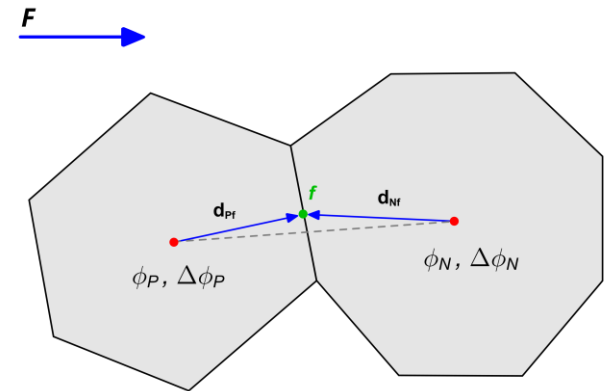
# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Unstructured meshes

- Another popular reconstruction technique is the Barth and Jespersen method [1]. Here, it is assumed that the solution is piecewise linearly distributed over the control volume.
- According to the flow direction, the left or right state at the face  $f$  can be found using the following relations,

$$\phi_f = \phi_P + \psi_f \nabla \phi_P \cdot \mathbf{d}_{Pf} \quad \text{if } F \geq 0$$

$$\phi_f = \phi_N + \psi_f \nabla \phi_N \cdot \mathbf{d}_{Nf} \quad \text{if } F < 0$$



- In the previous relations,  $\psi_f$  denotes a limiter function at the face (gradient or slope limiter), which is used to avoid over and under shoots on the gradient computations.
- Popular limiter functions are: Minmod, Barth-Jespersen, Venkatakrishnan.
- This linear reconstruction is likely the most popular among the reconstruction methods, and it is implemented in most commercial CFD solvers.

### Reference:

[1] Barth, T. J., Jespersen, D. C., "The Design and Application of Upwind Schemes on Unstructured Meshes"

# The Finite Volume Method: An overview

## Interpolation of the convective fluxes – Unstructured meshes

- It worth mentioning that the slope limiter function and the flux limiter (as in TVD schemes), are related according to the following relationship [1],

$$\psi(r) = \phi(r) \left( \frac{r + 1}{2} \right)$$

- Where  $\psi$  is the flux limiter and  $\phi$  is the slope limiter function.
- It can be easily seen that the method of Barth and Jespersen [2] corresponds to a Taylor-series expansion around the face center.
- This linear reconstruction is formally second order accurate provided the gradient  $\nabla\phi$  is evaluated accurately.
- The superbee and Barth-Jespersen limiters are the most compressive and are known to turn smooth waves into square waves.
- In multiple dimensions their overly compressive nature may lead to staircasing of discontinuities that are not aligned with the grid.

### Reference:

- [1] Spekreijse, S., "Multigrid Solution of Monotone Second-Order Discretizations of Hyperbolic Conservation Laws"
- [2] Barth, T. J., Jespersen, D. C., "The Design and Application of Upwind Schemes on Unstructured Meshes"

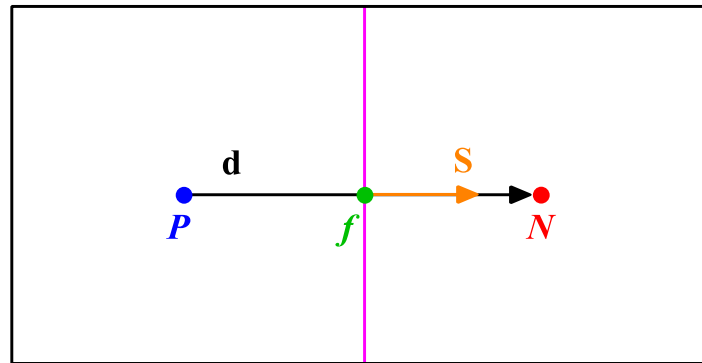
# The Finite Volume Method: An overview

**Interpolation of diffusive fluxes**

# The Finite Volume Method: An overview

## Interpolation of diffusive fluxes in an orthogonal mesh

- By looking the figures below, the face values appearing in the diffusive flux in an orthogonal mesh can be computed as follows,



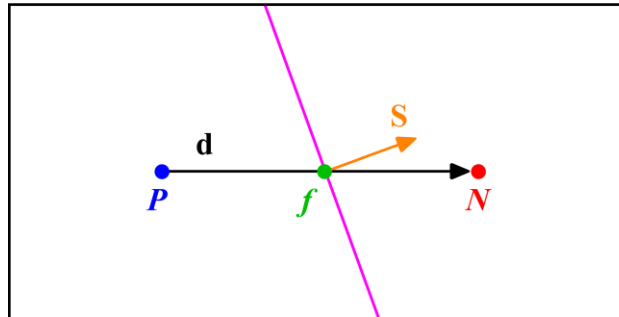
$$\mathbf{S} \cdot (\nabla \phi)_f = |\mathbf{S}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}.$$

- This is a central difference approximation of the first order derivative.
- This type of approximation is second order accurate.

# The Finite Volume Method: An overview

## Interpolation of diffusive fluxes in a non-orthogonal mesh

- In reference to the figure below, recall that non-orthogonality is the angle between the vector **S** normal to the face **f**, and the vector **d** connecting the two cell centers **P** and **N**.
- In non-orthogonal meshes (meshes with a non-orthogonal angle), using a central difference approximation is not good enough.
- In non-orthogonal meshes, the problem relies in the fact that we want to compute the face gradients in the same direction as the vector **d**, but we cannot because **d** and **S** are not parallel.
- From the point of view of the formulation of the diffusive flux using the FVM, the gradient **S** normal to the face cannot be written purely in terms of a gradient in the direction **d**.
- Therefore, due to non-orthogonality, we need to use an approach different to central differences to approximate the face gradients. We need to add some kind of correction.
- In the figure below, the non-orthogonal angle is equal to  $20^\circ$ .
- Now imagine how the cells will look like if the angle is more than  $80^\circ$  or more. You can also imagine angles greater than  $90^\circ$  or the critical angle where the cell volume is equal to zero, *i.e.*,  $90^\circ$ .
- Keeping non-orthogonality low is very important when generating the mesh.

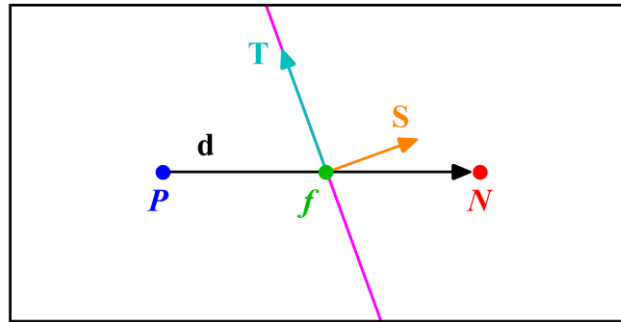




# The Finite Volume Method: An overview

## Interpolation of diffusive fluxes in a non-orthogonal mesh

- The non-orthogonal angle, gives rise to a secondary gradient **T** tangential at the face **f**.
- The gradient normal to the face (orthogonal contribution) can be approximated using central differences.
- The secondary gradient (non-orthogonal contribution) need to be computed somehow.
- From the figure and equation below, the face values appearing in the diffusive flux of a non-orthogonal mesh can be computed as a correction of the orthogonal contribution and the non-orthogonal contribution, as follows,



Using **T** and **d**, we can now write the gradient **S** normal to the face **f**.

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{\Delta_{\perp} \cdot (\nabla \phi)_f}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}} \quad \text{where} \quad \Delta_{\perp} \cdot (\nabla \phi)_f = |\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}$$

- This type of approximation is second order accurate but involves a larger truncation error.
- It also uses a larger numerical stencil, which make it less stable.

# The Finite Volume Method: An overview

## Correction of diffusive fluxes in a non-orthogonal mesh

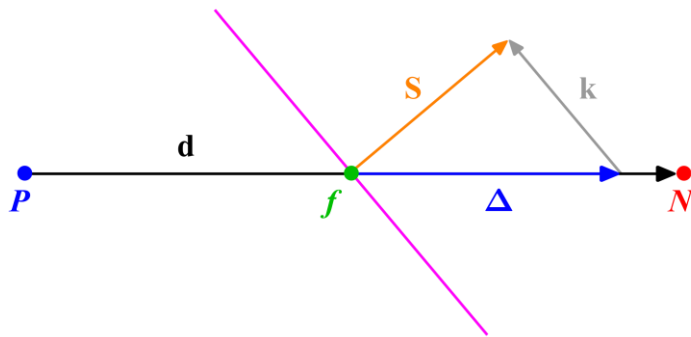
- The literature is very rich when it comes to correcting the non-orthogonal contribution.
- This is an area of active research and new ideas continues to emerge.
- For more information about correction of diffusive fluxes in non-orthogonal meshes, the interested reader can refer to the following references [1-8].
- In reference [8], Jasak gives a geometric interpretation to different correction approaches found in literature. He gave to the approaches the following names,
  - Minimum correction approach.
  - Orthogonal correction approach.
  - Over-relaxed approach.
- Of these approaches, the over-relaxed approach is the most widely used.
- The interested reader can peruse reference [3] for a review of different methods and a derivation of the algebra behind the afore mentioned geometrical interpretations.

- [1] S. R. Mathur, J. Y. Murthy. A Pressure-Based Method for Unstructured Meshes. Numer. Heat Transfer, Vol. 31, 1997.
- [2] L. Davidson. A Pressure Correction Method for Unstructured Meshes with Arbitrary Control Volumes. Int. J. Numer. Methods in Fluids. Vol. 22, 1996.
- [3] I. Demirdzic. On the Discretization of the Diffusion Term in Finite-Volume Continuum Mechanics. Numerical Heat Transfer, Part B: Fundamentals. Vol. 68, 2015.
- [4] J. H. Ferziger, M. Peric. Further discussion of numerical errors in CFD. Int. J. Numer. Methods in Fluids, Vol. 23, pp. 1263-1274, 1996.
- [5] W. J. Minkowycz, E. M. Sparrow, J. Y. Murthy. Handbook of Numerical Heat Transfer. Chapter 1. John Wiley & Sons. 2000.
- [6] H. K. Versteeg, W. Malalasekera. An Introduction to Computational Fluid Dynamics. Prentice Hall. 2000.
- [7] J. Ferziger, M. Peric, R. Street. Computational Methods for Fluid. Springer. 2001.
- [8] H. Jasak. Error analysis and estimation in the Finite Volume method with applications to fluid flows. PhD Thesis. Imperial College, London. 1996.

# The Finite Volume Method: An overview

## Correction of diffusive fluxes in a non-orthogonal mesh

- By looking the figure below, the face values appearing in the diffusive flux in a non-orthogonal mesh ( $40^\circ$  in the image below) can be computed as follows.
- Using the over-relaxed approach, the diffusive fluxes can be corrected as,



Over-relaxed approach

$$\Delta_{\perp} = \frac{d}{d \cdot S} |S|^2$$

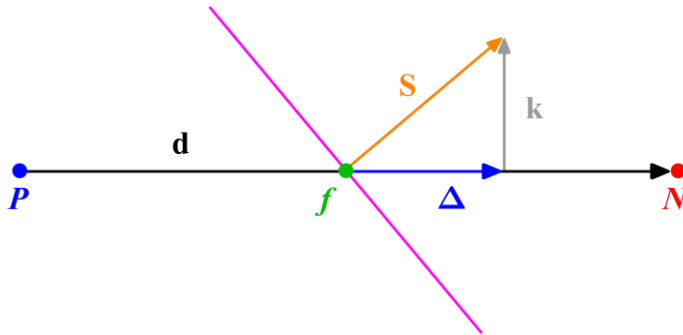
$$S = \Delta_{\perp} + k \longrightarrow k = S - \Delta_{\perp}$$

$$S \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|d|}}_{\text{orthogonal contribution}} + \underbrace{k \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}$$

# The Finite Volume Method: An overview

## Correction of diffusive fluxes in a non-orthogonal mesh

- By looking the figure below, the face values appearing in the diffusive flux in a non-orthogonal mesh ( $40^\circ$  in the image below) can be computed as follows.
- Using the minimum correction approach, the diffusive fluxes can be corrected as,



Minimum correction approach

$$\Delta_{\perp} = \frac{\mathbf{d} \cdot \mathbf{S}}{\mathbf{d} \cdot \mathbf{d}} \mathbf{d}$$

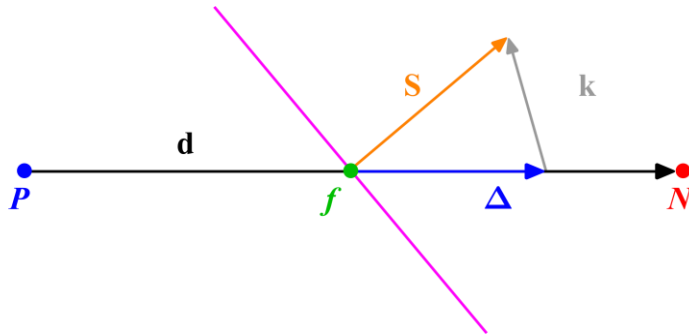
$$\mathbf{S} = \Delta_{\perp} + \mathbf{k} \longrightarrow \mathbf{k} = \mathbf{S} - \Delta_{\perp}$$

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}$$

# The Finite Volume Method: An overview

## Correction of diffusive fluxes in a non-orthogonal mesh

- By looking the figure below, the face values appearing in the diffusive flux in a non-orthogonal mesh ( $40^\circ$  in the image below) can be computed as follows.
- Using the orthogonal correction approach, the diffusive fluxes can be corrected as,



Orthogonal correction approach

$$\Delta_{\perp} = \frac{d}{|d|} |S|$$

$$\mathbf{S} = \Delta_{\perp} + \mathbf{k} \longrightarrow \mathbf{k} = \mathbf{S} - \Delta_{\perp}$$

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|d|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}$$

# The Finite Volume Method: An overview


## Correction of diffusive fluxes in a non-orthogonal mesh

- The secondary face gradients (non-orthogonal contribution) that arises from the discretization of the diffusive flux on non-orthogonal meshes, somehow need to be reconstructed from the cell center to the face center, this is explained in the next section.

- Secondary gradient due to mesh non-orthogonality.
- This gradient needs to be evaluated at the face center.

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}.$$

**Implicit part** **Explicit part**



- It is clear that if the mesh is orthogonal, you do not need to do any correction.
- Therefore, you can compute the gradients using centered differences (but this is the exception rather than the rule).
- When solving the NSE, non-orthogonality mainly affects the pressure equation, and in the case of compressible flows, it also affects the energy equation.
- Generally speaking, every equation where the Laplacian operator is present will be sensitive to mesh non-orthogonality.
- From the previous discussion, it is clear why we want to avoid large non-orthogonal angles.

# The Finite Volume Method: An overview

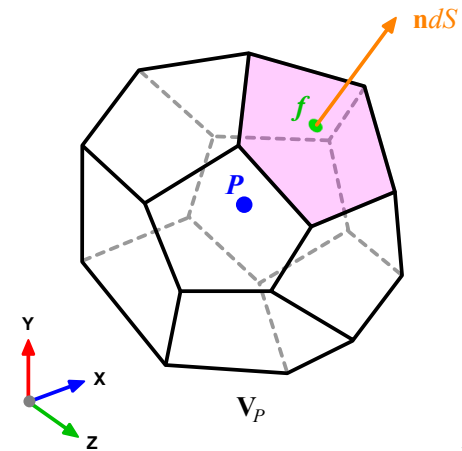
**Gradient computation at cell centers and  
gradient reconstruction at face centers**

# The Finite Volume Method: An overview

## Gradients computation at cell centers

- There are many methods for the computation of the cell centered gradients, e.g., least squares, Gauss cell-based, Gauss node-based, and so on.
- Using the Gauss cell-based method, the cell centered gradients can be computed as follows,

$$(\nabla \phi)_P = \frac{1}{V_P} \sum_f (\mathbf{s}_f \phi_f)$$



$$\mathbf{n}dS = d\mathbf{S}$$

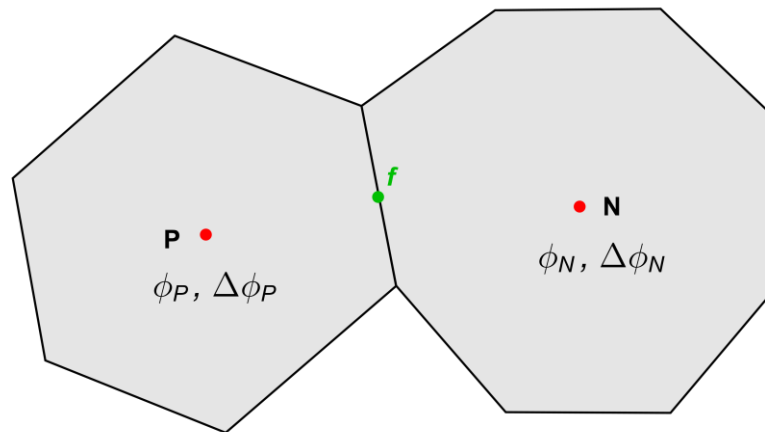
- This approximation is second order accurate given that the mesh quality is acceptable, and the volume of the cell is finite.
- In general, the least squares method tends to be more accurate.



# The Finite Volume Method: An overview

## Gradients reconstruction at face centers

- Face gradients  $\nabla \phi_f$  arise from the discretization process of the convective and diffusive terms.
- These secondary gradients are due to non-orthogonality and skewness in the pressure and energy equations (or any equation containing the diffusion term).
- They also appear when computing the face quantities in unstructured meshes.
- Have in mind that there are many methods to reconstruct (or interpolate) the face gradients, this is an area of active research.
- Hereafter we are going to show a few ways to do so.

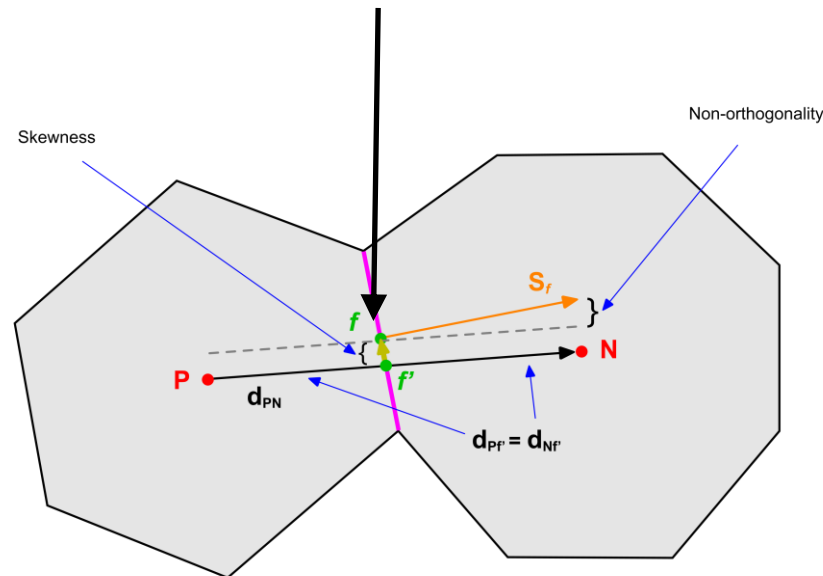


# The Finite Volume Method: An overview

## Gradients reconstruction at face centers

- The easiest way to reconstruct the face gradient  $\nabla\phi_f$  is by taking the average of the cell centered gradients  $\nabla\phi_P$  and  $\nabla\phi_N$ .
- However, this approach may be inaccurate in non-uniform, non-orthogonal and skew meshes (general unstructured meshes).

$$\nabla\phi_f = \frac{\nabla\phi_P + \nabla\phi_N}{2}$$

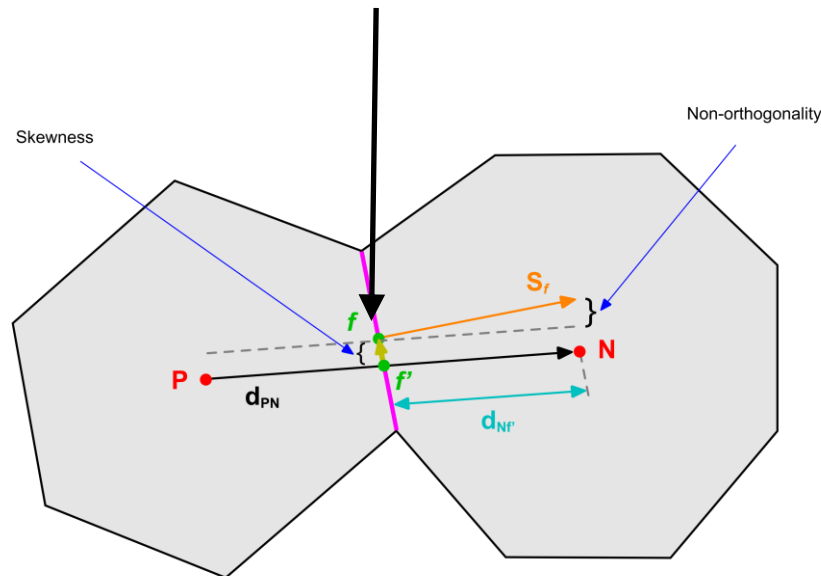


# The Finite Volume Method: An overview

## Gradients reconstruction at face centers

- Another way to reconstruct the face gradient  $\nabla\phi_f$  is by using weighted interpolation of the cell centered quantities  $\nabla\phi_P$  and  $\nabla\phi_N$ .
- Again, this approach may be inaccurate in meshes with high degree of non-orthogonality and skewness.

$$\nabla\phi_f = f_x \nabla\phi_P + (1 - f_x) \nabla\phi_N \quad \text{where} \quad f_x = fN/PN$$



# The Finite Volume Method: An overview

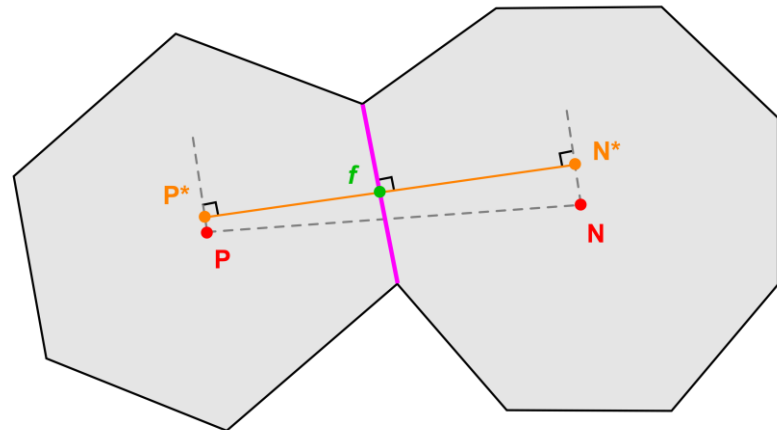
## Gradients reconstruction at face centers

- Yet another approach more accurate than the previous ones is by reconstructing the cell centered quantities in such a way that they create a vector that is normal to the face and passes thru its center.
- Starting from the cell centered quantities, the face gradients  $\nabla\phi_f$  can be reconstructed as follows:
  - First, reconstruct the cell centered quantities at the points  $P^*$  and  $N^*$ , as follows,

$$\phi_{N^*}^* = \phi_N + \nabla\phi_N \cdot \mathbf{d}_{N^*N} \qquad \phi_{P^*}^* = \phi_P + \nabla\phi_P \cdot \mathbf{d}_{P^*P}$$

- Then evaluate the face gradient along the vector  $\mathbf{d}_{P^*N^*}$  (which is normal to the face  $f$ ), as follows (you can also use weighted interpolation),

$$\nabla\phi_f = \frac{\phi_{N^*}^* - \phi_{P^*}^*}{|\mathbf{d}|_{P^*N^*}}$$

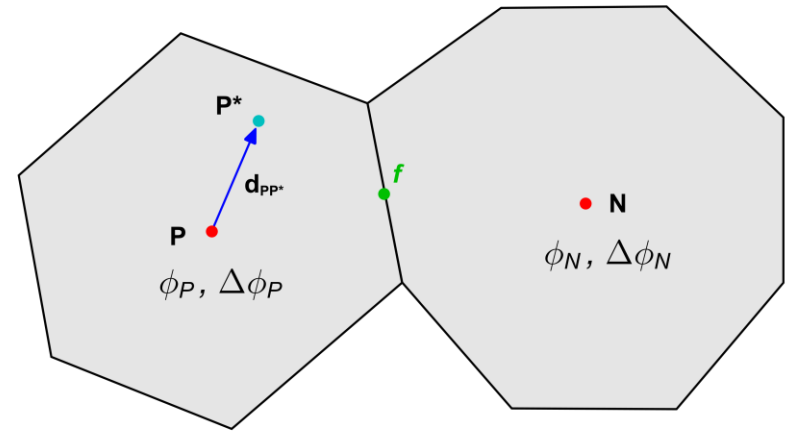


# The Finite Volume Method: An overview

## Gradients reconstruction at face centers

- In the previous formulation, recall that any cell centered quantity  $\phi_P$  can be reconstructed in a new location  $\mathbf{P}^*$  (within the cell volume), as follows,

$$\phi_{P^*} = \phi_P + \nabla \phi_P \cdot \mathbf{d}_{PP^*}$$

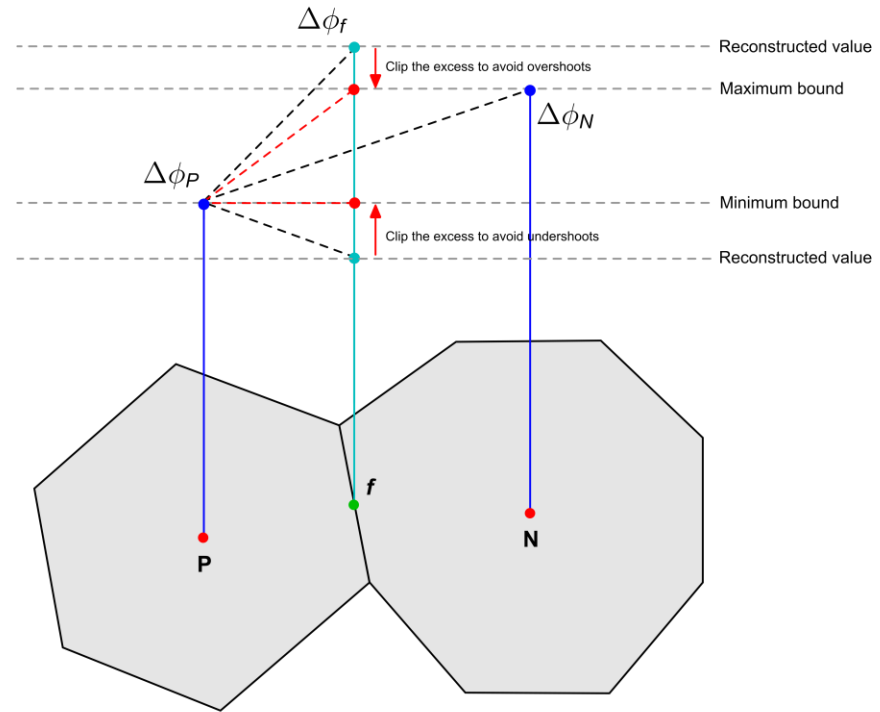


- All the previous approximations are second order accurate in good quality meshes.
- Also, the use of non-orthogonal corrections suggests the adoption of an iterative method to compute better face gradient approximations.

# The Finite Volume Method: An overview

## Gradients reconstruction at face centers

- As for cell centered variables, when reconstructing the gradients at the face centers it might happen that they become unbounded.
- So, to avoid over and under shoots on the gradient computations, we use gradient limiters (or slope limiters).
- By avoiding over and under shoots we are enforcing the monotonicity principle.
- Gradient limiters increase the stability of the method but might add diffusion due to clipping.
- The idea behind gradient limiters is similar to that of the limiters used in TVD schemes.
- For a more detailed discussion on gradient limiters, the interested reader should review references [1-3].



# The Finite Volume Method: An overview

**Iterative approach to compute cell centered and face centered gradients**

# The Finite Volume Method: An overview

## Iterative approach to compute cell centered and face centered gradients

- From the discussion on interpolation of diffusive fluxes, we have seen that to compute the viscous fluxes on non-orthogonal meshes we need to know the face gradients.

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{\Delta_{\perp} \cdot (\nabla \phi)_f}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}} \quad \text{where} \quad \Delta_{\perp} \cdot (\nabla \phi)_f = |\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}$$

**Implicit part****Explicit part**

- In this decomposition, the orthogonal contribution is treated implicitly (unknowns in the LHS), and the non-orthogonal contribution is treated explicitly (known or old values in the RHS).
- The secondary gradient in the diffusive fluxes (due to non-orthogonality), somehow needs to be computed and corrected, as discussed in the previous sections.
- The computation of the secondary gradient terms requires the knowledge of gradients at the cell centers.
- And the computation of the cell centered gradients requires the face values  $\phi_f$ , and to compute face values we need face gradients (at least in unstructured non-orthogonal meshes).
- This suggests the need of using an iterative approach for computing successively better approximations of the  $\phi_f$  values and the gradients at cell centers and face centers.



# The Finite Volume Method: An overview

## Iterative approach to compute cell centered and face centered gradients

- One way to iteratively compute the cell centered and face centered gradients is as follows.
- For the computation of the cell centered gradients we can use, for example, the Gauss cell-based method, which reads as follows,

$$(\nabla \phi)_P = \frac{1}{V_P} \sum_f (\mathbf{S}_f \phi_f)$$

- As a first approximation, the face value  $\phi_f$  can be computed as the average of the two cells values sharing the face so that,

$$\phi_f = \frac{\phi_0 + \phi_1}{2}$$

- Once the derivative has been obtained using the Gauss cell-based method (or any other method), the initial approximation of the gradient at the face center may be successively improved by reconstructing it from the cell value using any of the approaches described in the previous section (gradients reconstruction at face centers).
- Remember there are many alternatives to compute the cell centered gradients, *e.g.*, least squares, Gauss node-based, and so on.
- The guys developing videos games and dealing with rendering have developed very advanced methods for gradients computation.

# The Finite Volume Method: An overview

## Iterative approach to compute cell centered and face centered gradients

- In the previous steps, we can add an improvement to the face values  $\phi_f$  computation.
- Once we have obtained the cell gradient, we can improve our initial approximation of the face average by reconstructing a new value from the cell center values, as follows,

$$\phi_f = \frac{(\phi_P + \Delta_{Pf} \cdot \nabla \phi_P) + (\phi_N + \Delta_{Nf} \cdot \nabla \phi_N)}{2}$$

- At this point, it only rest iterate to obtain better approximations by looping.
- During each iteration, we can compute the face weighted average gradient using the cell gradient computed in the previous iteration and use these face values to compute new values of the gradients.

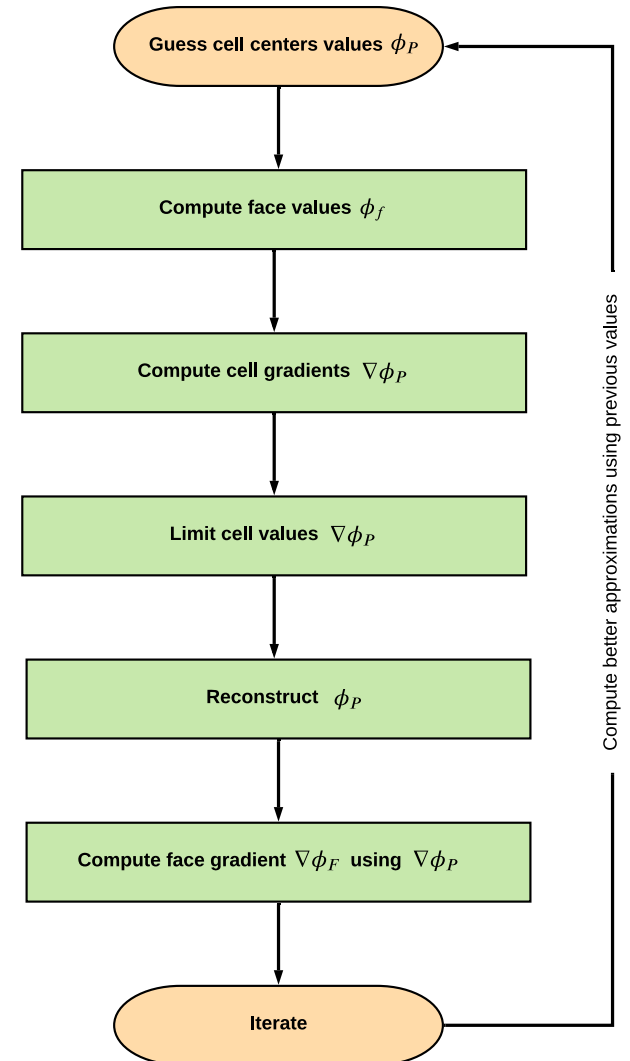
$$\nabla \phi_f = f_x \nabla \phi_P + (1 - f_x) \nabla \phi_N \quad \text{where} \quad f_x = f_N / P_N$$

- In addition, the gradients can be limited to avoid overshoots and undershoots on the solution (monotonicity principle).
- This is just one method to deal with the computations of cell centered and face centered gradients.

# The Finite Volume Method: An overview

## Iterative approach to compute cell centered and face centered gradients

- By iteratively applying the previous steps, the accuracy of the face values and gradients can be improved.
- During each iteration, we can compute the face gradient using the cell gradient and then use these face values to compute the diffusive or convective fluxes.
- As you can see, increasing the number of iterations will improve gradients computations.
- In practice, two or three iterations are sufficient to obtain accurate gradients.
- But depending on the physics involved, we might need to do more, or maybe less iterations.
- In general, it is recommended to do at least one iteration.
- By the way, the gradients computation is not only limited to the diffusive terms.
- Gradients are also required for the construction of higher-order convection operators, as well as many physical models (e.g., turbulence models, multiphase models, non-Newtonian viscosity models, and so on).
- **Accurate, robust, and stable computation of gradients is extremely important in CFD.**



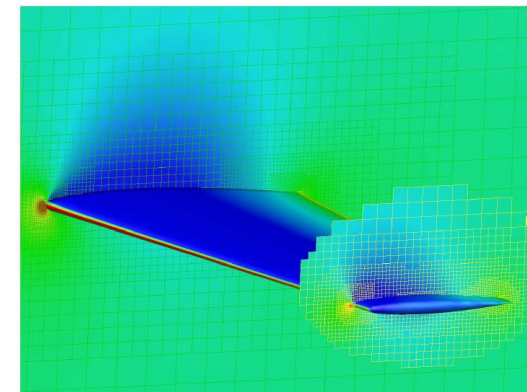
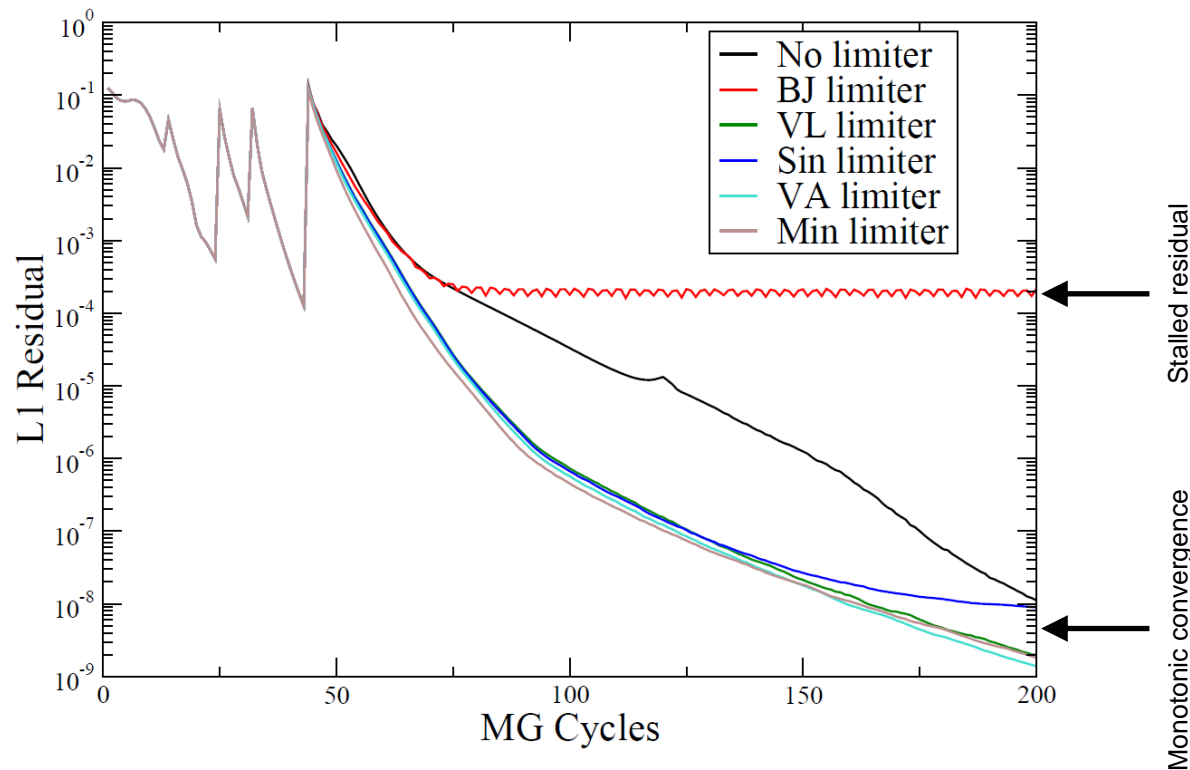
# The Finite Volume Method: An overview

**Effect of gradient limiters on solution accuracy  
and convergence to steady state**

# The Finite Volume Method: An overview

## Effect of gradient limiters on solution accuracy and convergence to steady state

- The non-differentiable nature of some limiters can adversely affect convergence to steady state.
- In some cases, they are responsible for the stalled residuals even if the solution is converging.
- In some other cases, they can add a lot of numerical diffusion to the solution.



Onera M6 Wing ( $Ma = 0.5$ ,  $AOA = 3.06$ )

Limiter Type	Drag Coefficient $\times 10^3$
No limiter	3.24
BJ limiter	4.56
VL limiter	5.41
Sin limiter	5.59
VA limiter	6.15
Min limiter	8.15

### Reference:

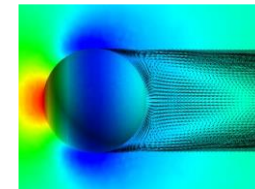
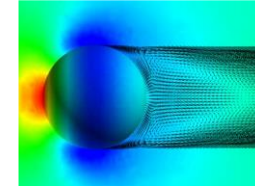
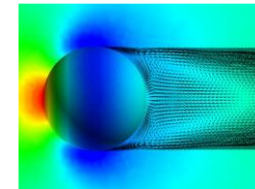
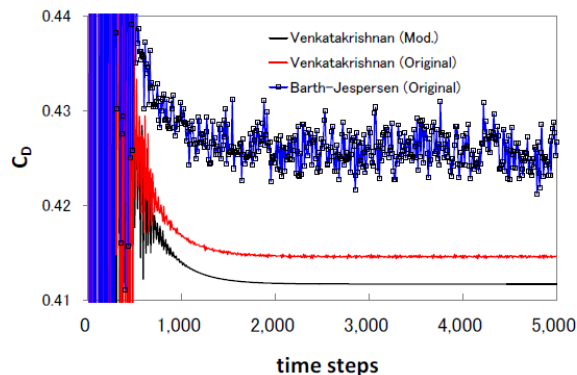
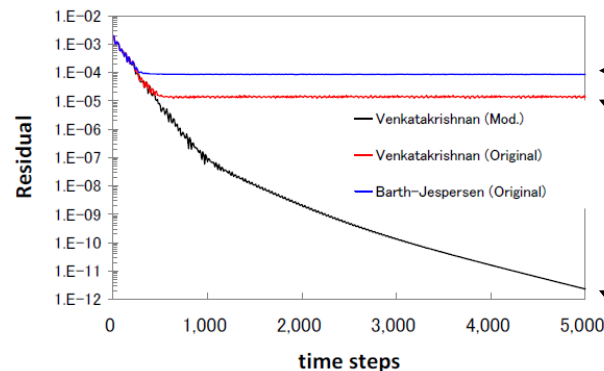
M. Berger, M. Aftosmis, S. Murman, "Analysis of Slope Limiters on Irregular Grids"

Computed drag for different limiter formulations, in order of increasing dissipation associated with the limiter. 102

# The Finite Volume Method: An overview

## Effect of gradient limiters on solution accuracy and convergence to steady state

- Illustration of gradient limiters effect on the convergence to steady state of a sample case – Viscous flow over sphere at low Reynolds number (Steady simulation).
- The use of limiters (for gradients and fluxes) to obtain second-order TVD schemes is a powerful and robust approach. There are further issues to be considered, such as accuracy and convergence issues resulting from clipping, systems of equations, multiple dimensions, unstructured meshes, higher-order time-marching methods and so on.



### Reference:

K. Kitamura, E. Shima, "Simple and Parameter-Free Second Slope Limiter for Unstructured Grid Aerodynamic Simulations"

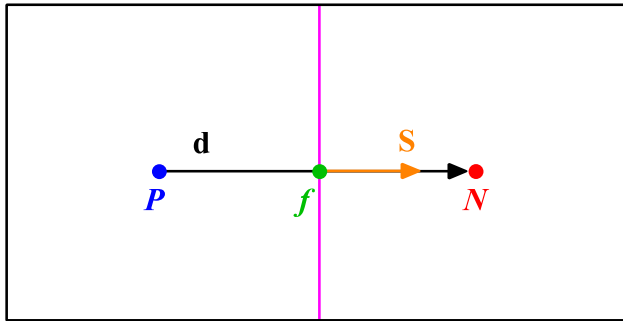
# The Finite Volume Method: An overview

**Mesh induced errors**

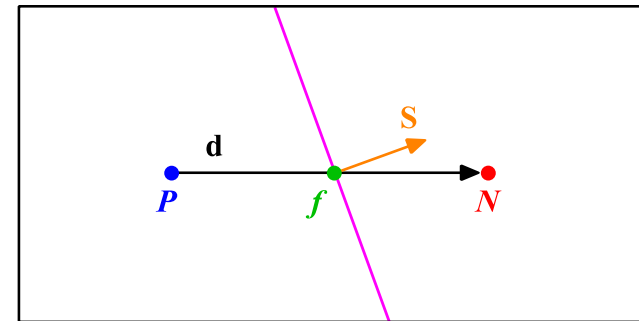
# The Finite Volume Method: An overview

## Mesh induced errors

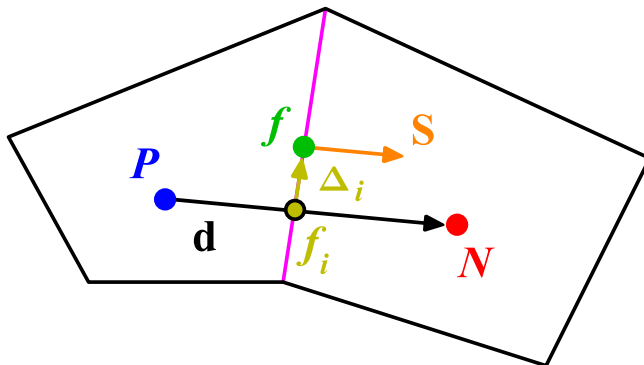
- In order to maintain second order accuracy, and to avoid unboundedness, we need to correct non-orthogonality and skewness errors.
- The ideal case is to have an orthogonal and non skew mesh, but this is the exception rather than the rule.
- The best practice to minimize mesh induced errors is to generate good quality meshes.



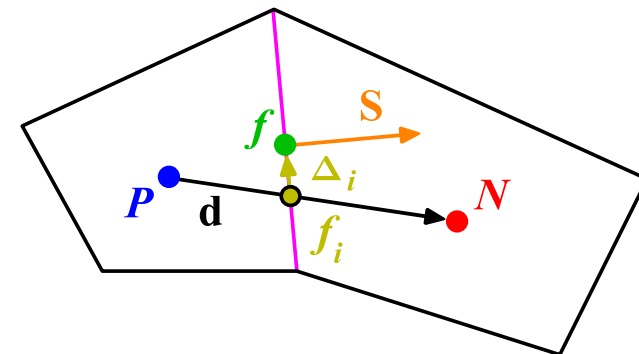
Orthogonal and non skew mesh



Non-orthogonal and non skew mesh



Orthogonal and skew mesh



Non-orthogonal and skew mesh



# The Finite Volume Method: An overview

**Time discretization**

# The Finite Volume Method: An overview

## Time discretization

- Using the previous equations to evaluate the general transport equation over all the control volumes, we obtain the following semi-discrete equation,

$$\underbrace{\int_{V_P} \frac{\partial \rho \phi}{\partial t} dV}_{\text{temporal derivative}} + \sum_f \underbrace{\mathbf{S}_f \cdot (\rho \mathbf{u} \phi)_f}_{\text{convective flux}} - \sum_f \underbrace{\mathbf{S}_f \cdot (\rho \Gamma_\phi \nabla \phi)_f}_{\text{diffusive flux}} = \underbrace{(S_c V_P + S_p V_P \phi_P)}_{\text{source term}}$$

where  $\mathbf{S} \cdot (\rho \mathbf{u} \phi) = F^C$  is the convective flux and  $\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi) = F^D$  is the diffusive flux.

- After spatial discretization, we can proceed with the time discretization. By proceeding in this way, we are using the Method of Lines (MOL).
- The main advantage of the MOL method, is that it allows us to select numerical approximations of different accuracy for the spatial and time terms. Each term can be treated differently to yield to different accuracies.

# The Finite Volume Method: An overview

## Time discretization

- Now, we evaluate in time the semi-discrete general transport equation

$$\int_t^{t+\Delta t} \left[ \left( \frac{\partial \rho \phi}{\partial t} \right)_P V_P + \sum_f \mathbf{S}_f \cdot (\rho \mathbf{u} \phi)_f - \sum_f \mathbf{S}_f \cdot (\rho \Gamma_\phi \nabla \phi)_f \right] dt$$
$$= \int_t^{t+\Delta t} (S_c V_P + S_p V_P \phi_P) dt.$$

- At this stage, we can use any time discretization scheme, e.g., Crank-Nicolson, Euler implicit, forward Euler, backward differencing, adams-bashforth, adams-moulton.
- It should be noted that the order of the time discretization of the transient term does not need to be the same as the order of the discretization of the spatial terms.
- Each term can be treated differently to yield different accuracies.
- So, as long as the individual terms are at least second order accurate, the overall accuracy will also be second order.

# The Finite Volume Method: An overview

**Linear system solution – Crunching numbers**

# The Finite Volume Method: An overview

## Linear system solution

- After spatial and time discretization and by using equation

$$\int_t^{t+\Delta t} \left[ \left( \frac{\partial \rho \phi}{\partial t} \right)_P V_P + \sum_f \mathbf{S}_f \cdot (\rho \mathbf{u} \phi)_f - \sum_f \mathbf{S}_f \cdot (\rho \Gamma_\phi \nabla \phi)_f \right] dt = \int_t^{t+\Delta t} (S_c V_P + S_p V_P \phi_P) dt$$

in every control volume  $V_P$  of the domain, a system of linear algebraic equations (LAE) for the transported quantity  $\phi$  is assembled,

= Diagonal contribution  
 = Off-diagonal contribution

$$\begin{bmatrix}
 a_1 & \square & & \square & & \\
 \square & \blacksquare & \square & & \square & \\
 & \ddots & \ddots & \ddots & \ddots & \\
 \ddots & & \ddots & \ddots & \ddots & \ddots \\
 & \square & & \square & a_P & \square \\
 & & \ddots & \ddots & \ddots & \ddots \\
 & & & \square & \square & \blacksquare & \square \\
 & & & & \square & \square & a_N
 \end{bmatrix}
 \times
 \begin{bmatrix}
 \phi_1 \\
 \vdots \\
 \phi_P \\
 \vdots \\
 \phi_N
 \end{bmatrix}
 =
 \begin{bmatrix}
 b_1 \\
 \vdots \\
 b_P \\
 \vdots \\
 b_N
 \end{bmatrix}$$

$\mathbf{A} \phi = \mathbf{b}$

# The Finite Volume Method: An overview

## Linear system solution

- In CFD, the fast and efficient solution of the following system is of paramount importance.

$$\begin{array}{c}
 \boxed{\text{Matrix of coefficients}} \leftarrow \mathbf{A}\phi = \mathbf{b} \rightarrow \boxed{\text{Boundary conditions and source terms vector}} \\
 \downarrow \\
 \boxed{\text{Solution vector}}
 \end{array}$$

- This system can be solved by using any iterative or direct method.
- But in practice, iterative methods are used most of the times.
- An equation for each cell is assemble, where the contribution in the diagonal of  $\mathbf{A}$  corresponds to  $a_p$ , and the off-diagonal contribution corresponds to the neighboring elements  $a_{np}$  (elements that shares a face with  $a_p$ ).

$$\begin{array}{l}
 \text{Equation for cell 1} \longrightarrow \\
 \text{Equation for cell P} \longrightarrow \\
 \text{Equation fir cell N} \longrightarrow
 \end{array}
 \begin{bmatrix}
 a_1 & \square & & \square & & \\
 \square & \blacksquare & \square & & \square & \\
 & \ddots & \ddots & \ddots & \ddots & \\
 \ddots & & \ddots & \ddots & \ddots & \ddots \\
 & \square & & \square & a_P & \square \\
 & & \ddots & \ddots & \ddots & \ddots \\
 & & & \square & \square & \blacksquare & \square \\
 & & & & \square & \square & a_N
 \end{bmatrix}
 \times
 \begin{bmatrix}
 \phi_1 \\
 \vdots \\
 \phi_P \\
 \vdots \\
 \phi_N
 \end{bmatrix}
 =
 \begin{bmatrix}
 b_1 \\
 \vdots \\
 b_P \\
 \vdots \\
 b_N
 \end{bmatrix}$$

$\blacksquare$  = Diagonal contribution  
 $\square$  = Off-diagonal contribution

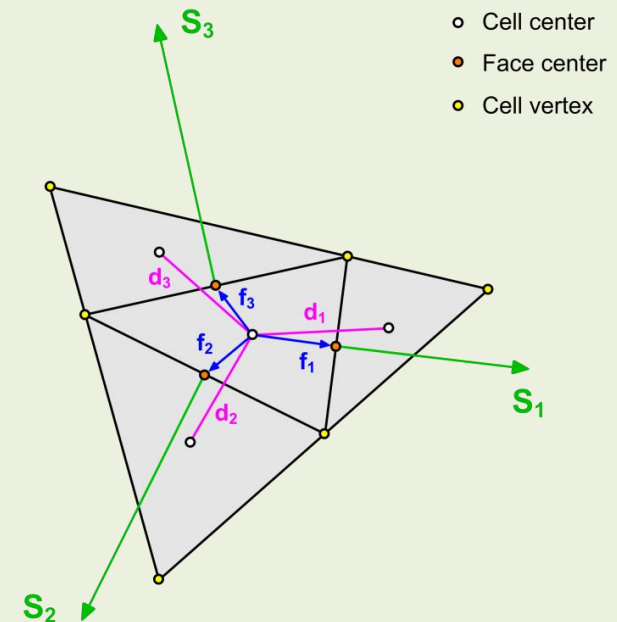
# The Finite Volume Method: An overview

## Linear system solution

- The matrix of coefficients  $\mathbf{A}$  of the discretized system of algebraic equations  $\mathbf{A}\phi = \mathbf{b}$  mostly depends on the geometry quantities.
- Specifically, on the dot product of  $\mathbf{S}$  (vector normal to face passing by the face center) and  $\mathbf{d}$  (vector connecting two cell centers), that is,  $\mathbf{S} \cdot \mathbf{d}$
- This dependence on the dot product  $\mathbf{S} \cdot \mathbf{d}$  is because the coefficients contain the following term,

$$\frac{\mathbf{S} \cdot \mathbf{S}}{\mathbf{S} \cdot \mathbf{d}}$$

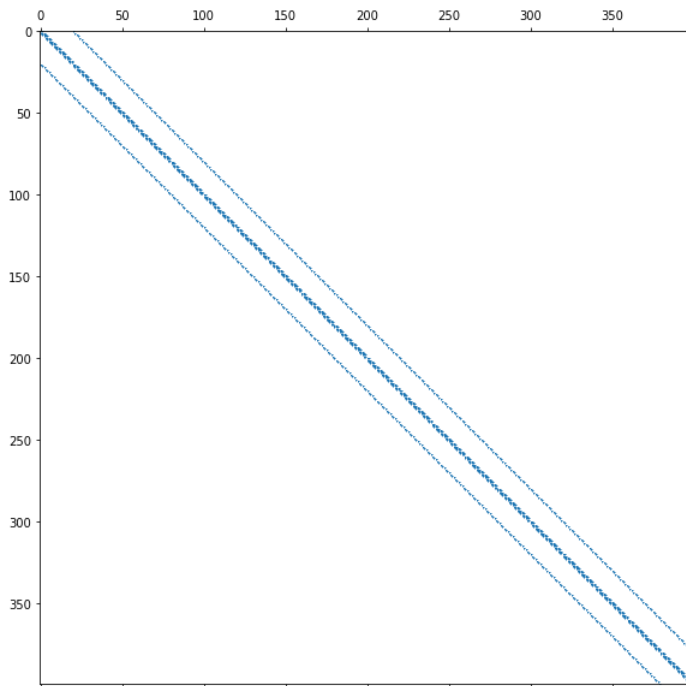
- For orthogonal meshes (perfect ones), the dot product is equal to one (there is no deviation between the vectors  $\mathbf{S}$  and  $\mathbf{d}$ ).
- The more a cell deviates from its perfect shape, the smaller the dot product becomes, and this results in large values of the matrix coefficients which increases the system stiffness.
- For very bad quality cells (e.g., very skew cells or cells with zero volume), this vector product may become zero, producing an undefined system (throwing a division by zero error).
- One single bad quality cell can make the solution diverge.



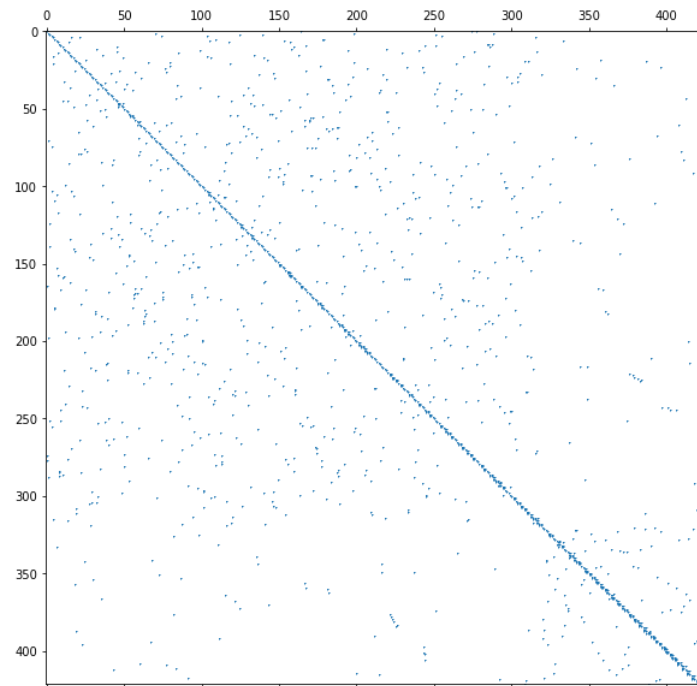
# The Finite Volume Method: An overview

## Linear system solution

- The matrices arising from the discretization of the governing equations are usually very large and sparse (they contain only a few non-zero elements).
- Banded sparse matrices tends to help convergence rate.
- In the figures below, the unknown quantity  $\phi_P$  is distributed along the diagonal.
- The off-diagonal entries, represent the contribution of the neighboring cells  $\phi_{nb}$



Sparse matrix – Banded type  
Typical of orthogonal meshes



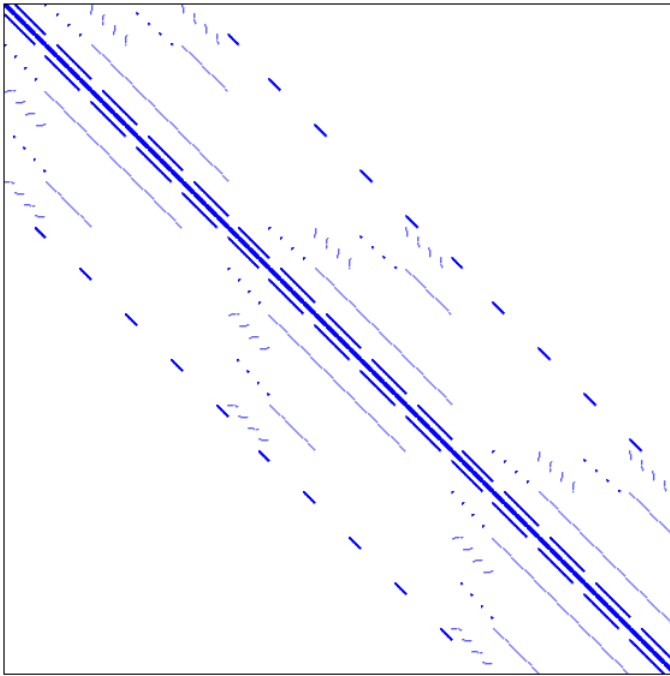
Sparse matrix – Non-banded structure  
Typical of general unstructured meshes



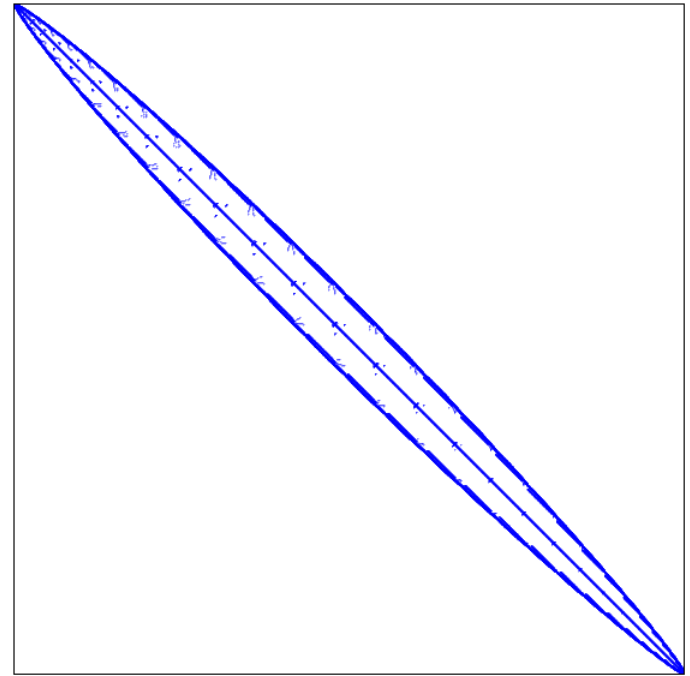
# The Finite Volume Method: An overview

## Linear system solution

- As we are solving a sparse matrix, the more diagonal the matrix is, the best the convergence rate will be.
- Linear solvers can be accelerated by using matrix reordering techniques that make the matrices more diagonally dominant.



Matrix structure plot before reordering



Matrix structure plot after reordering

### Note:

This is the actual pressure matrix from an OpenFOAM model case

# The Finite Volume Method: An overview

## Linear system solution

- In CFD, it is extremely important that the matrix **A** is diagonally dominant.
- A matrix is diagonally dominant if in each row the sum of the off-diagonal coefficient magnitude is equal or smaller than the diagonal coefficient,

$$a_{ii} \geq \sum_{j=1}^N |a_{ij}| \Rightarrow j \neq i$$

- And at least one  $i$ ,

$$a_{ii} \geq \sum_{j=1}^N |a_{ij}| \Rightarrow j \neq i$$

- Diagonal dominance is a very desirable feature for satisfying the boundedness criterion.
- To achieve diagonal dominance, we need large values of net coefficient (coefficients of the diagonal).
- This can be controlled by using under-relaxation, reducing the time-step, by assuring that any source term in the RHS is negative, and by having good quality meshes.
- If a matrix is diagonally dominant, it also satisfy the Scarborough criterion.

# The Finite Volume Method: An overview

## Linear system solution

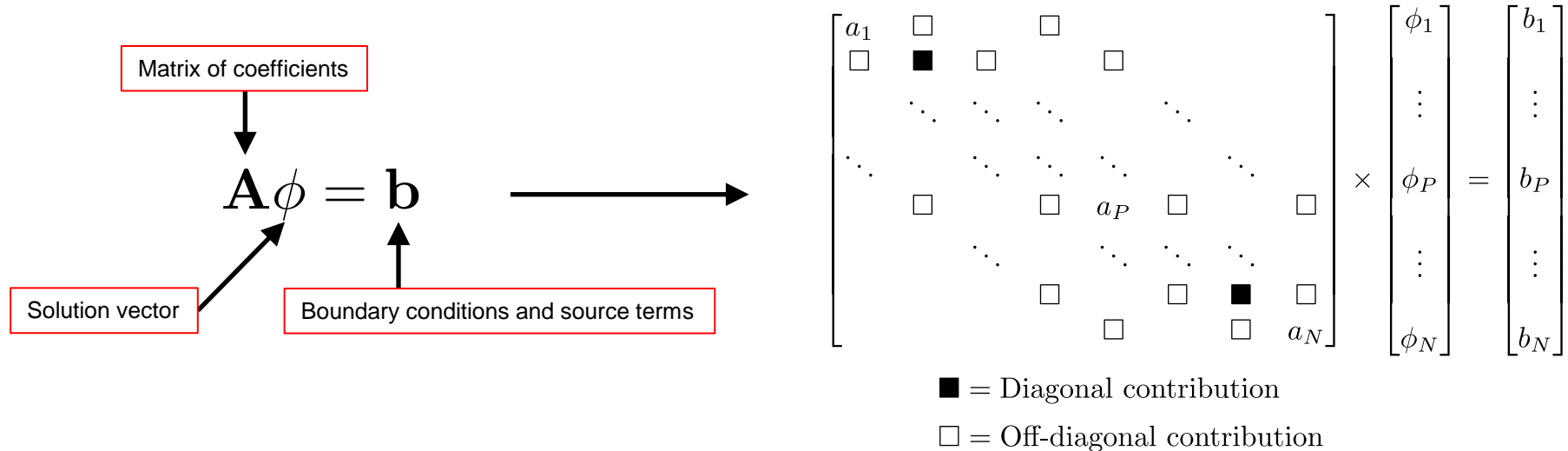
- If a matrix is diagonally dominant, it also satisfy the Scarborough criterion [1].

$$\frac{\sum |a_{nb}|}{|a_p|} \begin{cases} \leq 1, \text{ for all equations} \\ < 1, \text{ for at least one equation} \end{cases}$$

- The satisfaction of this criterion ensures that the equations will converge by at least one iterative method.
- This is a sufficient condition, not a necessary one. This means that we can get convergence, even if, at times, we violate this criterion.
- For example, if Scarborough criterion is not satisfied then Gauss–Seidel method iterative procedure is not guaranteed to converge to a solution.
- The finite volume method uses this criterion to set some basic discretization rules related to obtaining a convergent solution, implementing boundary conditions, and adding source terms.
  - When linearizing the source terms, they must be negative, so when they are added to  $a_p$  in the LHS, they help increasing the diagonal dominance.
  - All coefficients in the LHS and RHS of the linear system should have the same sign (essential requirement for boundedness).
  - If the boundedness requirement is not satisfied, it is possible that the solution does not converge at all, or if it does, the solution is oscillatory (contains wiggles).

# The Finite Volume Method: An overview

## Linear system solution



- After assembly the linear system, the solver will spend a great amount of time solving it.
- This system is solved using iterative solvers, where the algorithm starts from an initial guess and keeps iteration until reaching the desired convergence criterion.

$$\phi^{(0)} \Rightarrow \phi^{(1)} \Rightarrow \phi^{(2)} \Rightarrow \dots \Rightarrow \phi^{(i)} \Rightarrow \phi^{(Final)}$$

- Basically, iterative solvers incrementally reduce the error, until reaching a given residual  $\mathbf{r}$  (absolute or relative tolerance),

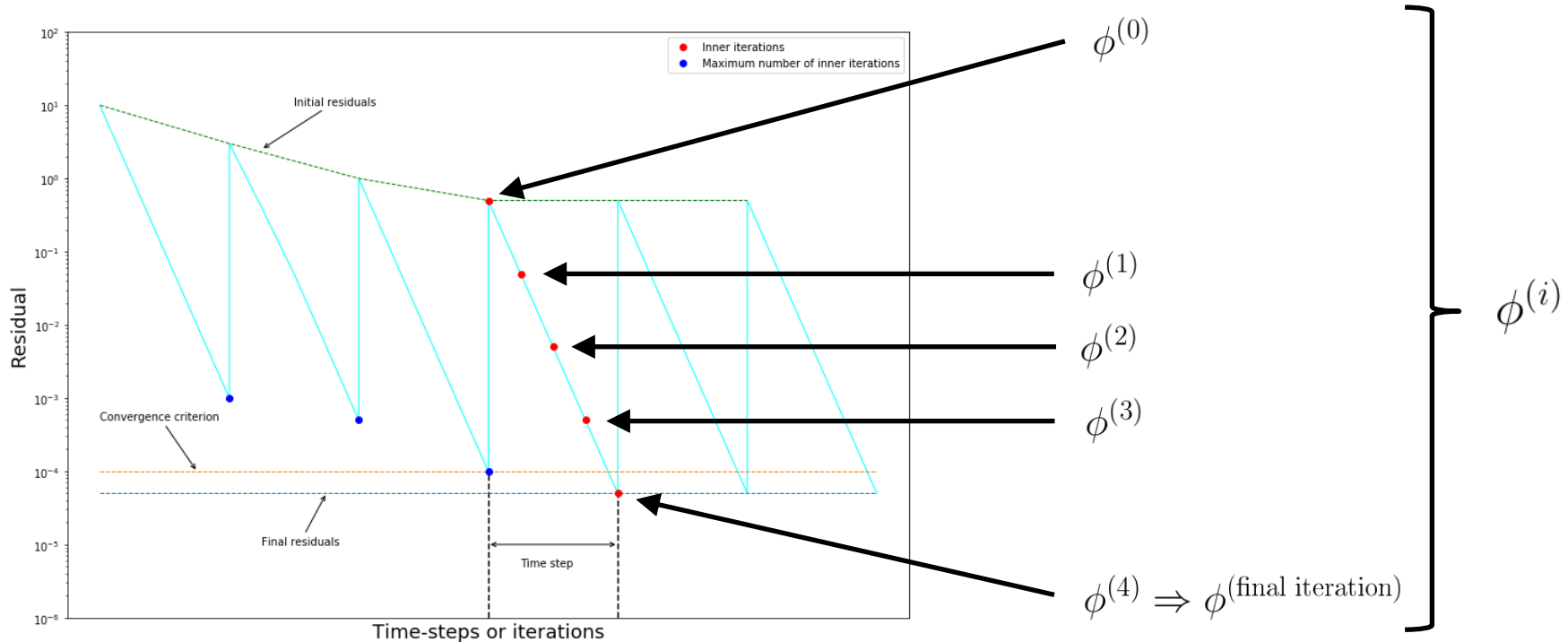
$$|\mathbf{A}\phi^i - \mathbf{b}| \leq |\mathbf{r}^i|$$

- The convergence rate of iterative solvers greatly depends on the matrix of coefficients  $\mathbf{A}$ .

# The Finite Volume Method: An overview

## Linear system solution

- To get a better idea of how iterative methods work, and what are initial residuals and final residuals, let us take another look at a residual plot.

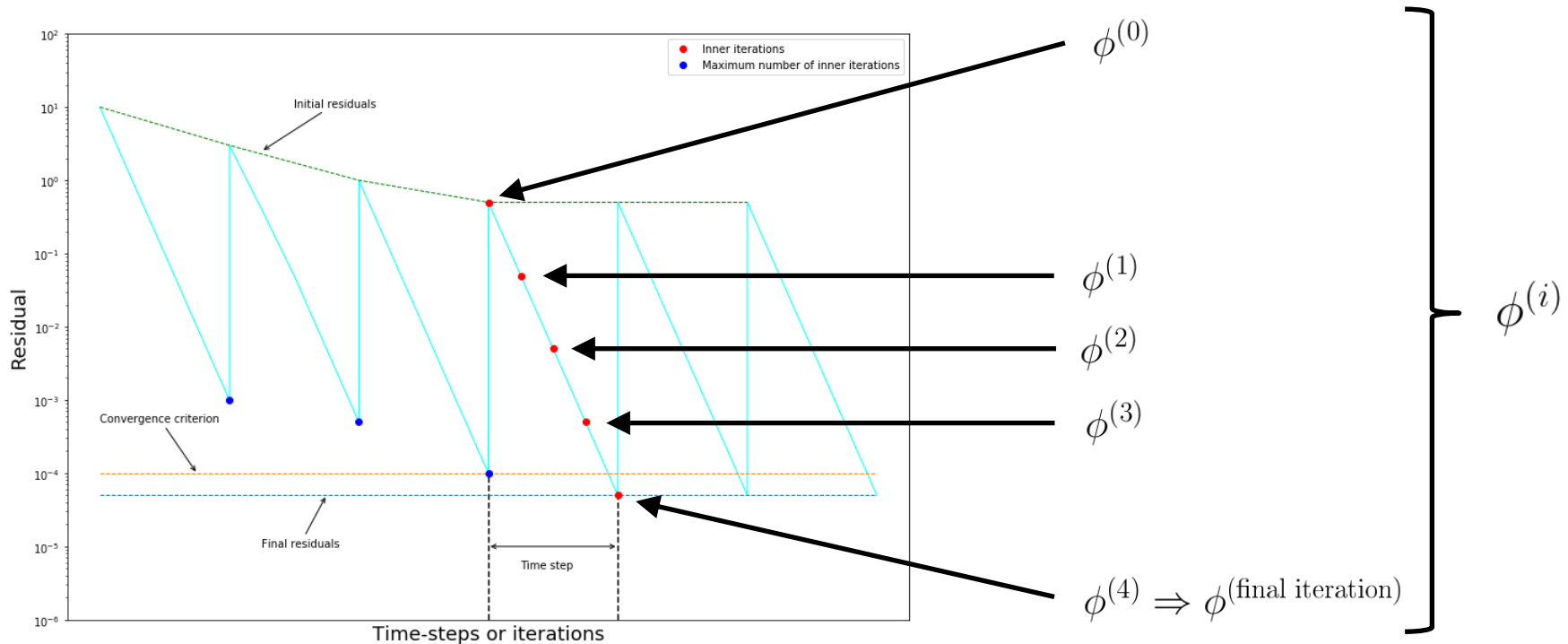


- $\phi^{(0)}$  is the initial guess used to start the iterative method.
- Iteration 0 defines the initial residual, and greatly influence the convergence rate.
- You can use any value at iteration 0, but usually is a good choice to take the previous solution vector.
- Remember, the closest you are to the actual solution, the faster the convergence rate will be. 120

# The Finite Volume Method: An overview

## Linear system solution

- To get a better idea of how iterative methods work, and what are initial residuals and final residuals, let us take another look at a residual plot.



- If the following condition is fulfilled, the linear solver will stop iterating and will advance to the next time-step.

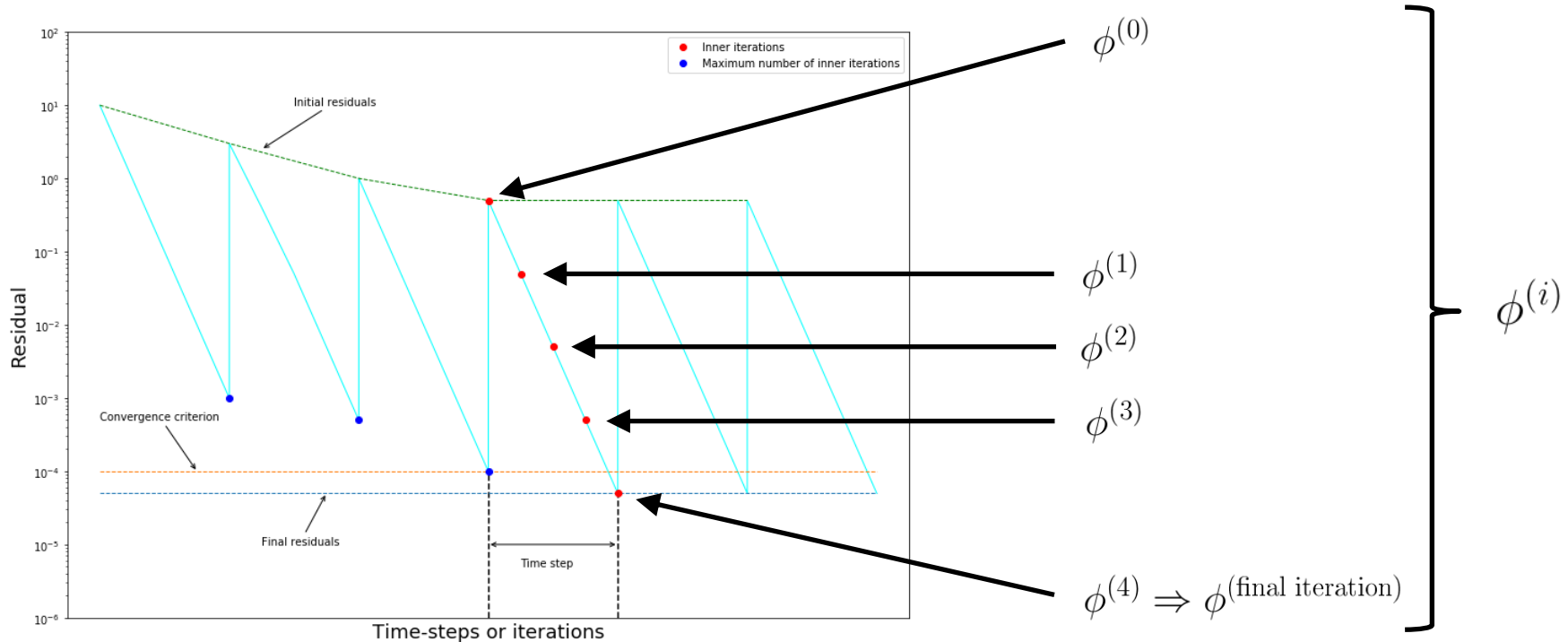
$$|\mathbf{A}\phi^i - \mathbf{b}| \leq |\mathbf{r}|$$

- This condition defines the final residual, where  $\mathbf{r}$  is the tolerance or convergence criterion (defined by the user).

# The Finite Volume Method: An overview

## Linear system solution

- To get a better idea of how iterative methods work, and what are initial residuals and final residuals, let us take another look at a residual plot.

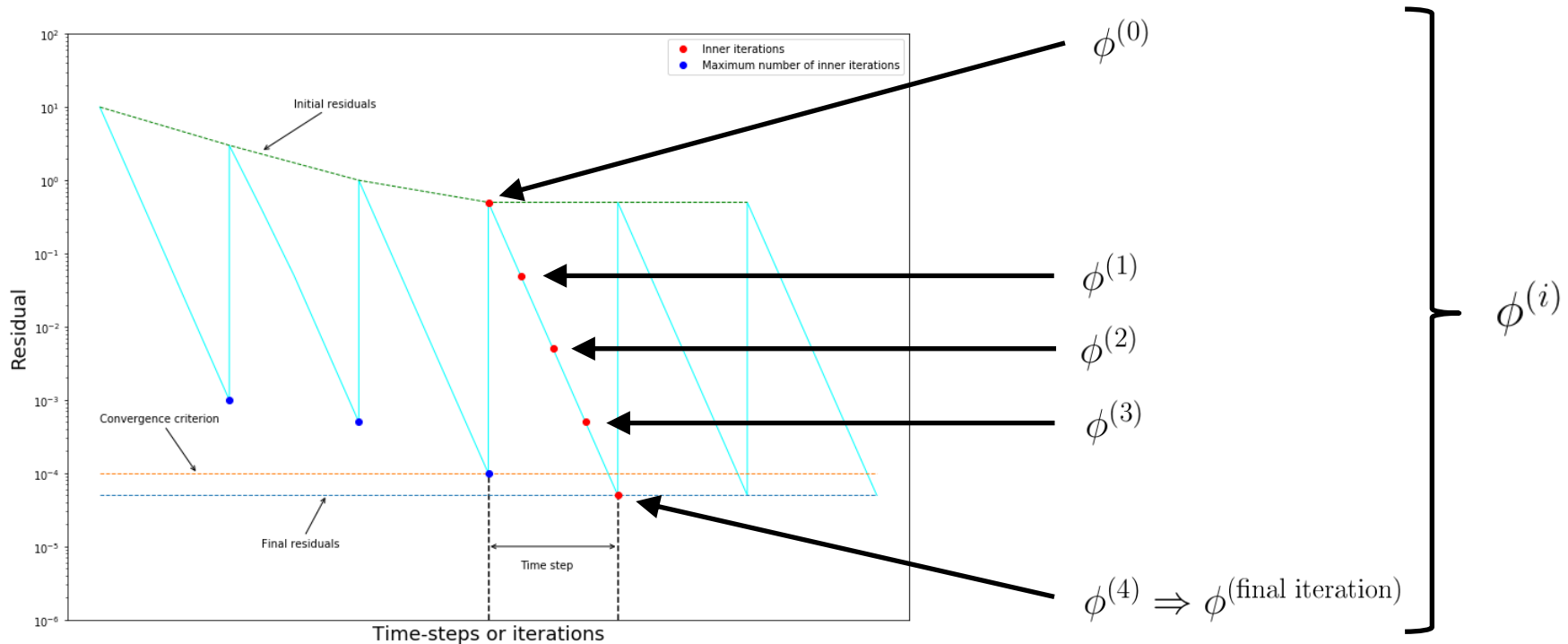


- By working in an iterative way, every single iteration  $\phi^{(i)}$  is a better approximation of the previous iteration  $\phi^{(i-1)}$
- Sometimes the linear solver might stop iterating because it has reached the maximum number of iterations, you should be careful of this because we are talking of unconverged iterations.
- Also, it is recommended to do at least one iteration as it helps at linearizing the equations.

# The Finite Volume Method: An overview

## Linear system solution

- To get a better idea of how iterative methods work, and what are initial residuals and final residuals, let us take another look at a residual plot.



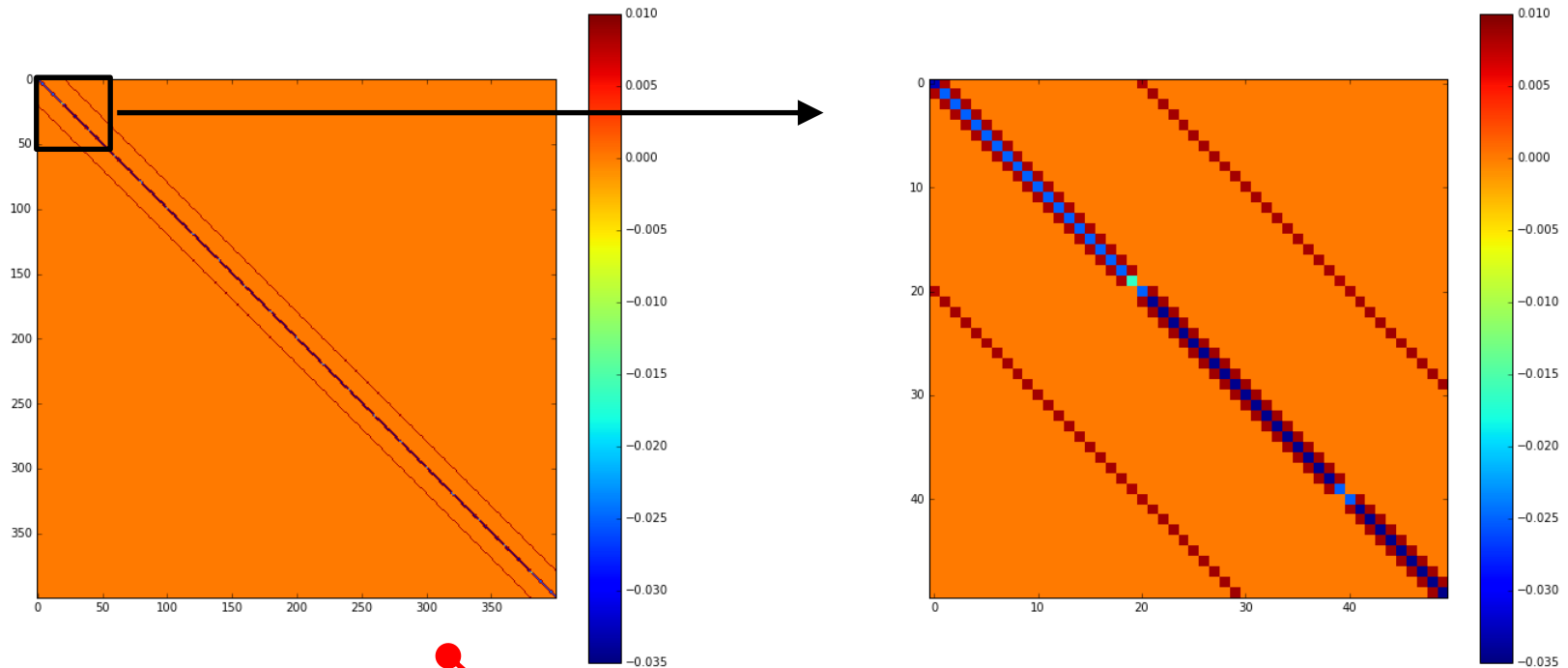
- It is clear that if the initial residual  $|\mathbf{A}\phi - \mathbf{b}|^0$  is the same as the final residual  $|\mathbf{A}\phi - \mathbf{b}|^{\text{Final iteration}}$  (we are converging in one iteration), we can say that we have reached a steady solution (this does not happen very often).
- Every iterative linear solver has different properties. Also, depending on the matrix type (symmetric or asymmetric), they might have different convergence rates.



# The Finite Volume Method: An overview

## Linear system solution

- Visualization of the pressure coefficient matrix **A** coming from a CFD simulation.
- Notice that in this case the matrix has a banded diagonal structure and is symmetric.
- In this case linear solvers perform extremely well.



Solution vector

$\text{Matrix of coefficients} \leftarrow \mathbf{A} \phi = \mathbf{b} \rightarrow \text{Boundary conditions and source terms}$

# The Finite Volume Method: An overview

## Multigrid and Newton-Krylov linear solvers – Some remarks

- The development of multigrid (**MG**) solvers (**GAMG** in OpenFOAM), together with the development of high-resolution TVD schemes and parallel computing, are among the most remarkable achievements of the history of CFD.
- Most of the time using **MG** linear solver is fine (for symmetric matrices).
- However, if you observe that the **MG** linear solver is taking too long to converge or is converging in more than 100 iterations, it is better to use a Newton-Krylov linear solver (e.g., preconditioned conjugate gradient or **PCG** in OpenFOAM).
- Particularly, we have found that the **GAMG** linear solver in OpenFOAM does not perform very well when you scale your computations to more than 1000 processors.
- Also, we have found that for some multiphase cases the **PCG** method outperforms the **GAMG**.
- But again, this is problem and hardware dependent.
- As you can see, you need to always monitor your simulations (stick to the screen for a while). Otherwise, you might end-up using a solver that is performing poorly (slow convergence rate), and this translate in increased computational time and costs.

# The Finite Volume Method: An overview

**So, what does an unstructured FVM solver do?**

# The Finite Volume Method: An overview

## So, what does an unstructured FVM solver do?

- It simply discretizes in space and time the governing equations in arbitrary polyhedral control volumes over the whole domain.
- Assembling in this way a large set of linear algebraic equations (LAE).
- It then solves this system of LAE to find the solution of the transported quantities.
- The FVM method basically converts the problem of calculus (surface and volume integrals) to that of linear algebra (solution of linear systems).

# The Finite Volume Method: An overview

## So, what does an unstructured FVM solver do?

- In the FVM method, the following information must be readily available to the solver:
  - The mesh.
  - Boundary conditions and initials conditions.
  - Physical properties such as density, gravity, diffusion coefficient, viscosity, etc.
  - Physical models, such as turbulence, mass transfer, etc.
  - How to discretize in space each term of the governing equations (diffusive, convective, gradient and source terms).
  - How to discretize in time the obtained semi-discrete governing equations.
  - How to solve the linear system of equations (crunching numbers).
  - Set runtime parameters and general instructions on how to run the case (such as time step, under-relaxation factors, and maximum CFL number).
  - Additionally, we may set monitors for post-processing.
- Every CFD solver will have a different way to ask for this information.
- Some of them use a GUI (e.g., Fluent, StarCCM+, CFX, NUMECA), and others interacts via ascii files using the command line interface (e.g., OpenFOAM).

# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- ~~2. Important concepts to remember~~
- ~~3. The Finite Volume Method: An overview~~
- 4. Navier-Stokes equations and pressure-velocity coupling**
5. On the CFL number
6. Unsteady and steady simulations
7. Understanding the residuals
8. Boundary conditions and initial conditions
9. The FVM in OpenFOAM: some implementation details and computational pointers
10. Best standard practices – General guidelines
11. Final remarks

# Navier-Stokes equations and pressure-velocity coupling

- To solve the Navier-Stokes equations we need to use a solution approach able to deal with the nonlinearities of the governing equations and with the coupled set of equations.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \mathbf{u}) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \mathbf{S}_u$$

$$\frac{\partial (\rho e_t)}{\partial t} + \nabla \cdot (\rho e_t \mathbf{u}) = -\nabla \cdot q - \nabla \cdot (p \mathbf{u}) + \boldsymbol{\tau} : \nabla \mathbf{u} + \mathbf{S}_{e_t}$$

+

Additional equations deriving from models, such as, volume fraction, chemical reactions, turbulence modeling, combustion, multi-species, thermodynamics, and so on.

# Navier-Stokes equations and pressure-velocity coupling

- Many numerical methods exist to solve the Navier-Stokes equations, just to name a few:
  - Pressure-correction methods (Predictor-Corrector type).
    - SIMPLE, SIMPLEC, SIMPLER, PISO.
  - Projection methods.
    - Fractional step (operator splitting), MAC, SOLA.
  - Density-based methods and preconditioned solvers.
    - Riemann solvers, ROE, HLLC, AUSM+, ENO, WENO.
  - Artificial compressibility methods.
  - Artificial viscosity methods.
  - Methods Based on Derived Variables
    - Stream Function-Vorticity
    - Vorticity-Velocity Method



# Navier-Stokes equations and pressure-velocity coupling

- We are going to briefly review the following two types of approaches for solving the NSE:
  - **Pressure-based approach (predictor-corrector).**
  - **Density-based approach.**
- Historically speaking, the **pressure-based approach** was developed for low-speed incompressible flows, while the **density-based approach** was mainly developed for high-speed compressible flows.
- However, both methods have been extended and reformulated to solve and operate for a wide range of flow conditions beyond their original intent.

# Navier-Stokes equations and pressure-velocity coupling

## Pressure-based approach

- Two pressure-based solution methods are generally available, namely:
  - Segregated method.
  - Coupled method.
- Pressure-based methods are intrinsically implicit.
- They are the default option in most of CFD solvers.
- In the pressure-based approach the velocity field is obtained from the momentum equations. The pressure is obtained by solving the pressure-Poisson equation. There is some mathematical manipulation involved.
- In the segregated algorithm, the individual governing equations for the primitive variables are solved one after another.
- The coupled approach solves the continuity, momentum, and energy equation simultaneously, that is, coupled together.
  - Conversely to the pressure-based approach, there is no mathematical manipulation of the governing equations.

# Navier-Stokes equations and pressure-velocity coupling

## Pressure-based approach – Pressure equation derivation

- The pressure equation is derived starting from the momentum equation,

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nu \nabla^2 \mathbf{U} = -\nabla p$$

- Then, by taking the divergence of the momentum equation and setting  $\nabla \cdot \mathbf{U} = 0$ , we obtain,

$$\cancel{\nabla \cdot \left( \frac{\partial \mathbf{U}}{\partial t} \right)} + \nabla \cdot (\nabla \cdot (\mathbf{U}\mathbf{U})) - \cancel{\nabla \cdot (\nu \nabla^2 \mathbf{U})} = -\nabla^2 p$$

- Then, the final form of the pressure equation is as follows,

$$\nabla^2 p = \nabla \cdot \mathbf{H}(\mathbf{U}) \quad \text{where} \quad \mathbf{H}(\mathbf{U}) = -\nabla \cdot (\mathbf{U}\mathbf{U})$$

- Notice that the continuity is enforced when deriving the pressure equation and in all boundaries of the domain.

# Navier-Stokes equations and pressure-velocity coupling

## Pressure-based approach – Pressure equation derivation

- In the pressure-based approach, the actual equations that are being solved are,

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nu \nabla^2 \mathbf{U} = -\nabla p$$

This system of equations is equivalent to the original Navier-Stokes equations.

$$\nabla^2 p = \nabla \cdot \mathbf{H}(\mathbf{U}) \quad \text{where} \quad \mathbf{H}(\mathbf{U}) = -\nabla \cdot (\mathbf{U}\mathbf{U})$$

- The previous equations are solved in a given domain, with boundary conditions **BCs**, and initial condition **ICs**.
- In this set of equations, continuity  $\nabla \cdot \mathbf{U} = 0$  is enforced while deriving the pressure equation (referred to as pressure-Poisson equation) and in all boundaries of the domain.
- We use these equations because in the original incompressible Navier-Stokes equations, pressure does not appear in the continuity equation, so is not possible to link the equations.
- Therefore, we derive an alternative set of equations where pressure appears (albeit in the form of a gradient and large pressure gradients may cause stability and accuracy problems).
- So now we can use the velocity obtained in the momentum equation (momentum predictor step) to compute the pressure using the pressure-Poisson equation (pressure corrector step), and then correct the velocity with the new pressure value (momentum corrector step).
- This is referred to as pressure-velocity coupling (P-V coupling).

# Navier-Stokes equations and pressure-velocity coupling

## Pressure-based approach – Segregated method

- In the pressure-based approach the velocity field is obtained from the momentum equations.
- In the pressure-based approach the pressure field is extracted by solving a pressure or pressure correction equation which is obtained by manipulating continuity and momentum equations.
- If it is required, the energy equation is solved.
- Then, equations for other scalars such as turbulence, volume fraction, chemical species, etc., are solved.
- The solution process keeps iterating over the entire set of governing equations until the solution converges to a given criterion or the user decides to stop the simulation.
- In this approach, each governing equation while being solved, is *decoupled* or *segregated* from other equations, hence its name.
- The segregated algorithm is memory-efficient, since the discretized equations need only be stored in the memory one at a time.
- However, the solution convergence is relatively slow (in comparison to coupled solvers) as the equations are solved one at a time.

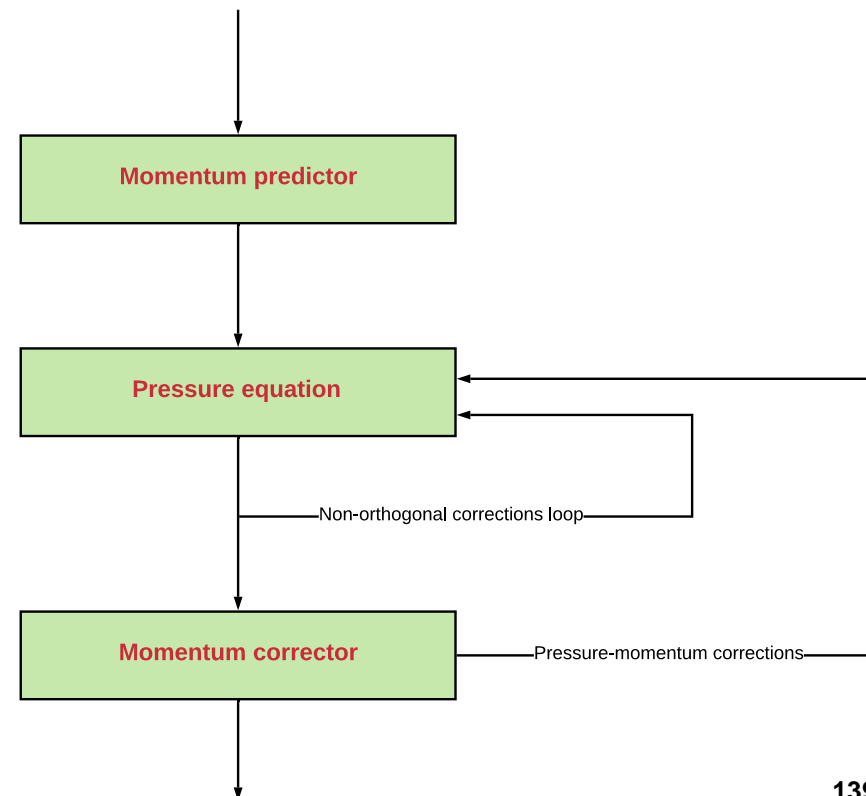
# Navier-Stokes equations and pressure-velocity coupling

## Pressure-based approach – Segregated method

- As we have seen, mesh non-orthogonality introduces secondary gradients into the pressure equation (the term  $f(\nabla p)$  in the equation below).

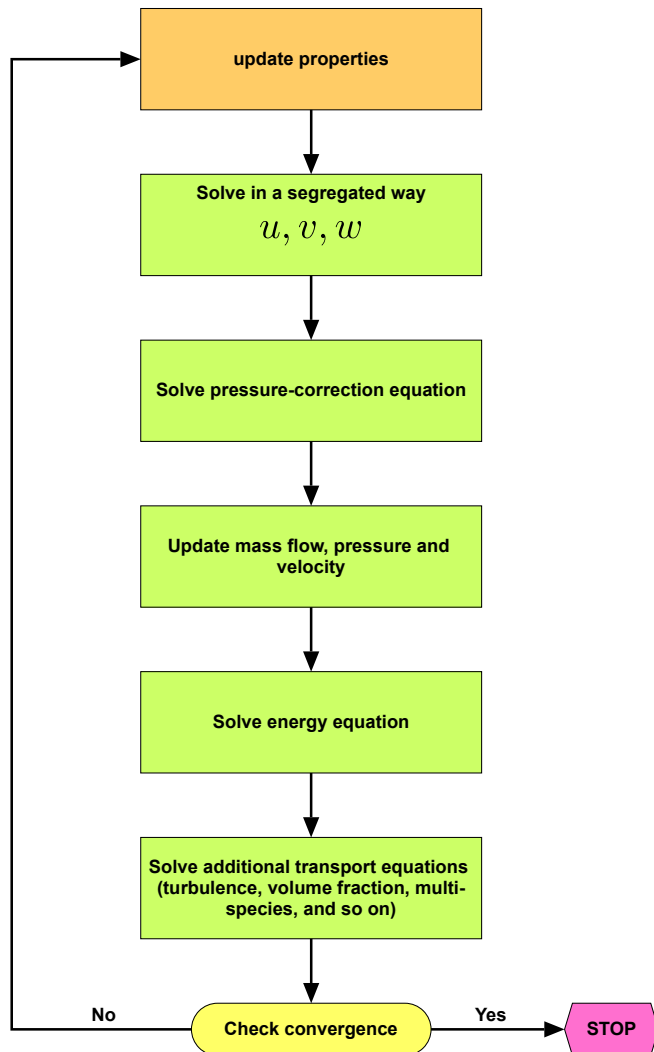
$$\nabla^2 p = \nabla \cdot \mathbf{H}(\mathbf{U}) + f(\nabla p) \quad \text{where} \quad \mathbf{H}(\mathbf{U}) = -\nabla \cdot (\mathbf{U}\mathbf{U})$$

- To reduce any error introduced by secondary gradients, we need to correct the pressure equation for non-orthogonality.
- That is, we solve for pressure and then we correct it, obtaining in this way better approximations.
- After correcting momentum with the previous pressure value, we can substitute the new value in the pressure equation and solve again (additional passes through pressure and momentum corrector equations).
- By looping in this way we gain more stability and accuracy by getting better approximations.



# Navier-Stokes equations and pressure-velocity coupling

## Pressure-based approach – Segregated method



Each iteration consists of the steps illustrated in the figure and outlined below:

1. Update fluid properties (for example, density, viscosity, specific heat, turbulent viscosity, and so on) based on the initial conditions or current solution.
2. Solve the momentum equations, one after another, using the recently updated values of pressure and face mass fluxes.
3. Solve the pressure correction equation using the recently obtained velocity field and the mass-flux.
4. Correct face mass fluxes, pressure, and the velocity field using the pressure correction obtained from Step 3.
5. Solve the energy equation using the current values of the solution variables.
6. Solve additional transport equations, such as turbulent quantities, volume fraction, species, and so on), using the current values of the solution variables.
7. Check for the convergence of the equations.

These steps are continued until the convergence criterion is met or the user decides to stop the simulation.

# Navier-Stokes equations and pressure-velocity coupling

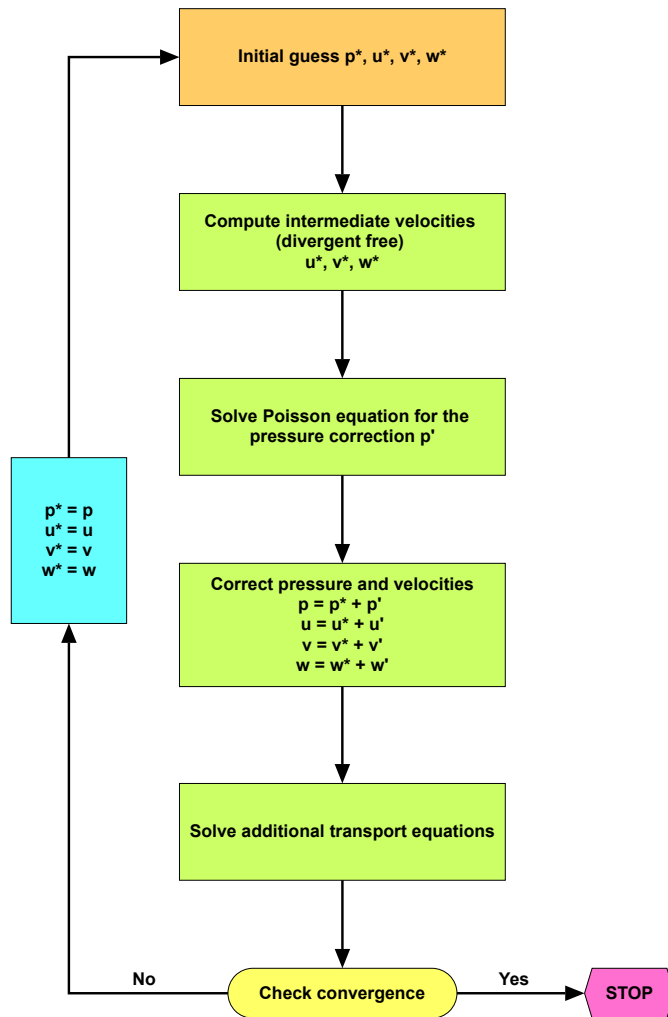
## Pressure-based approach – Segregated method

- To achieve pressure-velocity coupling using segregated solvers there are several methods available. To name a few:
  - **SIMPLE** (Semi-Implicit Method for Pressure-Linked Equations)
  - **SIMPLEC** (SIMPLE Corrected/Consistent)
  - **SIMPLER** (SIMPLE Revised)
  - **PISO** (Pressure Implicit with Splitting Operators)
- All the aforementioned methods are based on the predictor-corrector approach.
- Each one have different properties, options and loop in slightly different ways.
- But at the end of the day, all of them will iterative until reaching the convergence criterion.

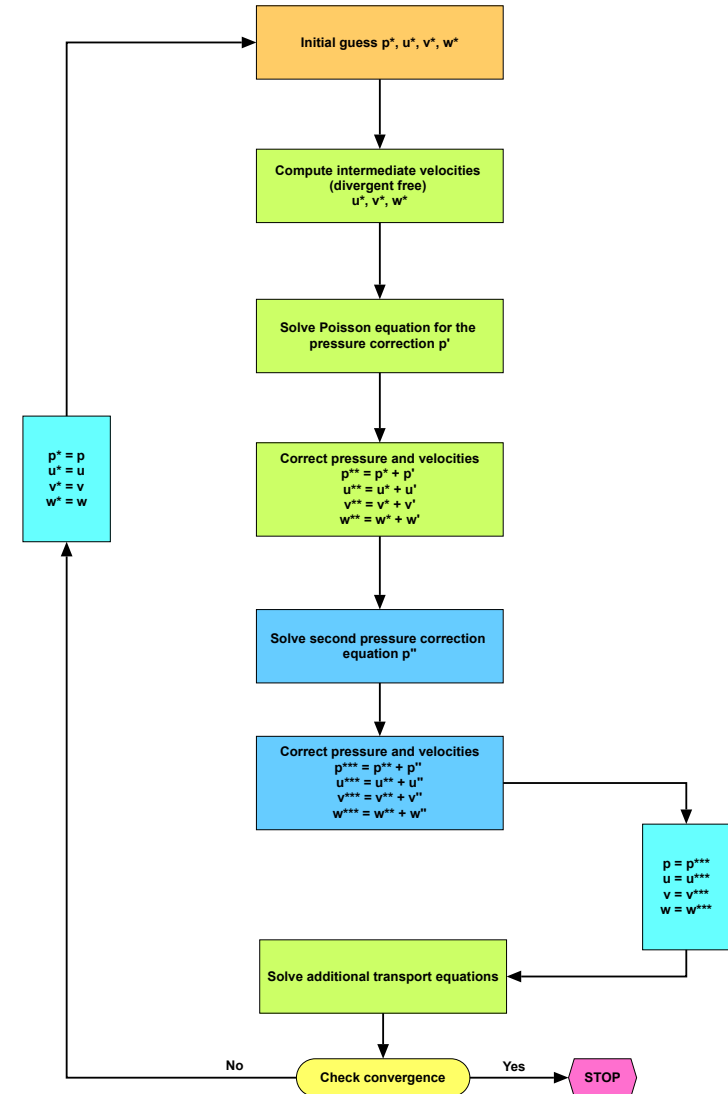


# Navier-Stokes equations and pressure-velocity coupling

## Pressure-velocity coupling using the SIMPLE method



## Pressure-velocity coupling using the PISO method



# Navier-Stokes equations and pressure-velocity coupling

## A few remarks about the SIMPLE and PISO methods

- The **SIMPLE** method [1] was initially developed in the 1970s for steady state flows and extended later to unsteady flows with iterative marching at each time-step.
- In the **SIMPLE** method, under-relaxation must be used for stability reasons. This means that many iterations are required.
- Also, in the **SIMPLE** method the under-relaxation factors (URF) need to be tuned according to the application.
- For time dependent flows, iterative marching is necessary at each time-step, making the **SIMPLE** method inefficient.
- The **PISO** method [2] was developed in the early 1980s for steady and unsteady flows, and to address some of the drawbacks of the **SIMPLE** method.
- The **PISO** method is very efficient for unsteady flows since there is no need for iterative marching at each time-step.
- In the **PISO** method, the pressure field is free from continuity errors after the second corrector. Therefore, two correctors are normally sufficient.

### References:

- [1] S. V. Patankar. D. B. Spalding, "A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows", Int. J. Heat Mass Transfer, 15, 1787-1806 (1972).
- [2] R. I. Issa, "Solution of the implicitly discretized fluid flow equations by operator-splitting", J. Comput. Phys., 62, 40-65 (1985).

# Navier-Stokes equations and pressure-velocity coupling

## A few remarks about the SIMPLE and PISO methods

- The **PISO** method often needs more than two correctors when large pressure gradients exist.
- The **PISO** method is stable for CFL numbers much greater than 1, and it does not need adjustable under-relaxation parameters.
- The **SIMPLE** and **PISO** methods are both implicit.
- Both methods were initially developed for staggered grids, but now they can be used with collocated meshes (standard practice in most modern CFD solvers).
- In the **SIMPLE** and **PISO** methods, to avoid the pressure-velocity decoupling that occurs when using collocated meshes, the cell-face velocity value is computed using Rhie-Chow interpolation [1].
- For unsteady problems, the **PISO** method is much faster than the **SIMPLE** method.
- For steady problems, both methods shows similar convergence rates. However, the **SIMPLE** is less computational expensive.
- The drawback of the **SIMPLE** method for steady simulations is that the URF are problem dependent.

### References:

[1] C. M. Rhie, W. L. Chow, "Numerical study of the turbulent flow past an airfoil with trailing edge separation", AIAA Journal, Vol. 21, 1525-1532 (1983).

# Navier-Stokes equations and pressure-velocity coupling

## On the origins of the SIMPLE and PISO methods

- **SIMPLE**
  - S. V. Patankar and D. B. Spalding, “A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows”, Int. J. Heat Mass Transfer, 15, 1787-1806 (1972).
- **SIMPLE-C**
  - J. P. Van Doormaal and G. D. Raithby, “Enhancements of the SIMPLE method for predicting incompressible fluid flows”, Numer. Heat Transfer, 7, 147-163 (1984).
- **SIMPLE-R**
  - S.V. Patankar, “A calculation procedure for two dimensional elliptic situations”, Numerical Heat Transfer, Vol. 14, pp. 409-425 (1984).
- **PISO**
  - R. I. Issa, “Solution of the implicitly discretized fluid flow equations by operator-splitting”, J. Comput. Phys., 62, 40-65 (1985).
- **Rhie-Chow interpolation**
  - C. M. Rhie and W. L. Chow, “Numerical study of the turbulent flow past an airfoil with trailing edge separation”, AIAA Journal, Vol. 21, 1525-1532 (1983).

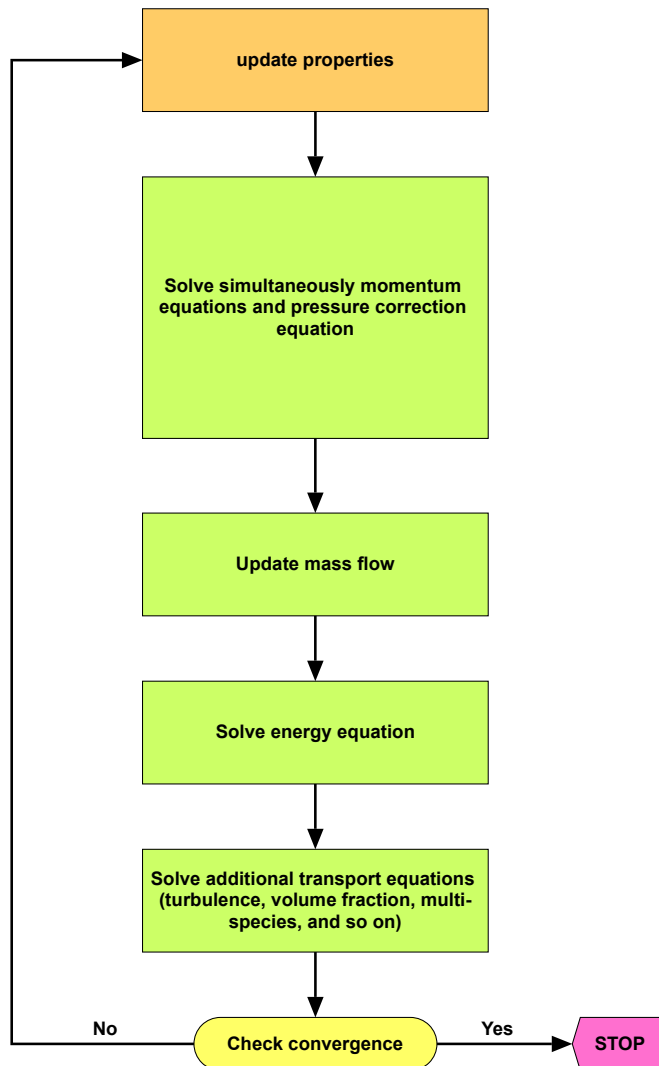
# Navier-Stokes equations and pressure-velocity coupling

## Pressure-based approach – Coupled method

- The pressure-based coupled algorithm solves a coupled system of equations comprising the momentum equations and the pressure-based continuity equation.
- The remaining equations are solved in a decoupled fashion as in the segregated algorithm.
- Since the momentum and continuity equations are solved in a closely coupled manner, the rate of convergence significantly improves when compared to the segregated algorithm.
- However, compared to the segregated algorithm the memory requirements are larger by at least 2 to 3 times.
- As memory requirements are very high, they are not very efficient for unsteady computations.

# Navier-Stokes equations and pressure-velocity coupling

## Pressure-based approach – Coupled method



Each iteration consists of the steps illustrated in the figure and outlined below:

1. Update fluid properties (for example, density, viscosity, specific heat) including turbulent viscosity based on the initial conditions or current solution.
2. Solve the momentum equations and pressure-correction equation in a coupled manner.
3. Correct face mass fluxes, pressure, and the velocity field using the pressure correction obtained from Step 2.
4. Solve the energy equation using the current values of the solution variables.
5. Solve additional transport equations, such as turbulent quantities, volume fraction, species, and so on), using the current values of the solution variables.
6. Check for the convergence of the equations.

These steps are continued until the convergence criterion is met or the user decides to stop the simulation.

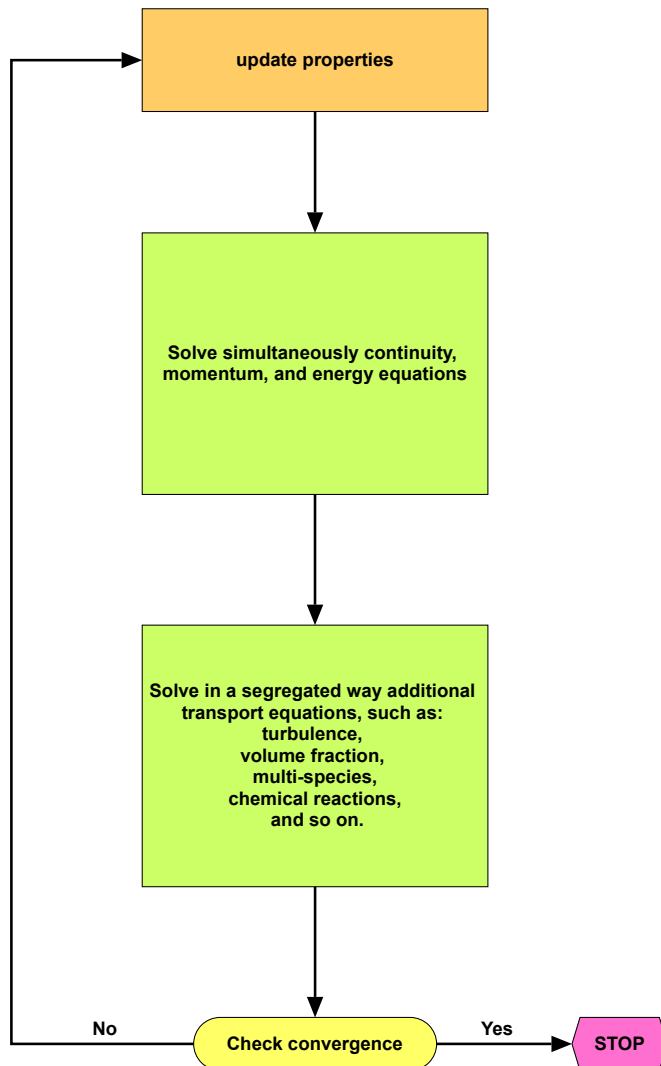
# Navier-Stokes equations and pressure-velocity coupling

## Density-based approach

- The density-based approach solves the continuity, momentum, and energy equation simultaneously, that is, coupled together.
- Conversely to the pressure-based approach, there is no mathematical manipulation on the governing equations.
- Pressure is obtained through an equation of state.
- Governing equations for additional scalars are solved afterward and sequentially, that is, segregated from one another and from the coupled set.
- The density-based solvers are recommended when there is a strong coupling between density, energy, momentum, and/or species.
- Because the governing equations are non-linear and coupled, several iterations of the solution loop must be performed before a converged solution is obtained.
- In the density-based solution method, you can solve the coupled system of equations using either an explicit formulation or an implicit formulation.
- Typical solution methods used in the density-based approach are:
  - ROE, AUSM+, HLLC

# Navier-Stokes equations and pressure-velocity coupling

## Density-based approach



Each iteration consists of the steps illustrated in the figure and outlined below:

1. Update fluid properties (for example, density, viscosity, specific heat) including turbulent viscosity based on the initial conditions or current solution.
2. Solve the continuity, momentum and energy equations in a coupled manner.
3. Solve additional transport equations, such as turbulent quantities, volume fraction, species, and so on), using the current values of the solution variables.
4. Check for the convergence of the equations.

These steps are continued until the convergence criterion is met or the user decides to stop the simulation.



# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- ~~2. Important concepts to remember~~
- ~~3. The Finite Volume Method: An overview~~
- ~~4. Navier-Stokes equations and pressure-velocity coupling~~
- 5. On the CFL number**
6. Unsteady and steady simulations
7. Understanding the residuals
8. Boundary conditions and initial conditions
9. The FVM in OpenFOAM: some implementation details and computational pointers
10. Best standard practices – General guidelines
11. Final remarks

# On the CFL number

- First of all, what is the CFL or Courant number?
- In one dimension, the CFL number is defined as,

$$CFL = \frac{u \Delta t}{\Delta x}$$

- The CFL number is a measure of how much information ( $u$ ) traverses a computational grid cell ( $\Delta x$ ) in a given time-step ( $\Delta t$ ).
- The CFL number is a necessary condition to guarantee the stability of the numerical scheme.
- But not all numerical schemes have the same stability requirements.
- By doing a linear stability study, we can find the stability requirements of each numerical scheme (but this is out of the scope of this lecture).

# On the CFL number

- Let us talk about the **CFL number condition** (which is related to the CFL number).
- The **CFL number condition** is the maximum allowable CFL number a numerical scheme can use.
- For the **N** dimensional case, the CFL number condition becomes,

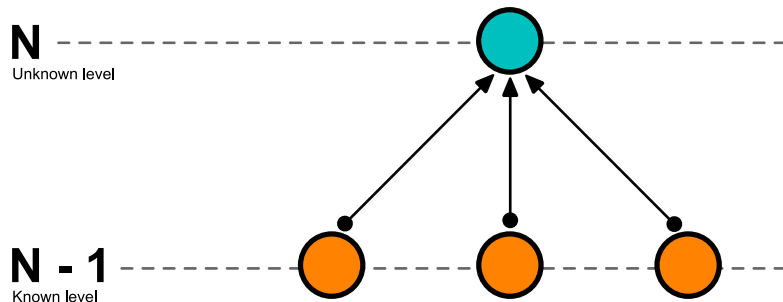
$$CFL = \Delta t \sum_{i=1}^n \frac{u_i}{\Delta x_i} \leq CFL_{max}$$

- To get a better idea of the importance of the CFL number condition, let us talk about explicit and implicit methods.
- Explicit and implicit methods are approaches used for obtaining the approximate numerical solution of time-dependent ODEs and PDEs.
- Explicit and implicit methods have different stability requirements.
- Also, the implementation details are different.

# On the CFL number

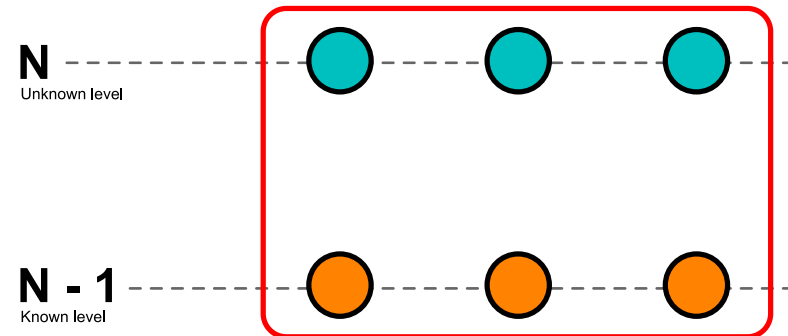
## Explicit methods

- For a given variable, the unknown value in each cell is computed using a relation that includes only existing values.
- Therefore, each unknown will appear in only one equation in the system and the equations for the unknown value in each cell can be solved one at a time to give the unknown quantities.
- In the figure, **N** is the current time level. We do not know the solution in this level.
- **N - 1** is the previous time level, where we know the solution in all control volumes.



## Implicit methods

- For a given variable, the unknown value in each cell is computed using a relation that includes both existing and unknown values from neighboring cells.
- Therefore, each unknown will appear in more than one equation in the system, and these equations must be solved simultaneously to give the unknown quantities.
- In the figure, **N** is the current time level. We do not know the solution in this level.
- **N - 1** is the previous time level, where we know the solution in all control volumes.



# On the CFL number

## Explicit methods

- Explicit methods are **conditionally stable**.
- They have a constraint on the maximum allowable CFL number (CFL number condition).
- If you choose a CFL number larger than the maximum allowable by the explicit method, your numerical solution will become unstable, and it will diverge.
- Usually, the maximum allowable **CFL** number is limited to 1.0.
- Some explicit methods have a **CFL condition** of 0.5, and some of them can go up to 2.0
- In OpenFOAM you will find explicit solvers (last time we checked there was only one solver).

## Implicit methods

- Implicit numerical methods are **unconditionally stable**.
- In other words, they are not constrained to the **CFL number condition**.
- However, the fact that you are using a numerical method that is unconditionally stable, does not mean that you can choose a time step of any size.
- The time-step must be chosen in such a way that it resolves the time-dependent features, and it maintains the solver stability.
- When we use implicit methods, we need to assemble a large system of equations.
- The memory requirements of implicit methods are much higher than those of explicit methods.
- In OpenFOAM, most of the solvers are implicit.

# On the CFL number

## Some facts of explicit and implicit methods

- Explicit methods are extremely accurate, but they have terrible time steps constraints.
- For the same CFL number, the time-step of explicit methods is usually an order of magnitude lower than the time-step required for implicit methods.
- This means that they are approximately ten times slower than implicit methods.
- The memory requirements of explicit are really low and they are extremely easy to parallelize.
- Explicit methods perform really well in GPUs.
- Also, explicit methods are extremely fast (clock time per iteration), and easy to implement
- In order to arrive to a converged solution, you will need to perform a lot of iterations. This is mainly related to the time step constraint.
- If you are interested in using large time steps (large CFL number) you will need to use implicit methods.

# On the CFL number

## Some facts of explicit and implicit methods

- Due to the fact that implicit methods let you use large time steps; you can arrive to a converge solution much faster than with explicit methods.
- Also, implicit methods tend to be more stable than explicit methods.
- It is highly advisable that you choose a time step in such a way that it resolves the time scales.
- If you use large time steps with implicit methods, it is likely that you will need to increase the cell count in order to maintain the accuracy of the solution, and this translates in an increased computational cost.
- In our personal experience, we have been able to go up to a  $CFL = 5.0$  while maintaining the accuracy and without increasing too much the computational cost.
- But as we are often interested in the unsteadiness of the solution, we usually use a CFL number in the order of 1.0

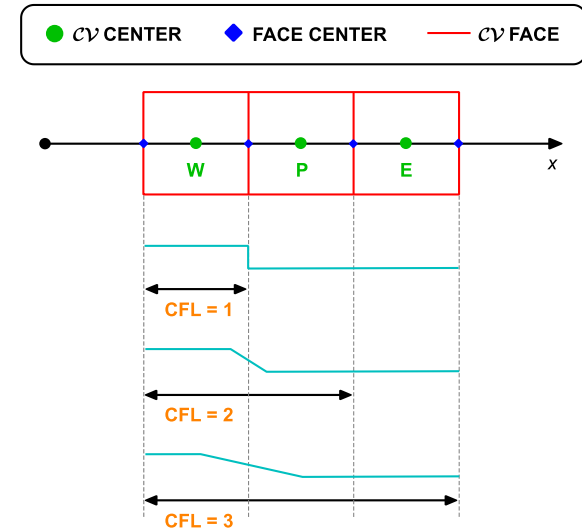
# On the CFL number

## The CFL number for dummies

- I like to see the CFL number as follows,

$$CFL = \frac{u \Delta t}{\Delta x} = \frac{u}{\Delta x / \Delta t} = \frac{\text{speed of the PDE}}{\text{speed of the mesh}}$$

- It is an indication of the amount of information that propagates through one cell (or many cells), in one time-step.



- By the way, and this is extremely important, the CFL condition is a necessary condition for stability (and hence convergence).
- But it is not always sufficient to guarantee stability.
- Other properties of the discretization schemes that you should observe are: conservation, boundedness, transportiveness, and accuracy.
- The CFL number is not a magical number!



# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- ~~2. Important concepts to remember~~
- ~~3. The Finite Volume Method: An overview~~
- ~~4. Navier-Stokes equations and pressure-velocity coupling~~
- ~~5. On the CFL number~~
- 6. Unsteady and steady simulations**
- ~~7. Understanding the residuals~~
- ~~8. Boundary conditions and initial conditions~~
- ~~9. The FVM in OpenFOAM: some implementation details and computational pointers~~
- ~~10. Best standard practices – General guidelines~~
- ~~11. Final remarks~~

# Unsteady and steady simulations

- Nearly all flows in nature and industrial applications are unsteady (also known as transient or time-dependent).
- If you are dealing with turbulence (almost every scenario), you need to take into account the unsteadiness inherent of turbulent flows.
- And if you are dealing with multiphase flows, you need to take into account the multiscale nature of such flows, which makes multiphase flows intrinsically unsteady.
- Unsteadiness is due to:
  - Instabilities.
  - Non-equilibrium initial conditions.
  - Time-dependent boundary conditions.
  - Source terms.
  - Chemical reactions.
  - Moving or deforming bodies.
  - Turbulence.
  - Buoyancy.
  - Convection.
  - Multiple phases

# Unsteady and steady simulations

- A few examples of unsteady applications:
  - Internal and external aerodynamics.
  - Shock wake interaction.
  - Hydrodynamics, sea keeping, free surface, waves.
  - Multiphase flows.
  - Turbomachinery.
  - Moving and deforming bodies.
  - Fluid-structure interaction.
  - Vortex-induced vibrations.
  - Unsteady heat transfer.
  - HVAC.
  - Aero-vibro-acoustics.
  - And many more...

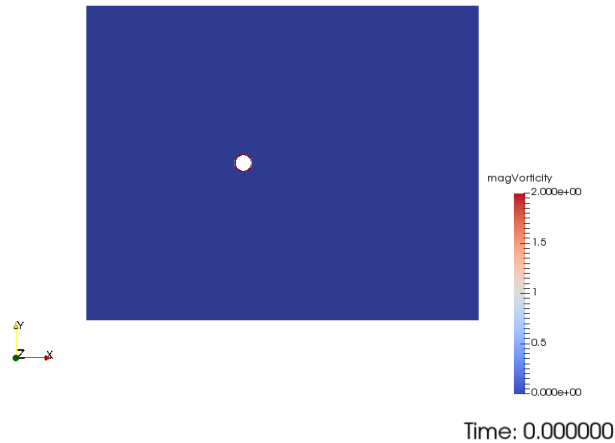
# Unsteady and steady simulations

- A few examples of unsteady applications:

## Vortex shedding

[www.wolfdynamics.com/wiki/FVM\\_uns/ani1.gif](http://www.wolfdynamics.com/wiki/FVM_uns/ani1.gif)

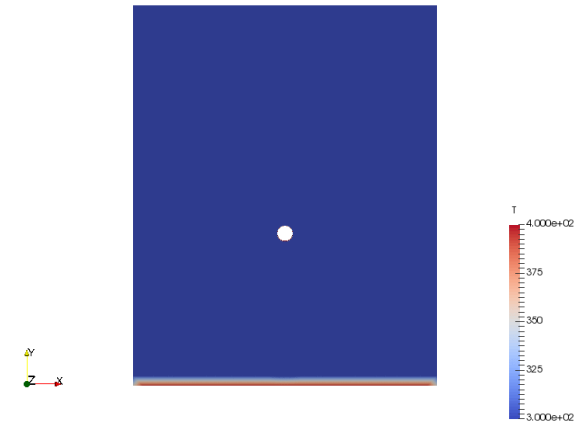
Time: 0.000000



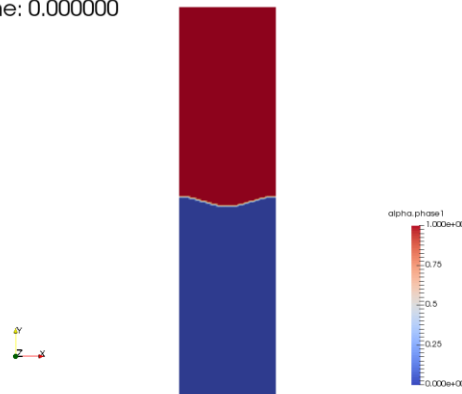
## Buoyant flow

[www.wolfdynamics.com/wiki/FVM\\_uns/ani2.gif](http://www.wolfdynamics.com/wiki/FVM_uns/ani2.gif)

Time: 0.000000



Time: 0.000000



## Multiphase flow

[www.wolfdynamics.com/wiki/FVM\\_uns/ani3.gif](http://www.wolfdynamics.com/wiki/FVM_uns/ani3.gif)

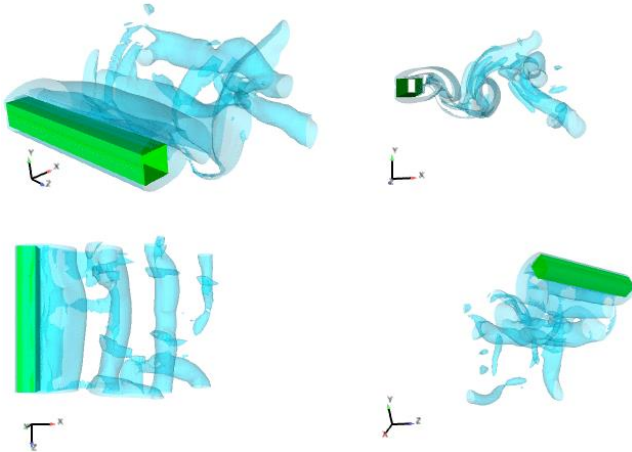


# Unsteady and steady simulations

- A few examples of unsteady applications:

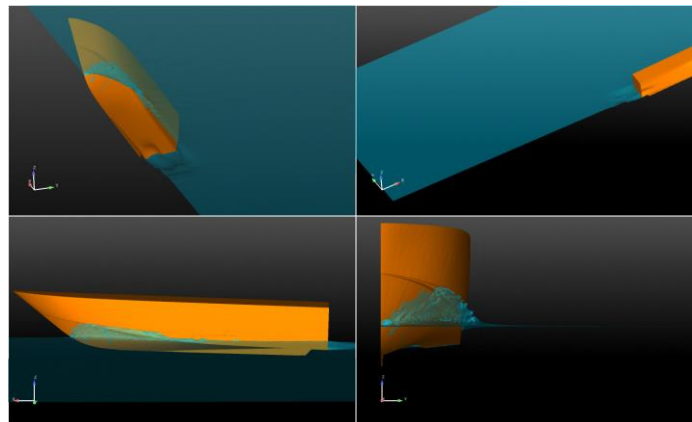
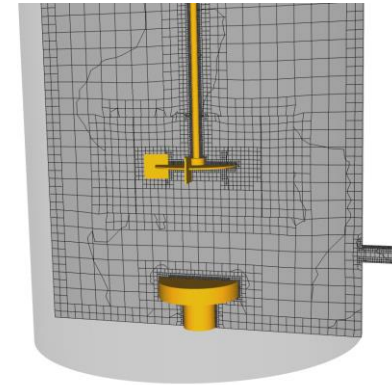
## Turbulent flows - SRS

[www.wolfdynamics.com/wiki/FVM\\_uns/ani4.gif](http://www.wolfdynamics.com/wiki/FVM_uns/ani4.gif)



## Sliding grids – Continuous stirred tank reactor

[www.wolfdynamics.com/wiki/FVM\\_uns/ani5.gif](http://www.wolfdynamics.com/wiki/FVM_uns/ani5.gif)



## Marine applications - Sea keeping

[www.wolfdynamics.com/wiki/FVM\\_uns/ani6.gif](http://www.wolfdynamics.com/wiki/FVM_uns/ani6.gif)



# Unsteady and steady simulations

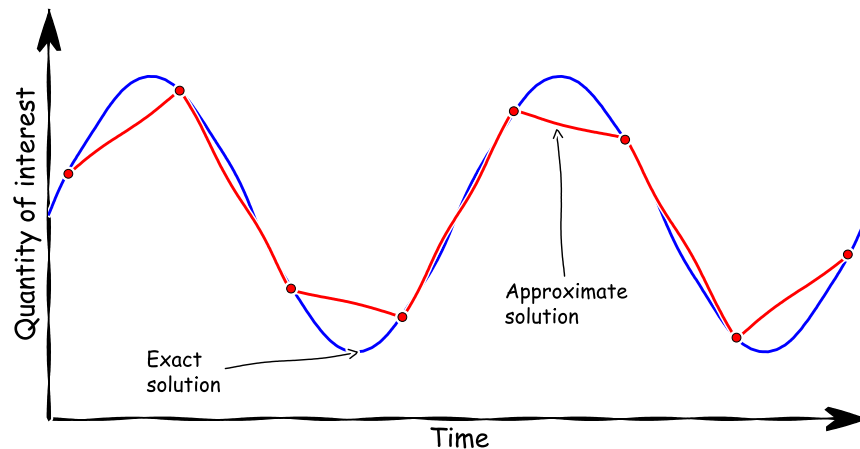
## How to run unsteady simulations using a general CFD solver?

- Select the time discretization scheme.
- Set the time step.
- Remember, the time-step must be chosen in such a way that it resolves the time-dependent features and maintains solver stability.
- Set the tolerance (absolute and/or relative) of the linear solvers.
- If it applies, monitor the CFL number.
- Monitor the stability and boundedness of the solution.
- Monitor a quantity of interest.
- And of course, you need to save the solution with a given frequency.
- Have in mind that unsteady simulations generate a lot of data.
- End time of the simulation?, it is up to you.
- Reducing the time-step will make the coefficient matrix more diagonally dominant.

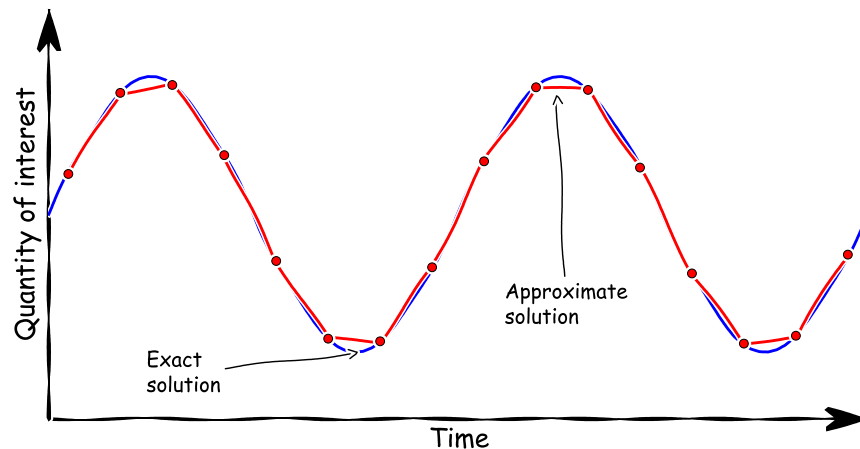
# Unsteady and steady simulations

## How to choose the time-step in unsteady simulations and monitor the solution

- Remember, when running unsteady simulations, the time-step must be chosen in such a way that it resolves the time-dependent features and maintains solver stability.



When you use large time steps you do not resolve well the physics

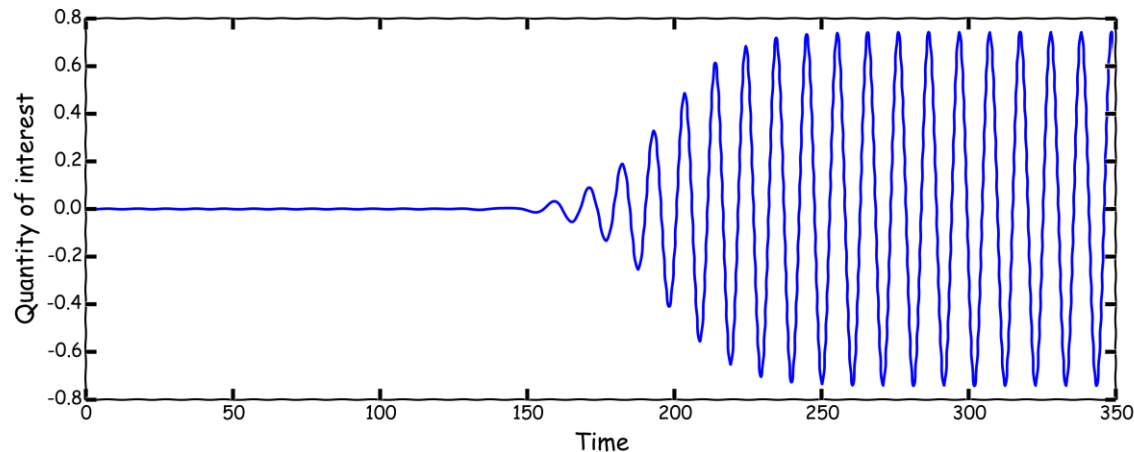
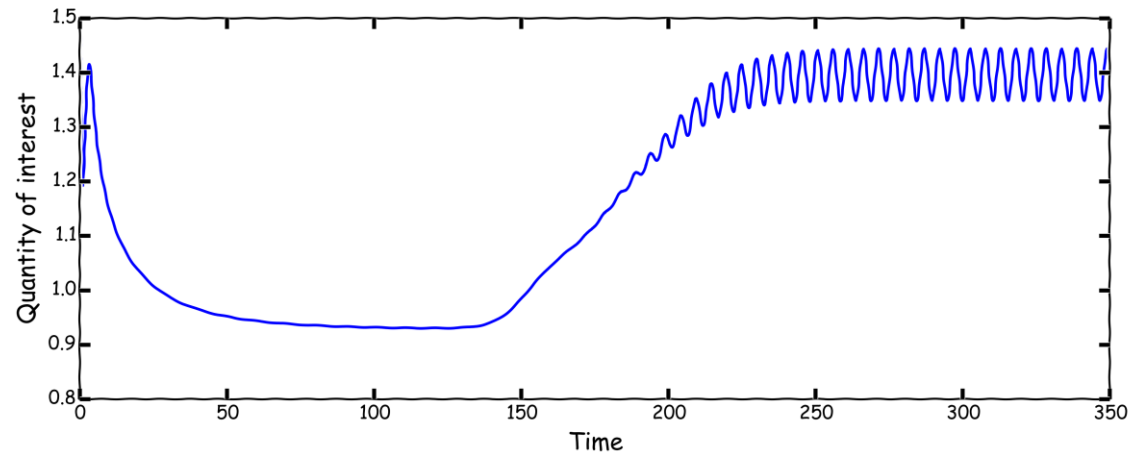


By using a smaller time step you resolve better the physics and you gain stability

# Unsteady and steady simulations

## Monitoring and sampling unsteady simulations

- When running unsteady simulations, it is highly advisable to monitor a quantity of interest.
- The quantity of interest can fluctuate in time, this is an indication of unsteadiness.

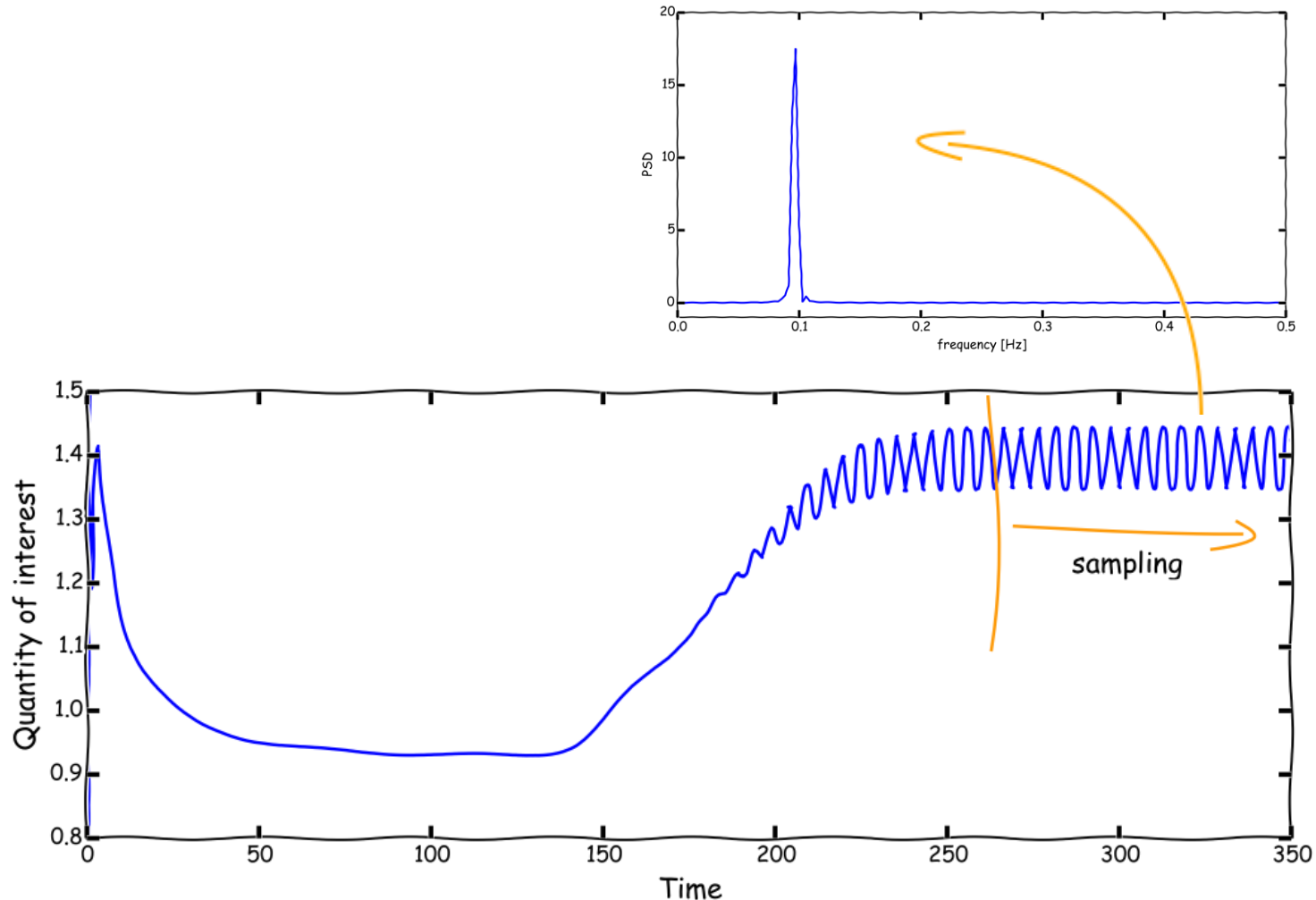




# Unsteady and steady simulations

## Monitoring and sampling unsteady simulations

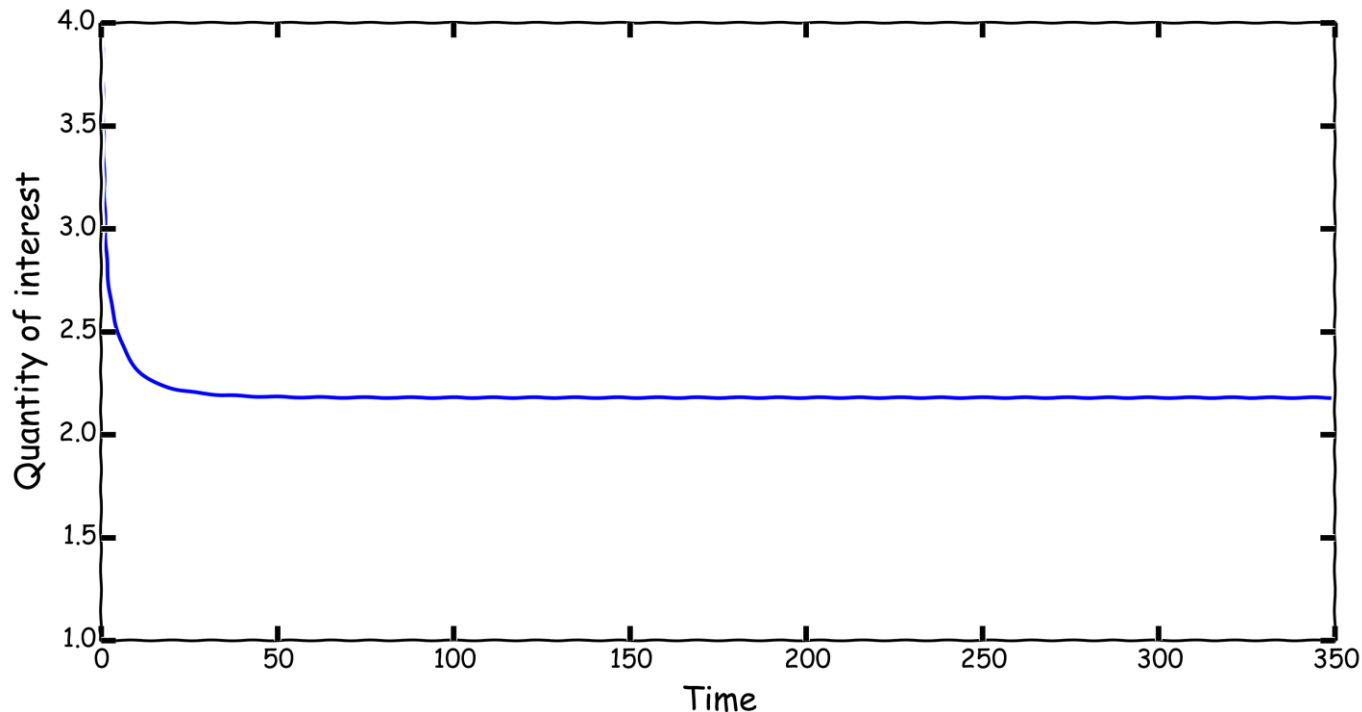
- Remember to choose wisely where to do the sampling.



# Unsteady and steady simulations

## I am running an unsteady simulations and the QOI does not change

- When you run unsteady simulations, flow variables can stop changing with time. When this happens, we say we have arrived at a steady state.
- Remember this is the exception rather than the rule.
- If you use a steady solver, you will arrive to the same solution (maybe not), in much less iterations.



# Unsteady and steady simulations

## What about steady simulations?

- First of all, steady simulations are a big simplification of reality.
- Steady simulations are a trick used by CFDers to get fast outcomes with results that might be very questionable.
- As mentioned before, most of the flows you will encounter are unsteady.
- In steady simulations we made two assumptions:
  - We ignore unsteady fluctuations. That is, we neglect the time derivative in the governing equations.
  - We perform time or iterative averaging when dealing with stationary turbulence (RANS modeling)
- The advantage of steady simulations are:
  - They require low computational resources.
  - They give fast outcomes.
  - They are easy to post-process and analyze. We usually take a lot at the last saved solution.

# Unsteady and steady simulations

## What about steady simulations?

- To run steady simulations using a general CFD solver, you need to use the appropriate solver and set the discretization scheme to deal with a steady simulation.
- As you are not solving the time derivative, you do not need to set the time step. However, you need to tell to the CFD solver how many iterations you would like to run.
- You can also set the residual controls. If you do not set the residual controls, the simulation will run until reaching the maximum number of iterations.
- As there is no time derivative (the time step is infinite), there must be a way to control the iterative marching in steady simulations. This is done by adjusting the under-relaxation factors (URF).
- Under-relaxation works by limiting the amount which a variable changes from one iteration to the next, either by modifying the solution matrix and source (implicit under-relaxation) prior to solving for a field or by modifying the field directly (explicit under-relaxation).
- In other words, under-relaxation will make the coefficient matrix more diagonally dominant.

# Unsteady and steady simulations

## What about steady simulations?

- You also need to set the under-relaxation factors (URF).
- The under-relaxation factors control the change of the variable  $\phi$ .

$$\phi_P^n = \phi_P^{n-1} + \alpha(\phi_P^{n*} - \phi_P^{n-1})$$

- If  $\alpha < 1$  we are using under-relaxation.
- Under-relaxation is a feature typical of steady solvers using the **SIMPLE** family of methods.
- Many times, steady simulations diverge because of wrongly chosen URF.
- In CFD, under-relaxation can implicit or explicit.

# Unsteady and steady simulations

## What about steady simulations?

- In explicit under-relaxation we relax the field variable,

$$\phi = \phi_{n-1} + \alpha \Delta \phi$$

- In implicit under-relaxation we relax the discretized algebraic equation variable,

$$\frac{a_P \phi}{\alpha} = \sum_N a_N \phi_N + b + \frac{1 - \alpha}{\alpha} a_P \phi_{n-1}$$

- Choosing the right under-relaxation factors (URF) is equivalent to choosing the right time step.
- You can relate URF to the CFL number as follows,

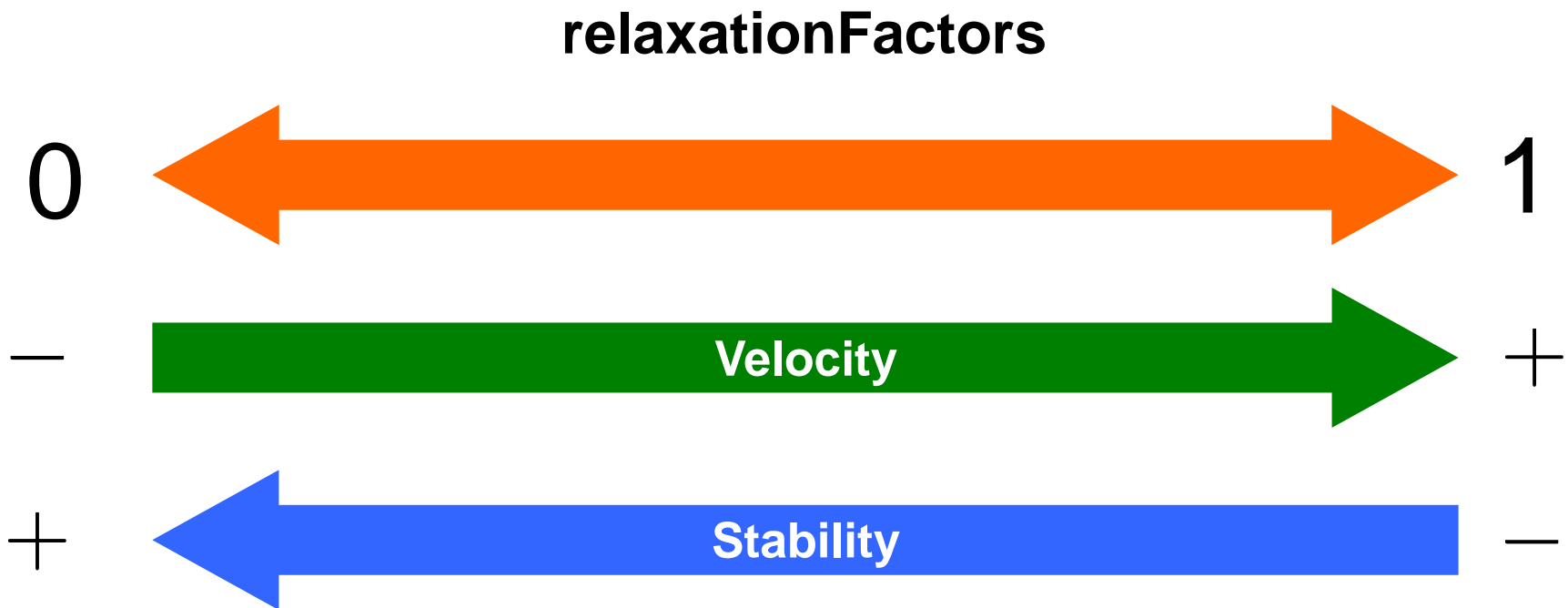
$$CFL = \frac{\alpha}{1 - \alpha} \qquad \alpha = \frac{CFL}{1 + CFL}$$

- A large CFL number is equivalent to small URF.

# Unsteady and steady simulations

## What about steady simulations?

- The under-relaxation factors are bounded between 0 and 1.



- Selecting the under-relaxation factors, it is kind of equivalent to selecting the right time step.

# Unsteady and steady simulations

## What about steady simulations?

- Finding the right under-relaxation factors involved experience and a lot of trial and error.
- Choosing the wrong under-relaxation factors can stall the convergence or give you oscillatory/noisy convergence rate (residuals and monitored quantities).
- Generally speaking, is not recommended to reduce implicit under-relaxation factors to values below 0.5 as it can stalled the convergence rate, add an oscillatory behavior or it will take much longer to convergence.
- If you hit the 0.5 mark when using implicit under-relaxation factors, it is better to stabilize the solution in a different way (increase viscosity, ramp boundary conditions, use upwind, increase corrections and so on).
- Instead, explicit under-relaxation factors can be reduced to as low as 0.1 and still obtain convergence in a reasonable number of iterations.
- It is recommended to use the values mentioned in literature (referred to as industry standard).



# Unsteady and steady simulations

## What about steady simulations?

- An optimum choice of under-relaxation factors is one that is small enough to ensure stable computation but large enough to move the iterative process forward quickly.
- Different methods (**SIMPLE**, **SIMPLEC**, **SIMPLER**), have different URF requirements.
- These are the under-relaxation factors commonly used with **SIMPLE** and **SIMPLEC** methods (industry standard),

### **SIMPLE**

<b>p</b>	→ <b>0.3</b>
<b>U</b>	→ <b>0.7</b>
<b>k</b>	→ <b>0.7</b>
<b>omega</b>	→ <b>0.7</b>
<b>epsilon</b>	→ <b>0.7</b>

### **SIMPLEC**

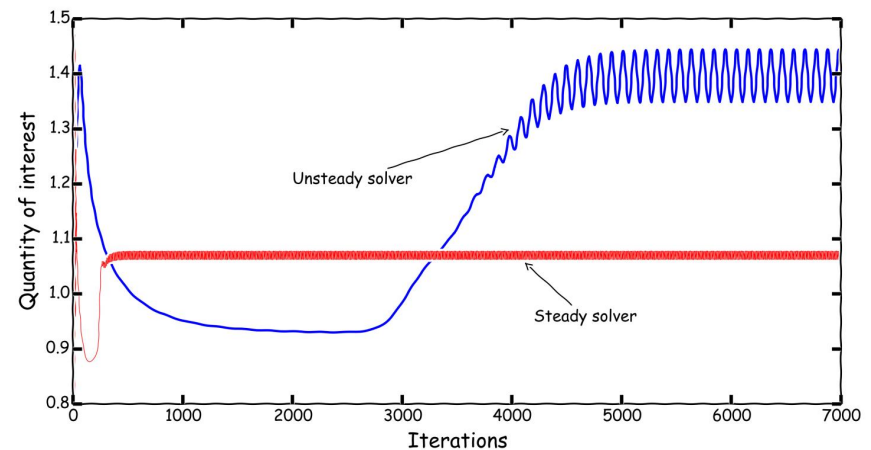
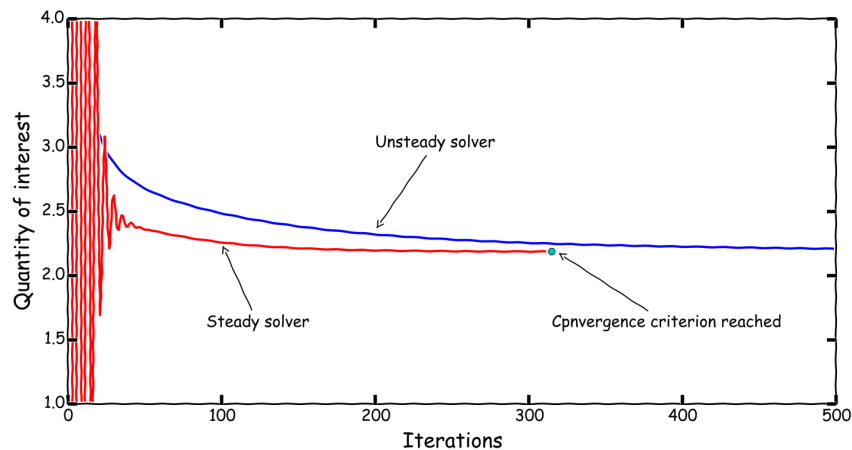
<b>p</b>	→ <b>1.0</b>
<b>U</b>	→ <b>0.9</b>
<b>k</b>	→ <b>0.9</b>
<b>omega</b>	→ <b>0.9</b>
<b>epsilon</b>	→ <b>0.9</b>

- According to the physics involved you will need to add more under-relaxation factors.
- Finding the right under-relaxation factors involved experience and a lot of trial and error.

# Unsteady and steady simulations

## Steady simulations vs. Unsteady simulations

- Steady simulations require less computational power than unsteady simulations.
- They are also much faster than unsteady simulations.
- But sometimes they do not converge to the right solution.
- They are easier to post-process and analyze (you just need to take a look at the last saved solution).
- You can use the solution of an unconverged steady simulation as initial conditions for an unsteady simulation.
- Remember, steady simulations are not time accurate, therefore we can not use them to compute time statistics or compute the shedding frequency



# Unsteady and steady simulations

## Under-relaxation factors and unsteady solvers

- It is also possible to use under-relaxation factors with unsteady solvers.
- You should be careful not to use too low URF with unsteady solvers because you might lose time accuracy.
- You can use large URF (close to one) or the industry standard URF with unsteady solvers.
- If you use low values (less than 0.5 for all variables), it is recommended to run a time convergence test to determine if you are losing time accuracy.
- The unsteady solution without URF must match the unsteady solution with URF, otherwise your solution is not time-accurate.
- When you use URF with unsteady solvers you increase the diagonal dominance of the linear system. Therefore, they improve the stability of unsteady solvers.

### Industry standard URF

#### **SIMPLE**

p	→ 0.3
U	→ 0.7
k	→ 0.7
omega	→ 0.7
epsilon	→ 0.7

#### **SIMPLEC**

p	→ 1
U	→ 0.9
k	→ 0.9
omega	→ 0.9
epsilon	→ 0.9

### Recommended URF

#### **SIMPLE – SIMPLEC – PIMPLE**

p	→ 0.7 (0.3 IN SIMPLE)
U	→ 0.7
k	→ 0.7
omega	→ 0.7
epsilon	→ 0.7

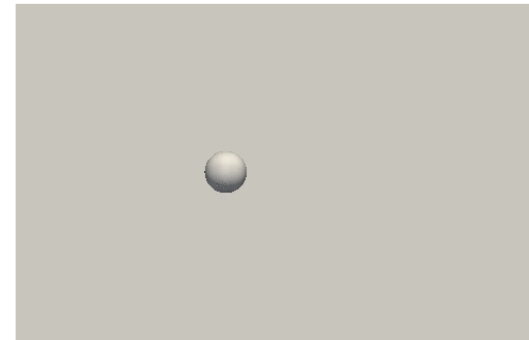
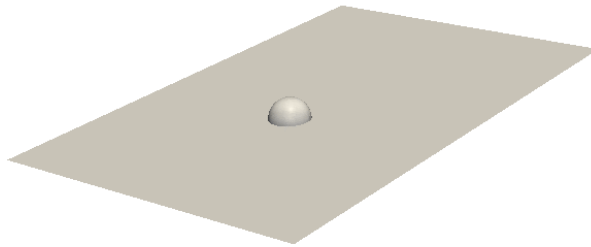
**Note: use these guidelines with unsteady solvers**

# Unsteady and steady simulations

**Unsteady or steady solver?**

# Unsteady and steady simulations

## Unsteady or steady solver?



Hairpin vortices - Hemisphere

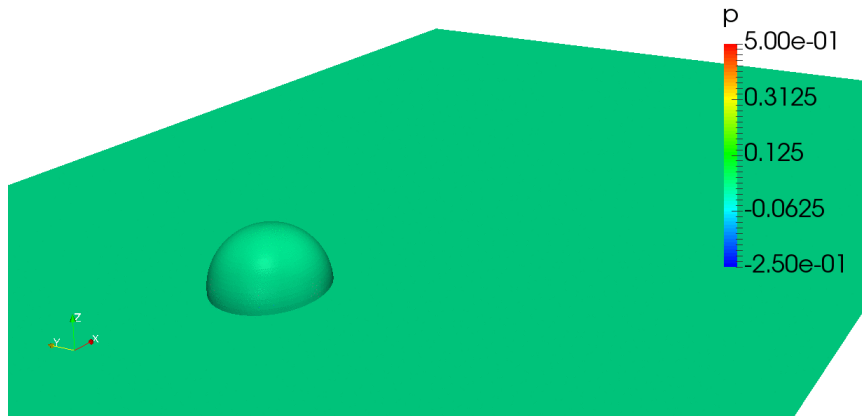
[www.wolfdynamics.com/wiki/hairpin\\_vortices/uns\\_or\\_ste.gif](http://www.wolfdynamics.com/wiki/hairpin_vortices/uns_or_ste.gif)

# Unsteady and steady simulations

## Unsteady or steady solver?



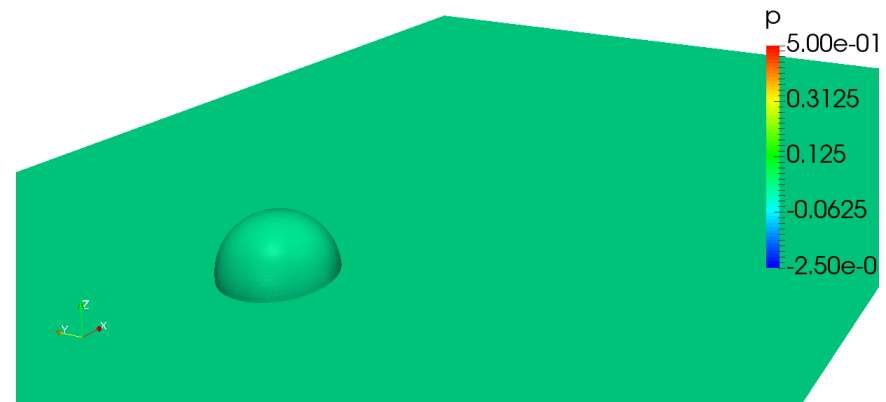
Iteration: 0.000000



### Steady simulation

[www.wolfdynamics.com/wiki/hairpin\\_vortices/st/ani1.gif](http://www.wolfdynamics.com/wiki/hairpin_vortices/st/ani1.gif)

Time: 0



### Unsteady simulation

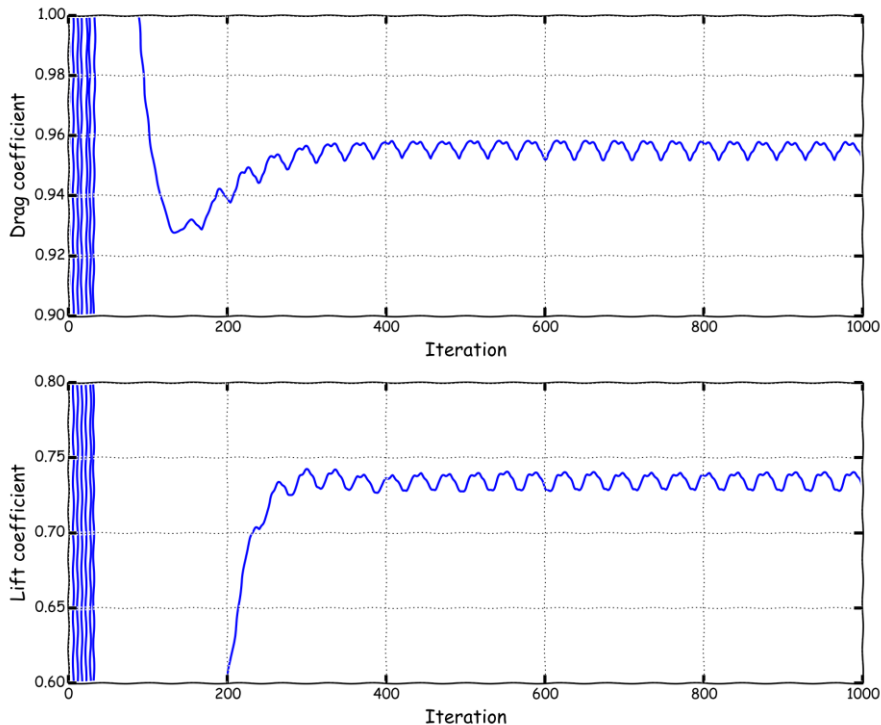
[www.wolfdynamics.com/wiki/hairpin\\_vortices/uns0/ani\\_smallcfl.gif](http://www.wolfdynamics.com/wiki/hairpin_vortices/uns0/ani_smallcfl.gif)

- Steady simulations are not time accurate, hence we can not use them to compute time statistics or compute the shedding frequency.
- Generally speaking, and in the absence of highly unsteady flows, steady simulations should give a result that is close to the mean solution of an unsteady simulation.
- Be careful when post-processing steady simulations, the animations you obtain does not represent time scales, they only show you how the solution change from iteration to iteration.
- When post-processing steady simulations, you should use the last saved iteration.
- You can also compute the average of a series of iterations.

- Unsteady simulation are time-accurate.
- They capture the unsteadiness of the flow (time scales).
- You can use these simulations to compute shedding frequency or other time scales
- Post-processing unsteady simulations can be difficult and time-consuming.
- When you post-process unsteady simulations, you access all the time-steps saved.
- You can also compute the average of a series of time-steps.
- Remember, you need to define an adequate saving frequency and time-step.
- You can use steady simulations to initialize unsteady simulations.

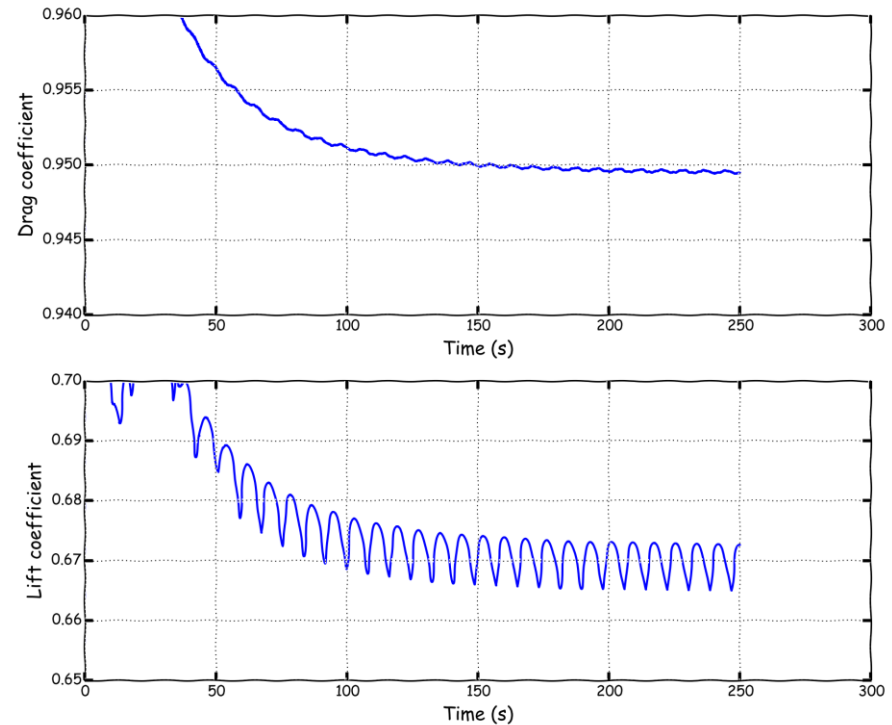
# Unsteady and steady simulations

## Unsteady or steady solver?



### Steady simulation

- Steady simulations are not time accurate, hence we can not use them to compute time statistics or compute the shedding frequency.
- Generally speaking, and in the absence of highly unsteady phenomena, steady simulations should give a result that is close to the mean solution of an unsteady simulation.
- Steady simulations are a very good starting point for unsteady simulations.

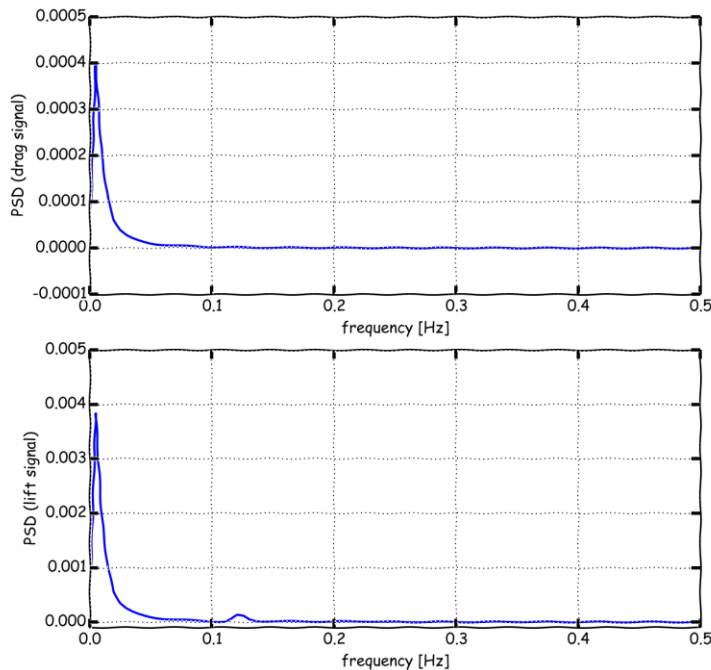


### Unsteady simulation

- Unsteady simulations are time-accurate.
- They capture the unsteadiness of the flow (time scales).
- You can use these simulations to compute shedding frequency, but remember, you need to define an adequate saving frequency and time-step.
- Numerical diffusion can give you the impression that you have arrived at a steady state.

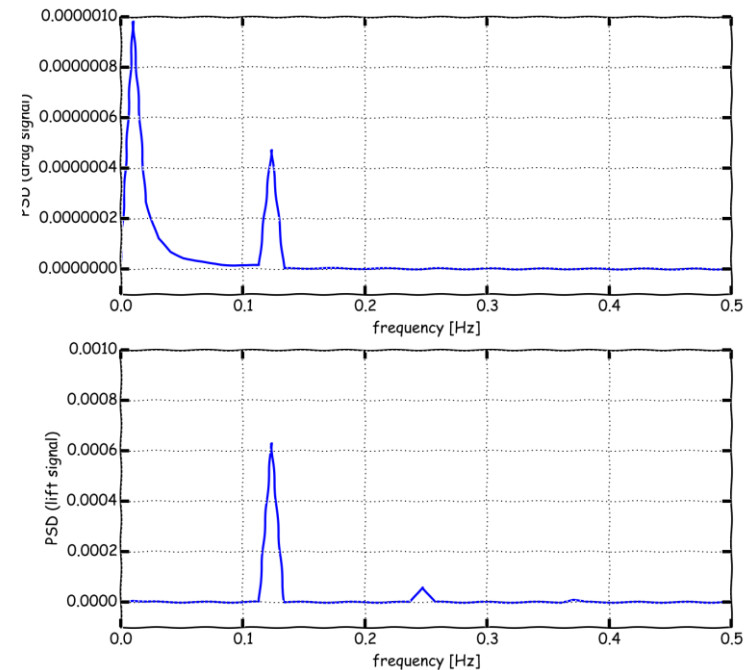
# Unsteady and steady simulations

## Unsteady or steady solver?



### Coarse mesh

- Due to numerical diffusion (under-resolve time and/or spatial scales), it is not possible to use this solution to conduct a time series analysis of the solution.



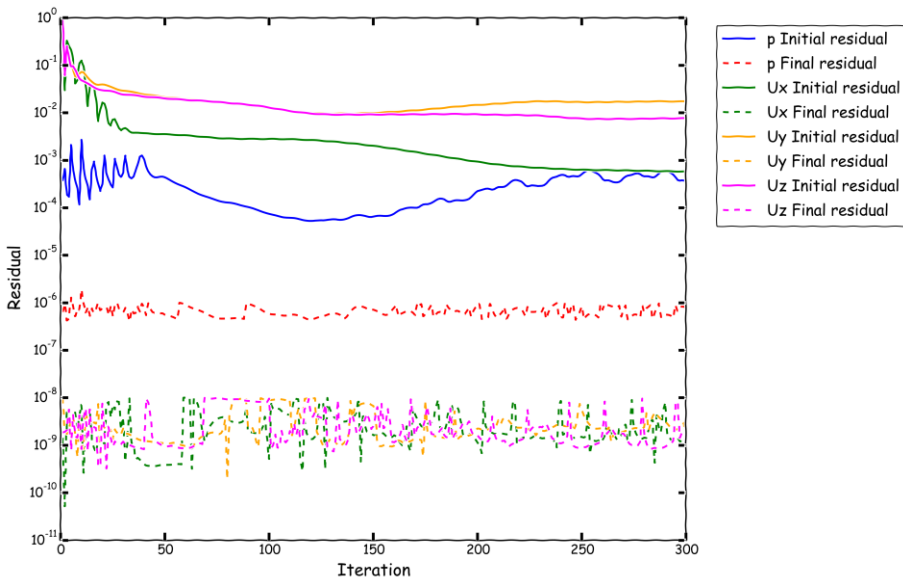
### Fine mesh

- As the accuracy is better in the fine mesh, it manages to capture the shedding frequency.

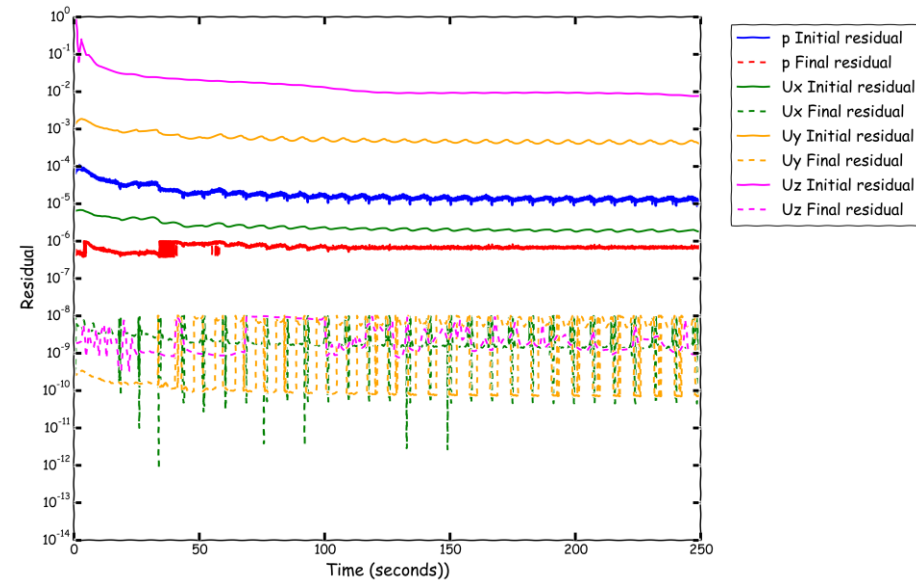


# Unsteady and steady simulations

## Unsteady or steady solver?



Steady simulation residuals



Unsteady simulation residuals

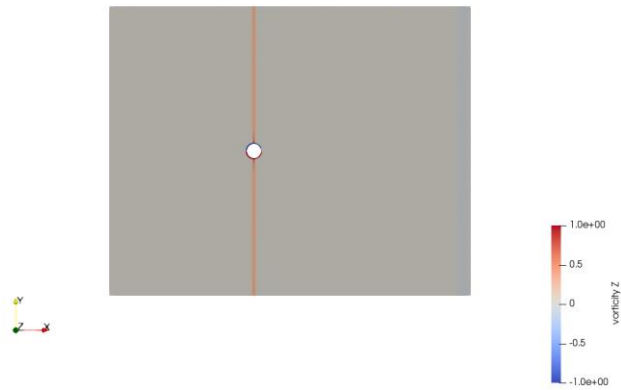
# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- ~~2. Important concepts to remember~~
- ~~3. The Finite Volume Method: An overview~~
- ~~4. Navier-Stokes equations and pressure-velocity coupling~~
- ~~5. On the CFL number~~
- ~~6. Unsteady and steady simulations~~
- 7. Understanding the residuals**
- ~~8. Boundary conditions and initial conditions~~
- ~~9. The FVM in OpenFOAM: some implementation details and computational pointers~~
- ~~10. Best standard practices – General guidelines~~
- ~~11. Final remarks~~

# Understanding the residuals

- To demonstrate how to interpret the residuals, let us use the flow around a cylinder case at different Reynolds number.
- We will work with a Reynolds number equal to 20 (steady) and 200 (unsteady).

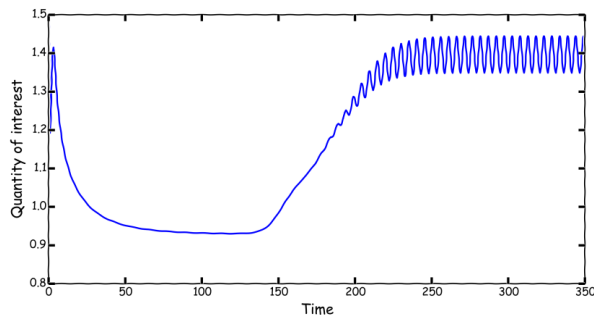
Time: 0.000000



## Unsteady behavior – Vortex shedding

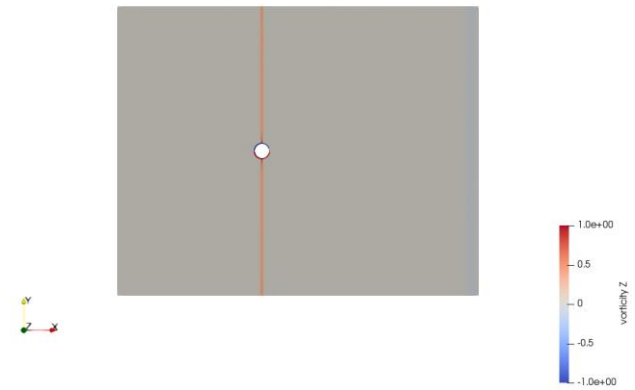
Notice that to accelerate the convergence rate we are using non-uniform initialization

[www.wolfdynamics.com/training/fvm/FVM\\_uns1.gif](http://www.wolfdynamics.com/training/fvm/FVM_uns1.gif)



Monitored quantity of interest – Drag coefficient

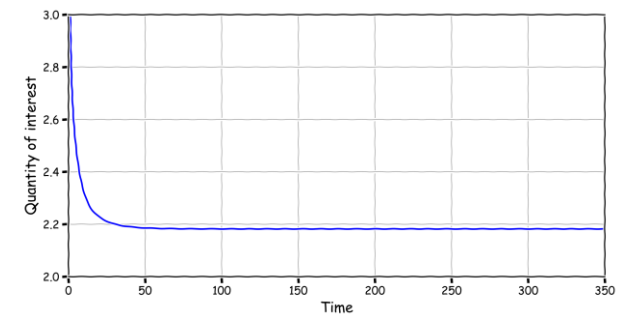
Time: 0.000000



## Steady behavior – No flow separation

The initial unsteadiness is due to the non-uniform initialization

[www.wolfdynamics.com/training/fvm/FVM\\_ste1.gif](http://www.wolfdynamics.com/training/fvm/FVM_ste1.gif)

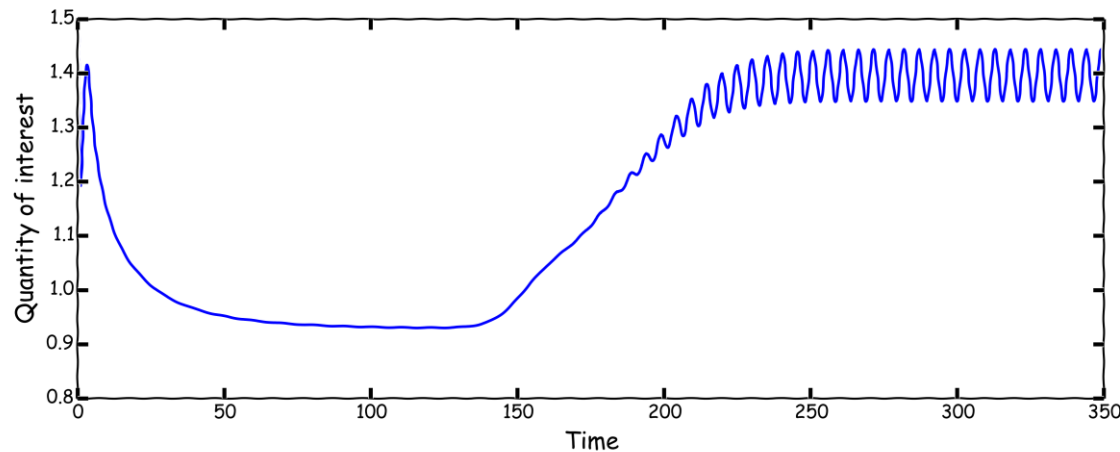


Monitored quantity of interest – Drag coefficient

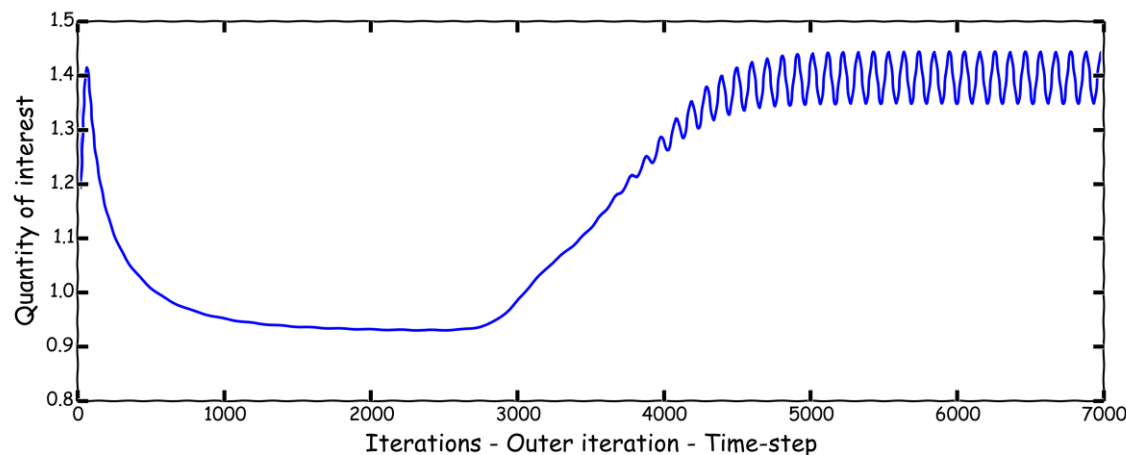


# Understanding the residuals

- Before talking about residuals, let us clarify something.
- When we talk about iterations in unsteady simulations, we are talking about the time-step or outer-iterations.



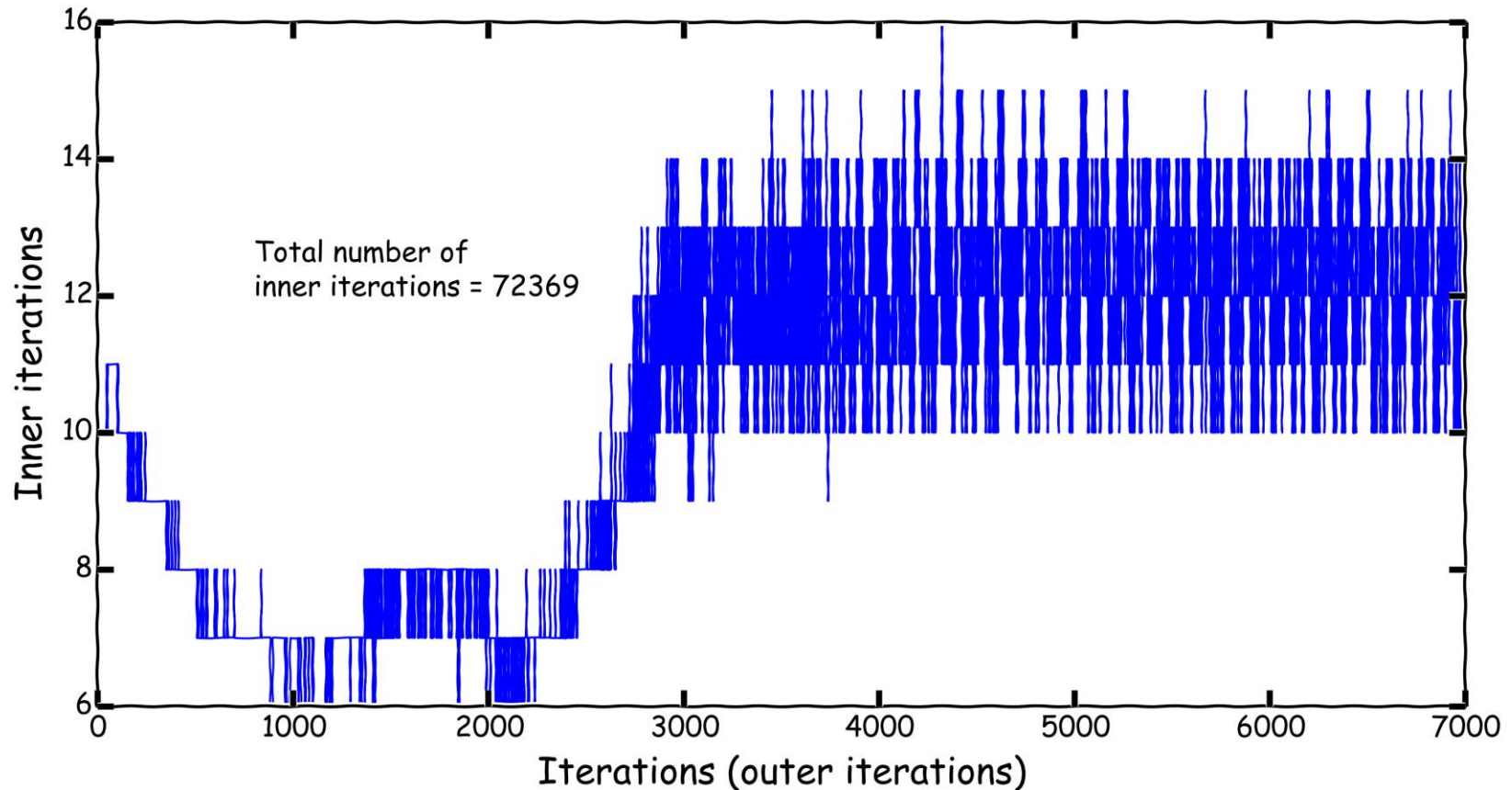
← 1. To arrive to this physical time



← 2. We iterate this many times

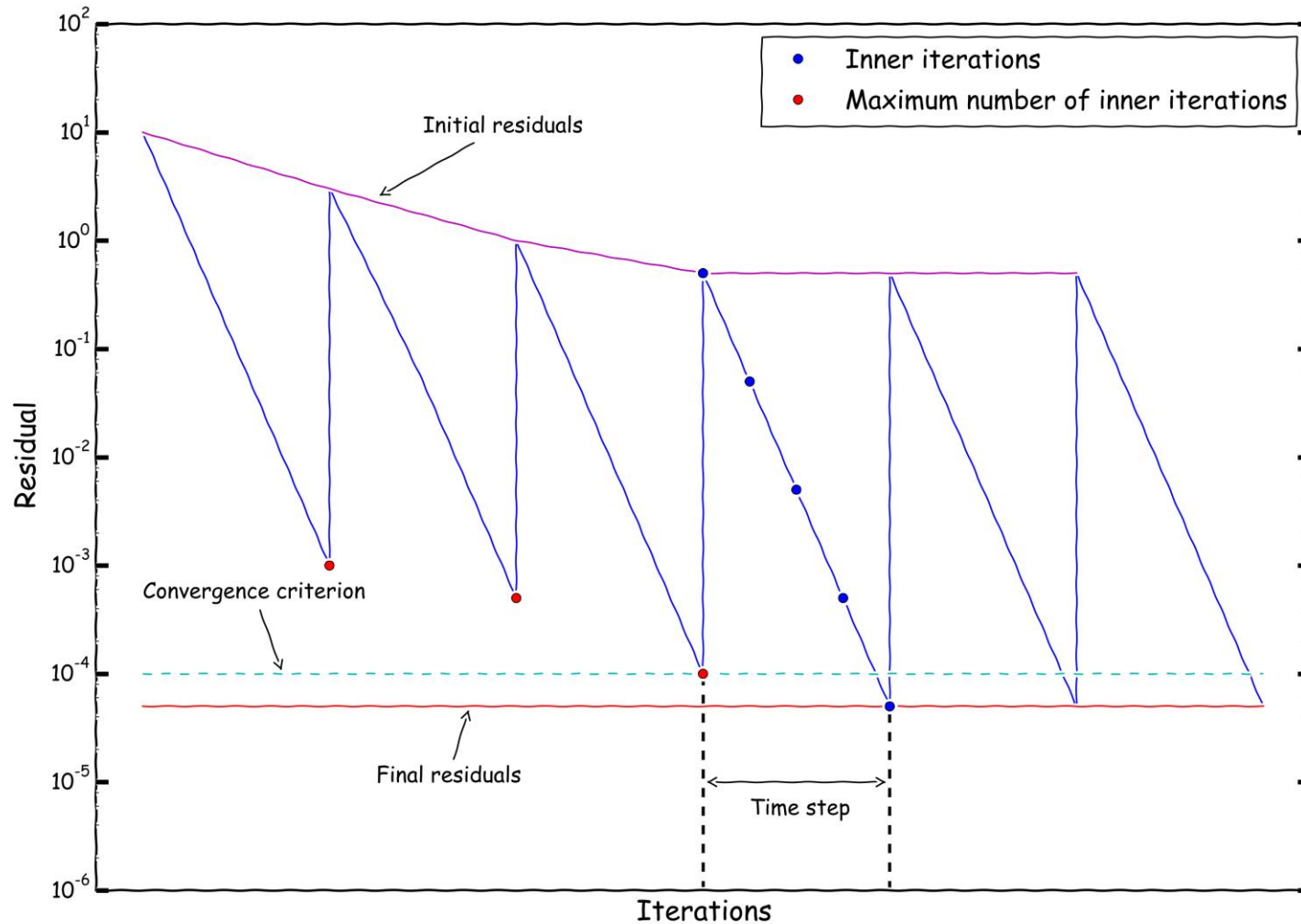
# Understanding the residuals

- And we iterate inside each time-step (or outer-iteration), until reaching the linear solver tolerance or maximum number of iterations.



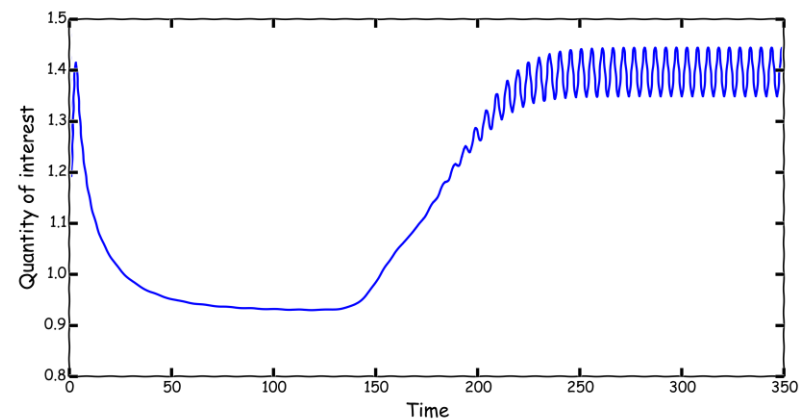
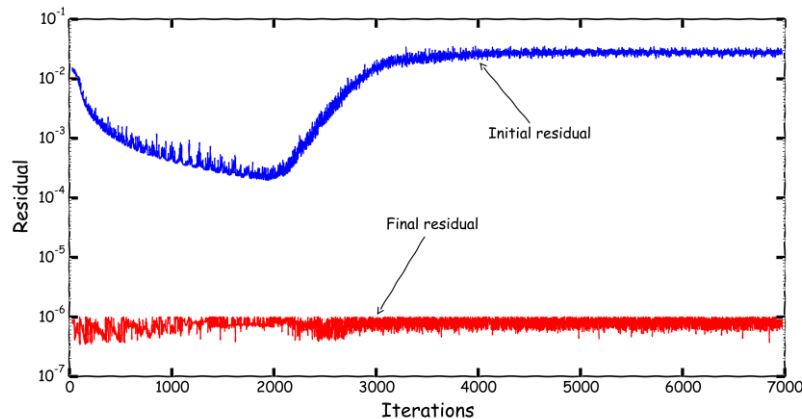
# Understanding the residuals

- This is a typical residual plot for an unsteady simulation.



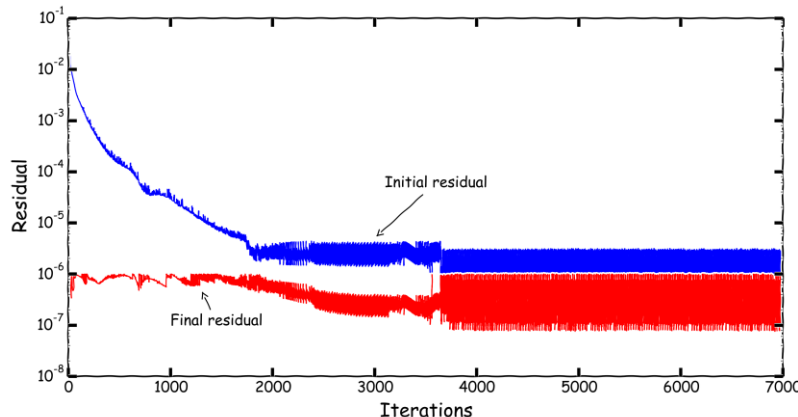
# Understanding the residuals

- This is a typical residual plot for an unsteady simulation.
- In this case, the fact that the initial residuals are not decreasing is not an indication that the solution is diverging. Most likely this behavior is due to unsteadiness.
- However, the final residuals should reach the predefined tolerance criterion (linear solvers).
- Remember, residuals are not a direct indication that you are converging to the right solution.
- It is better to monitor a quantity of interest.
- No need to say that you should get physically realistic values.
- You should also monitor stability.
- To monitor the stability of the solution, you can check the minimum and maximum values of the field variables.
- If you have bounded quantities, check that you do not have over-shoots or under-shoots.

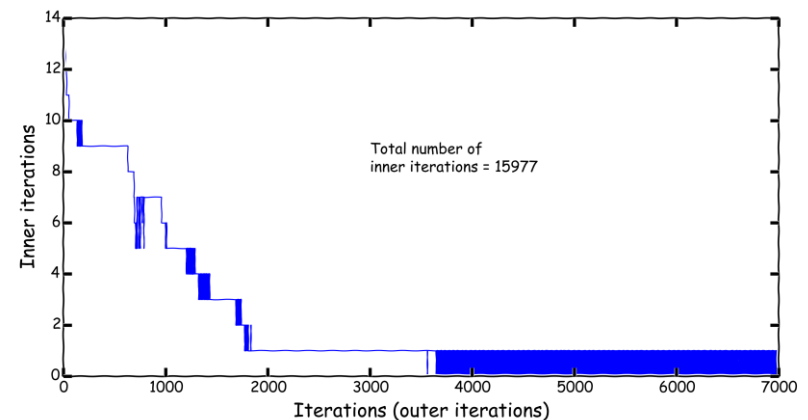


# Understanding the residuals

- This is the residual plot of an unsteady solution that has reached a steady-state behavior.
- Notice that after a couple of thousands iterations the initial residuals and final residuals are the same.
- We are plotting the residuals against iterations (or time-steps or outer-iterations).



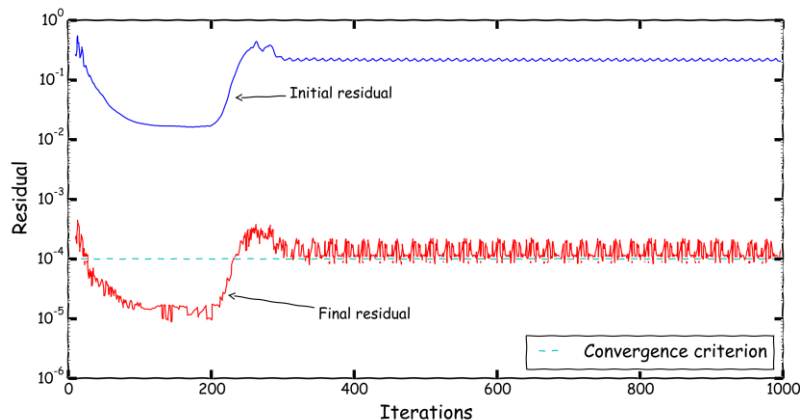
- And this is the plot of the number of inner-iterations against the number of outer-iterations (time-steps).
- Notice that after 2000 time-steps, the solution arrives to the convergence criterion (linear solvers).
- It is important to do at least one iteration.
- Most CFD solvers will let you choose the minimum and maximum number of iterations (in the linear solvers).



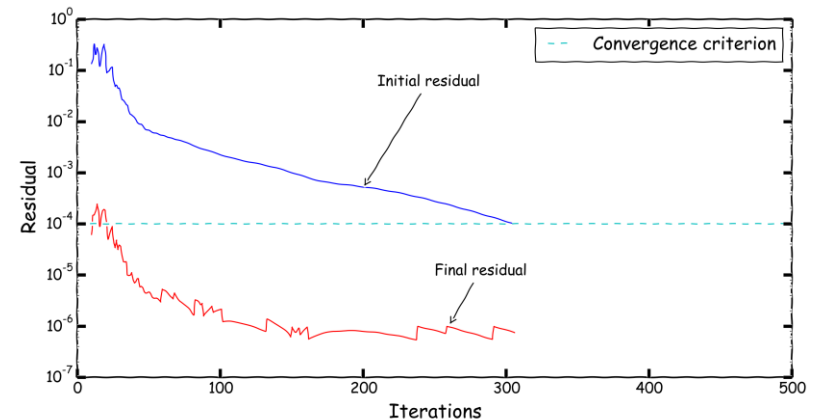


# Understanding the residuals

- Let us study the residual plot of an unsteady solution (or flow), using a steady solver.
- Notice that we are making the distinction between steady solver and unsteady solution.
- Here the initial residuals are not falling (stalled convergence). This is an indication of an unsteady solution or the wrong numerics.

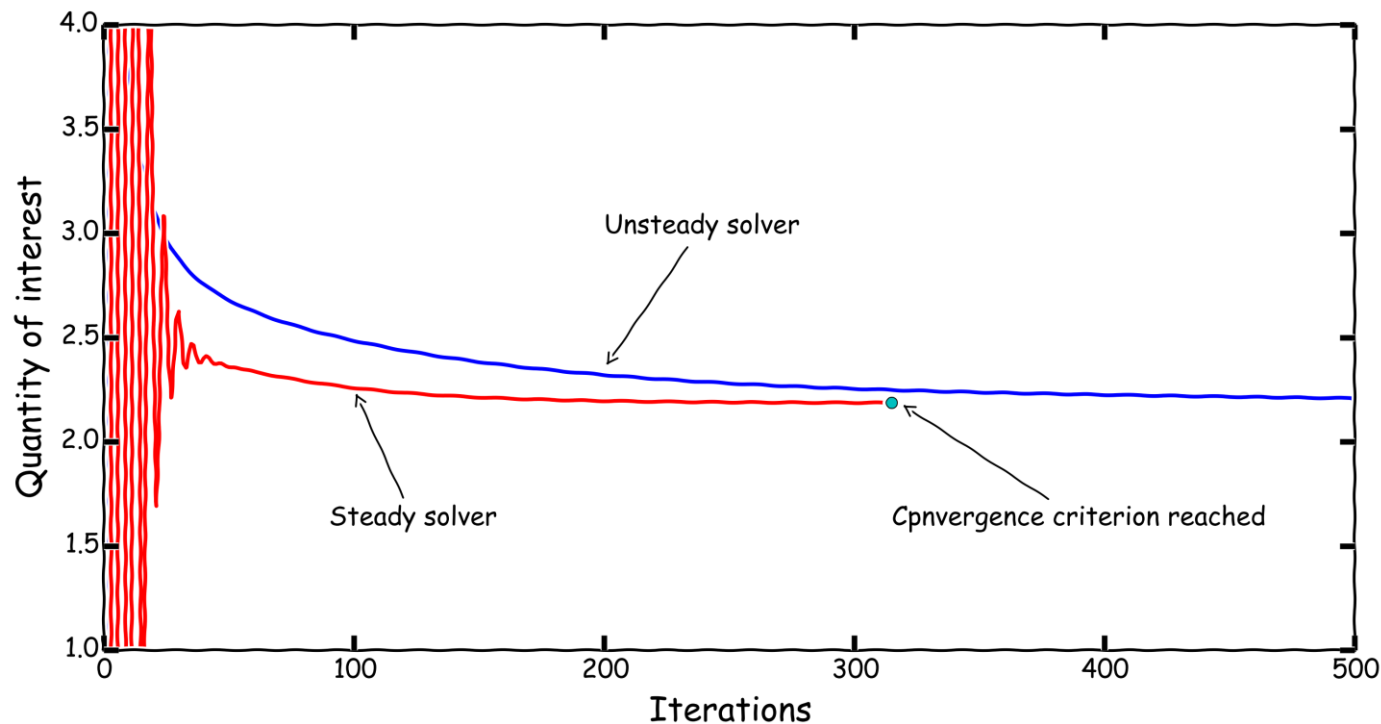


- Let us study the residual plot of an steady solution (or flow), using a steady solver.
- In this case, the initial residuals are falling below the convergence criterion (monolithic convergence), hence we have reached a steady-state.
- In comparison to unsteady solvers, steady solvers require less iterations to arrive to a converge solution, if they arrive.



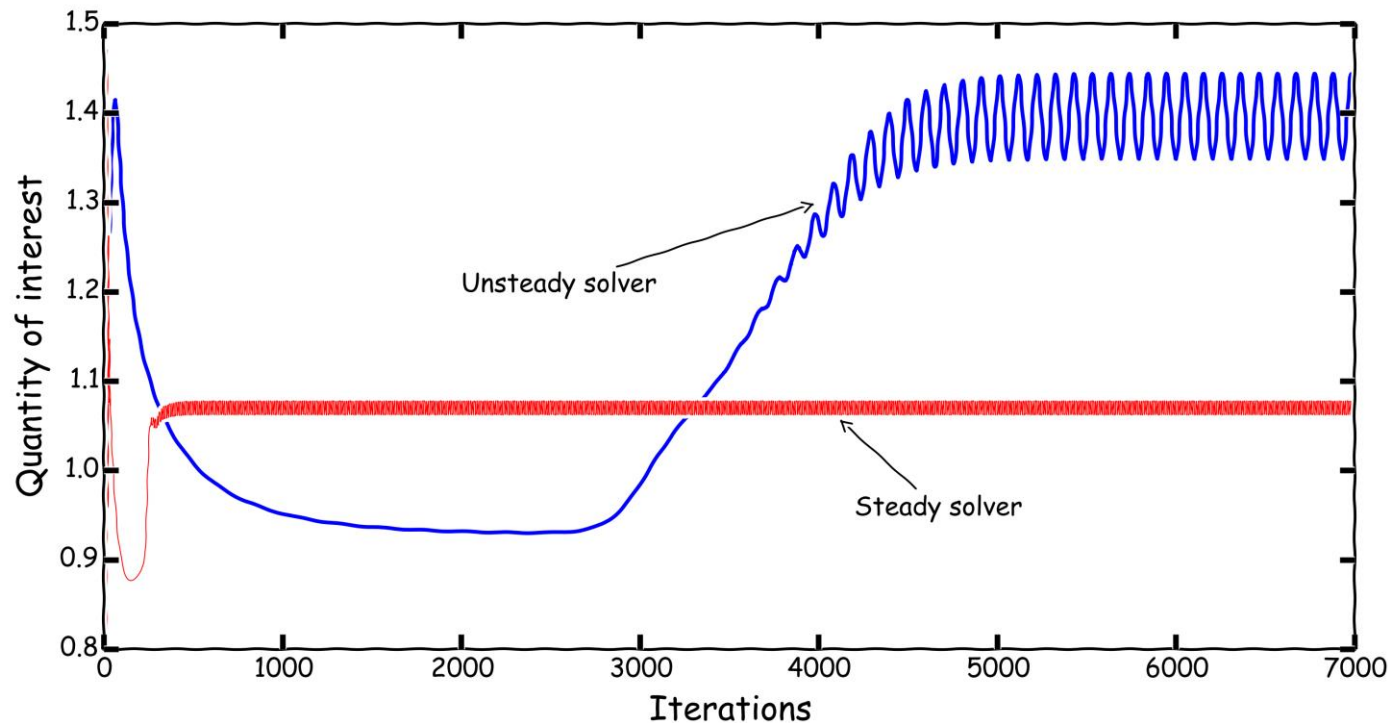
# Understanding the residuals

- For a steady flow (low Reynolds number), let us compare an integral quantity computed using a steady solver and the same quantity computed using an unsteady solver
- As we can see, both quantities are roughly speaking the same.
- In comparison to unsteady solvers, steady solvers require less iterations to arrive to a converge solution (if they arrive).
- Unsteady solvers need to run for longer times in order to get an average solution.



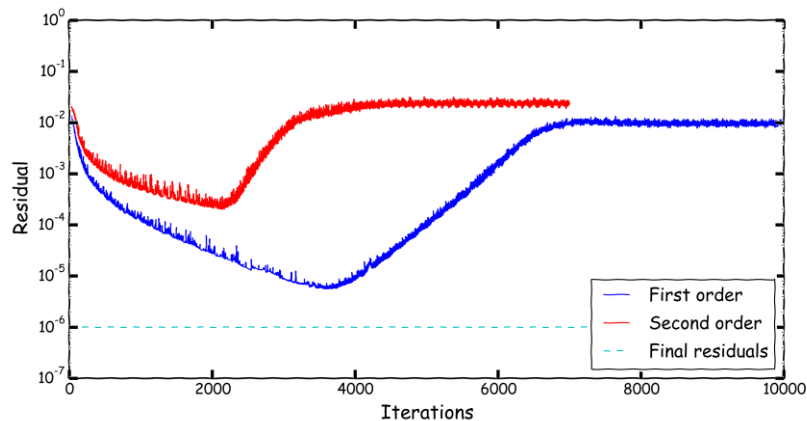
# Understanding the residuals

- For an unsteady flow (high Reynolds number), let us compare an integral quantity computed using a steady solver and the same quantity computed using an unsteady solver
- As we can see, the outcomes are very different.
- When the flow is unsteady, steady solvers do not capture well the mean solution.
- On the other hand, unsteady solver captures well the unsteadiness.
- Remember, unsteady solution can be averaged.

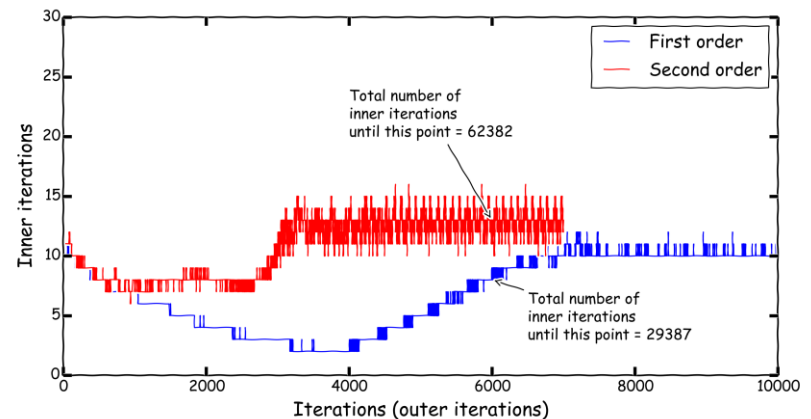


# Understanding the residuals

- Finally, let us compare the residuals of a first order and a second order numerical scheme.
- This is the residual plot of an unsteady simulation.
- Both methods are converging to the desired tolerance.
- Also, the fact that the residuals drop faster using a first order methods, does not mean that they are better.

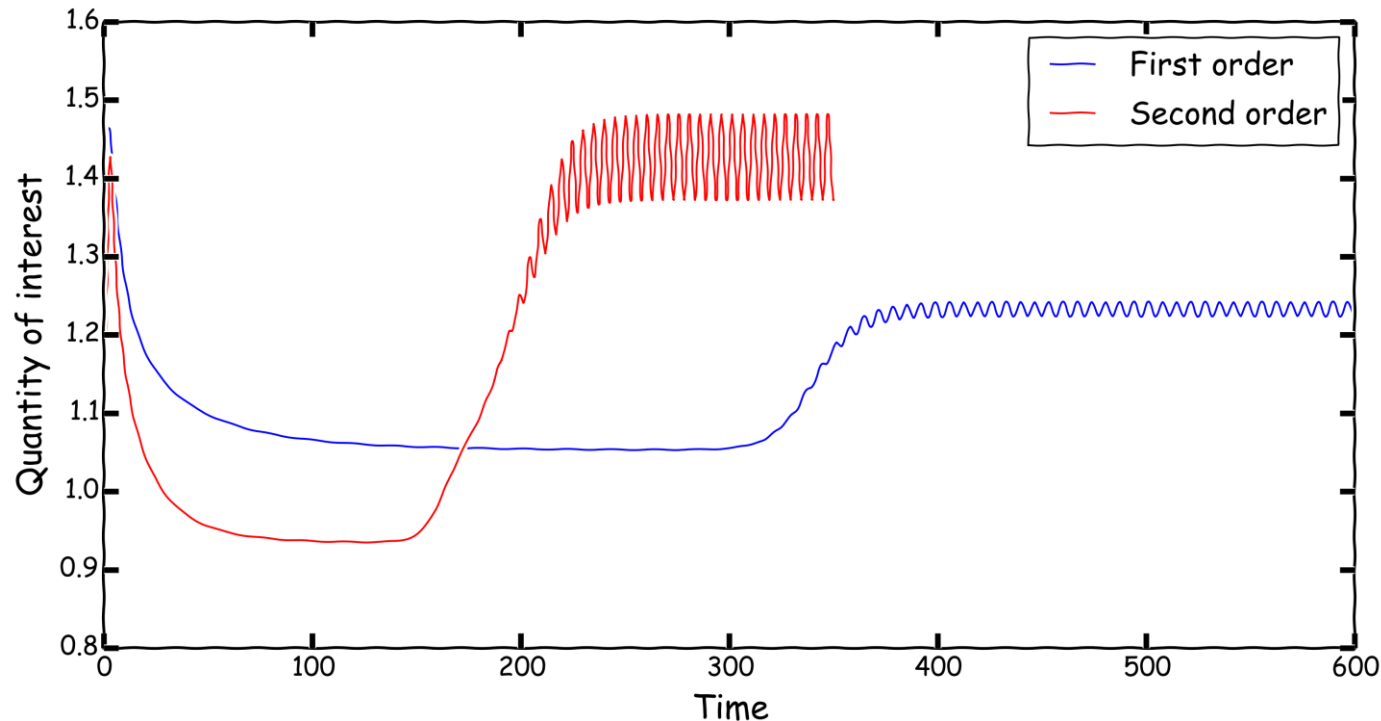


- This is the plot of the number of inner iterations against the number of outer iterations (time-steps).
- As you can see, the first order method is less computationally expensive.
- However, the fact that the first order method converges faster and is less computationally expensive, it does not mean it is better.



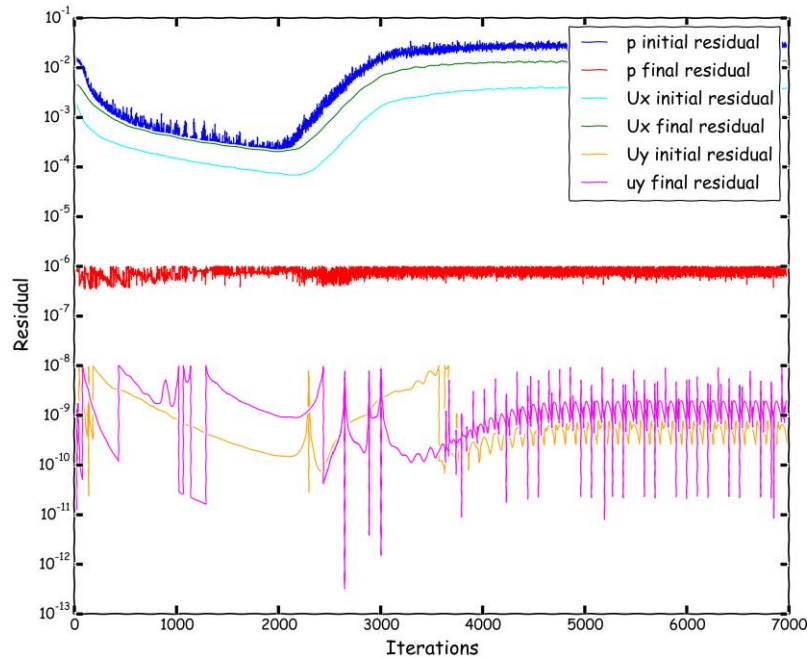
# Understanding the residuals

- However, the fact that the first order method converge faster and is less computationally expensive, it does not mean it is better.
- As you can see, first order methods highly under-predict the quantity of interest.
- In this case, the first order method takes more time to onset the instability (as they are highly diffusive).

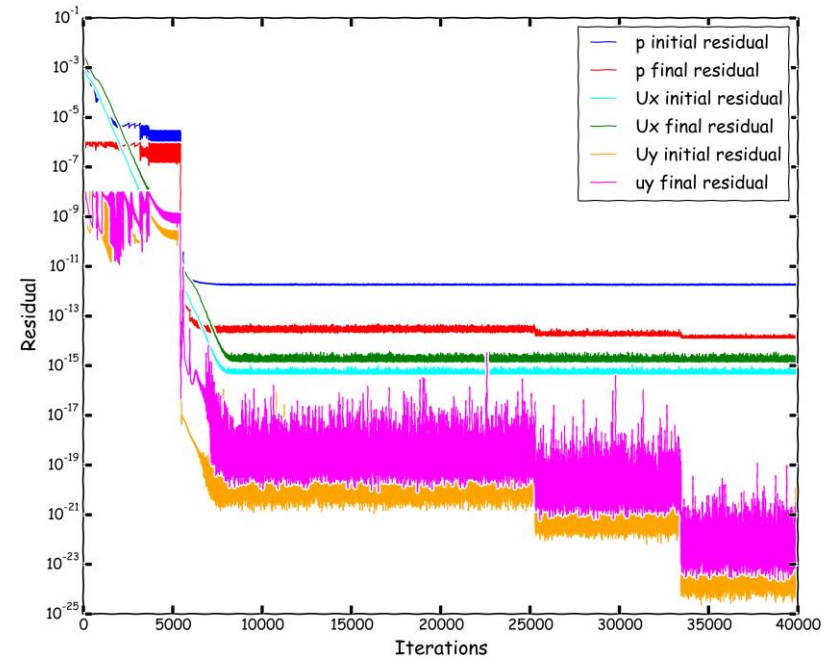


# Understanding the residuals

- This is the output of all residuals for the unsteady case (Reynolds number equal to 200)

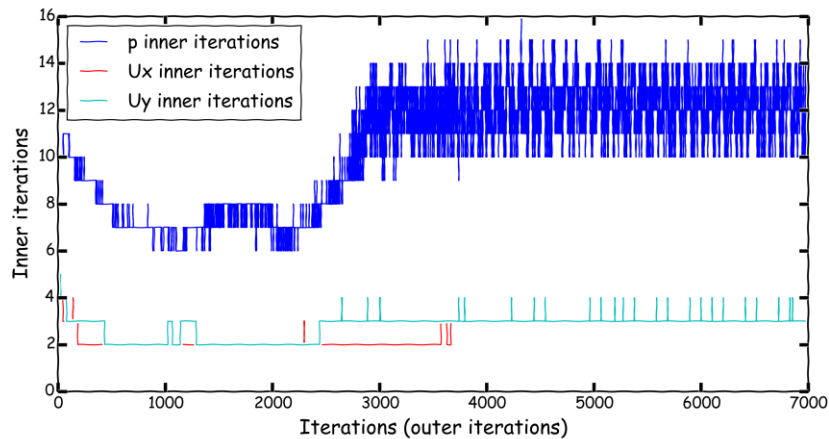


- This is the output of all residuals for the steady case (Reynolds number equal to 20).
- The jumps are due to the changes in tolerance introduced while running the simulation.

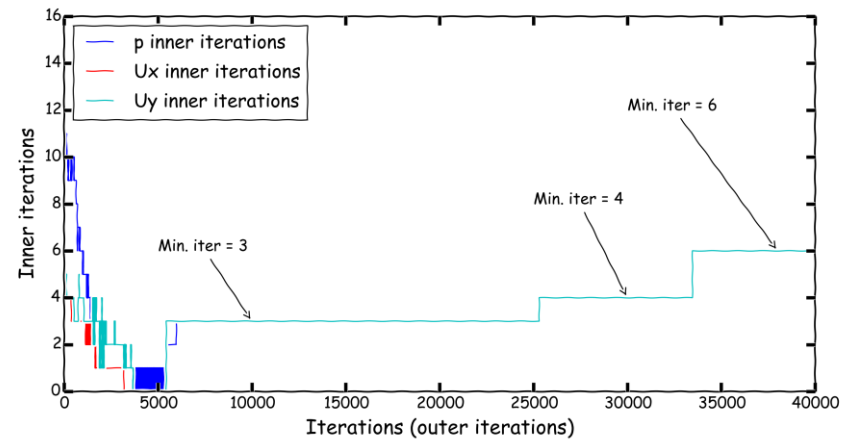


# Understanding the residuals

- This is the output of the inner-iterations against the outer-iterations for the unsteady case (Reynolds number equal to 200).

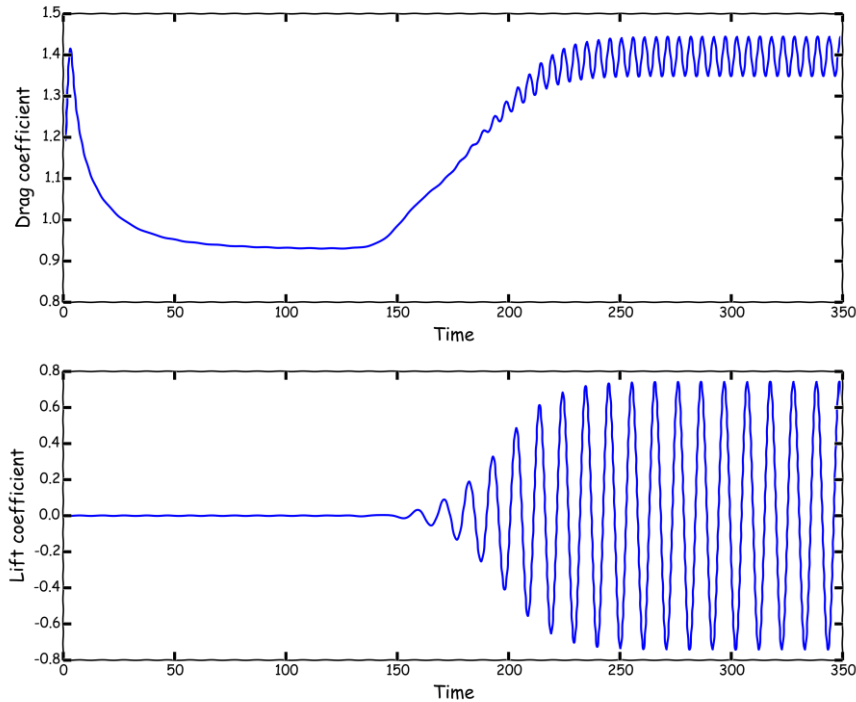


- This is the output of the inner-residual against the outer-residuals for the steady case (Reynolds number equal to 20).

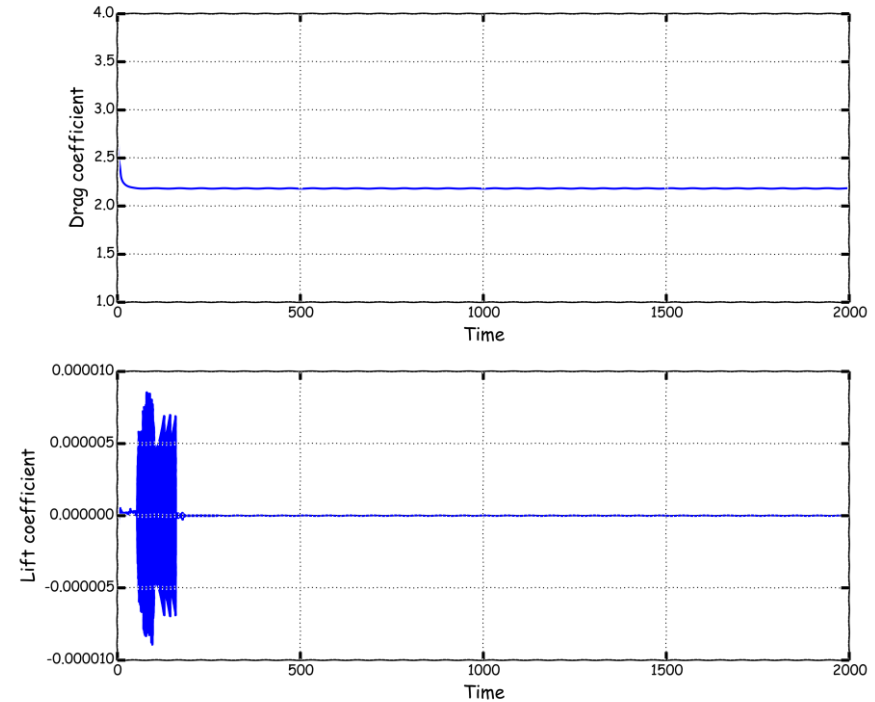


# Understanding the residuals

- This is the output of the aerodynamic coefficients for the unsteady case (Reynolds number equal to 200).



- This is the output of the aerodynamic coefficients for the steady case (Reynolds number equal to 20).

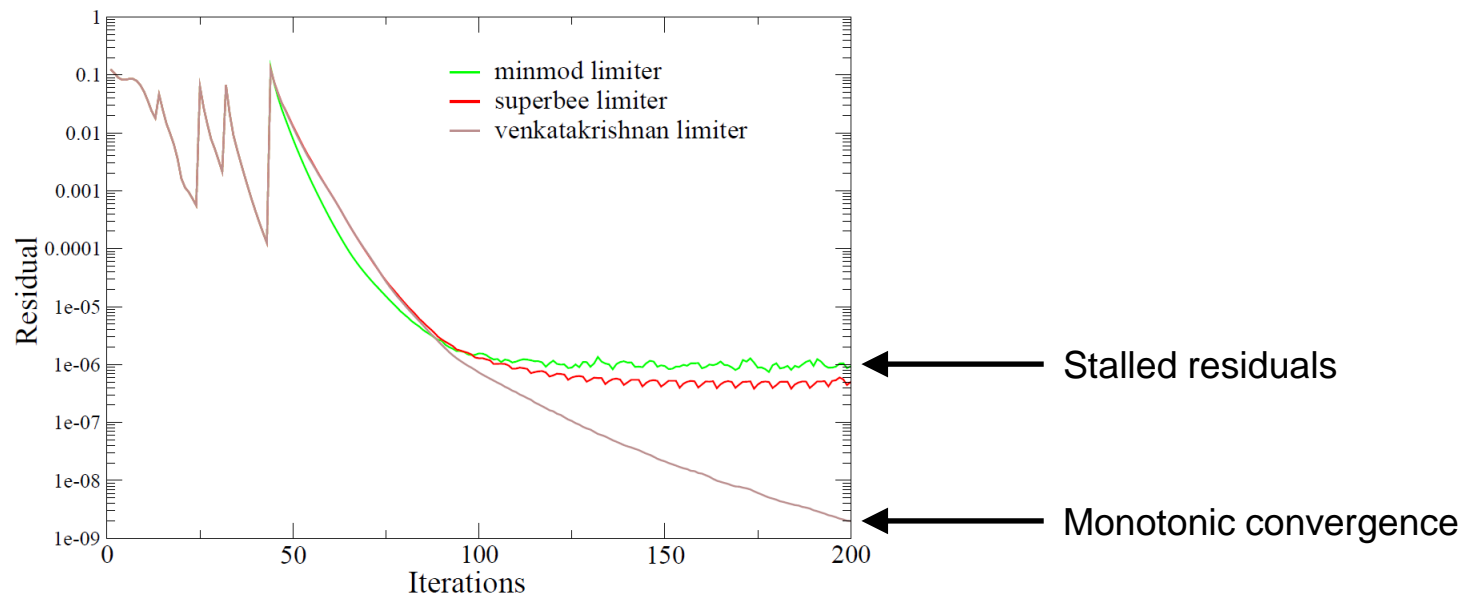




# Understanding the residuals

## Stalled residuals and effect of gradient limiters on convergence rate

- Some gradient/slope limiters can adversely affect convergence rate towards steady state (this behavior has been extensively documented).
- In some cases, the residuals get stalled (or flattened) even if you keep the solution running for long times, and even if you evidence that a monitored quantity of interest is converging.
- Stalled residuals are mainly due to unsteadiness or the effect of gradient limiters.
- This behavior (which is mainly annoying), can be remediated by changing the gradient limiter or switching to an unsteady solver.
- In some cases, this behavior can add some numerical diffusion to the solution.



# Understanding the residuals

## **Support slides:**

- **Under relaxation-factors.**
- **Steady and unsteady simulations.**
- **Solution of the linear system and preconditioning**

# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- ~~2. Important concepts to remember~~
- ~~3. The Finite Volume Method: An overview~~
- ~~4. Navier-Stokes equations and pressure-velocity coupling~~
- ~~5. On the CFL number~~
- ~~6. Unsteady and steady simulations~~
- ~~7. Understanding the residuals~~
- 8. Boundary conditions and initial conditions**
- ~~9. The FVM in OpenFOAM: some implementation details and computational pointers~~
- ~~10. Best standard practices – General guidelines~~
- ~~11. Final remarks~~

# Boundary conditions and initial conditions

## On the initial boundary value problem (IBVP)

- First of all, when we use a CFD solver to find the approximate solution of the governing equations, we are solving an Initial Boundary Value Problem (IBVP).
- In an IBVP, we need to impose appropriate boundary conditions and initial conditions.
- Boundary conditions are a required component of the numerical method, they tell the solver what is going on at the boundaries of the domain.
- You can think of boundary conditions as source terms.
- Initial conditions are also a required component of the numerical method, they define the initial state of the problem, and from this initial guess we start to iterate.
- No need to say that the boundary conditions and initial conditions need to be physically realistic.

# Boundary conditions and initial conditions

## A few words about boundary conditions

- Boundary conditions (BC) can be divided into three fundamental mathematical types:
  - **Dirichlet boundary conditions:** when we use this BC, we prescribe the value of a variable at the boundary.
  - **Neumann boundary conditions:** when we use this BC, we prescribe the gradient normal to the boundary.
  - **Robin Boundary conditions:** this BC is a mixed of Dirichlet boundary conditions and Neumann boundary.
- You can use any of these three boundary conditions in any general CFD solver, OpenFOAM included.

# Boundary conditions and initial conditions

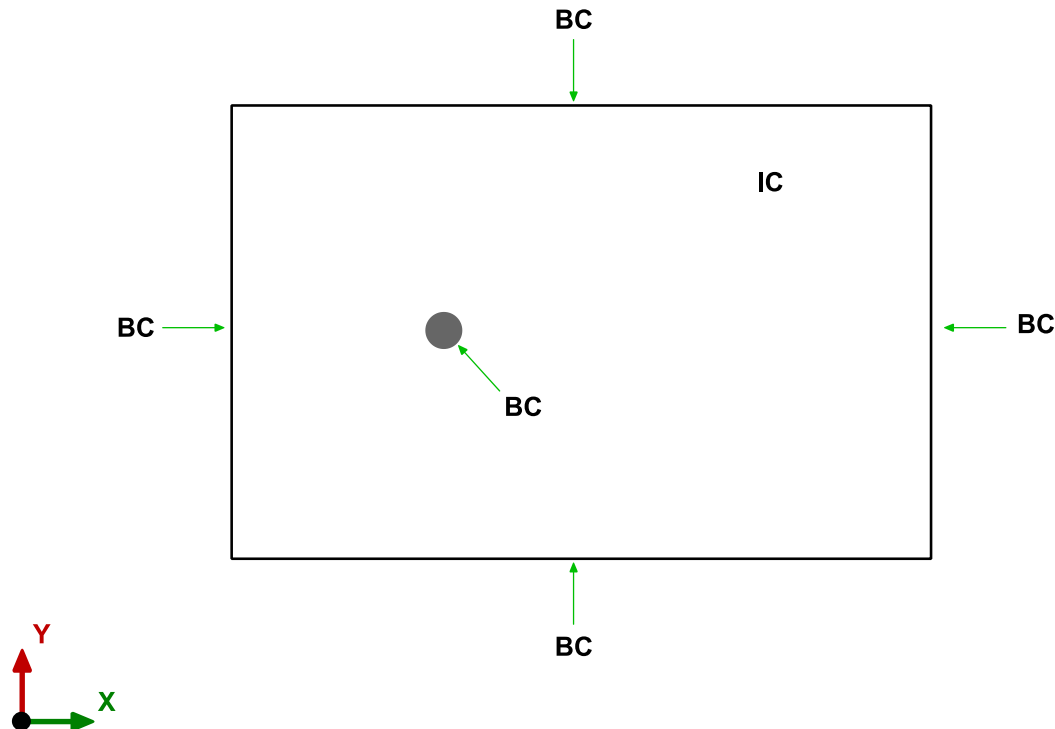
## A few words about boundary conditions

- During this discussion, the semantics is not important, that depends of how you want to call the BCs or how they are named in the solver, *i.e.*, in, inlet, inflow, velocity inlet, incoming flow and so on.
- Defining boundary conditions involves:
  - Finding the location of the boundary condition in the domain.
  - Determining the boundary condition type.
  - Giving the required physical information.
- The choice of the boundary conditions depend on:
  - Geometrical considerations.
  - Physics involved.
  - Information available at the boundary condition location.
  - Numerical considerations.
- And most important, you need to understand the physics involved.

# Boundary conditions and initial conditions

## A few words about boundary conditions

- To define boundary conditions, you need to know the location of the boundaries (where they are in your mesh).
- You also need to supply the information at the boundaries.
- Last but not least important, you must know the physics involved.



# Boundary conditions and initial conditions

## A few words about initial conditions

- Initial conditions (IC) can be divided into two groups:
  - **Uniform initial conditions.**
  - **Non-uniform initial conditions.**
- For non-uniform IC, the value used can be obtained from:
  - Another simulation, including a solution with different grid resolution.
  - A potential solver.
  - Experimental results.
  - A mathematical function
  - Reduced order models.



# Boundary conditions and initial conditions

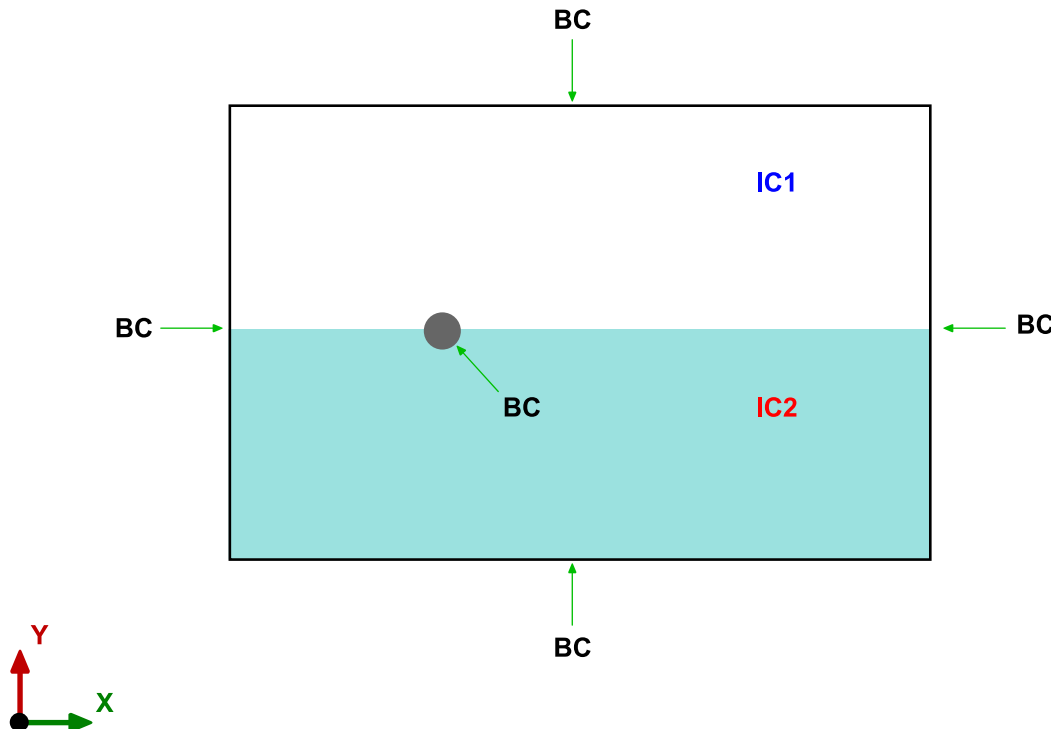
## A few words about initial conditions

- Defining initial conditions involves:
  - Finding the location of the initial condition in the domain.
  - Determining the initial condition type.
  - Giving the required physical information.
- The choice of the initial conditions depend on:
  - Geometrical considerations.
  - Physics involved.
  - Information available.
  - Numerical considerations.
- And most important, you need to understand the physics involved.

# Boundary conditions and initial conditions

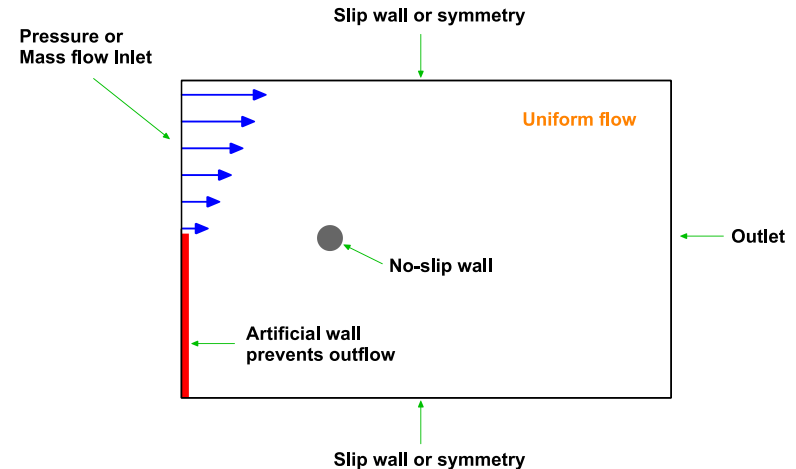
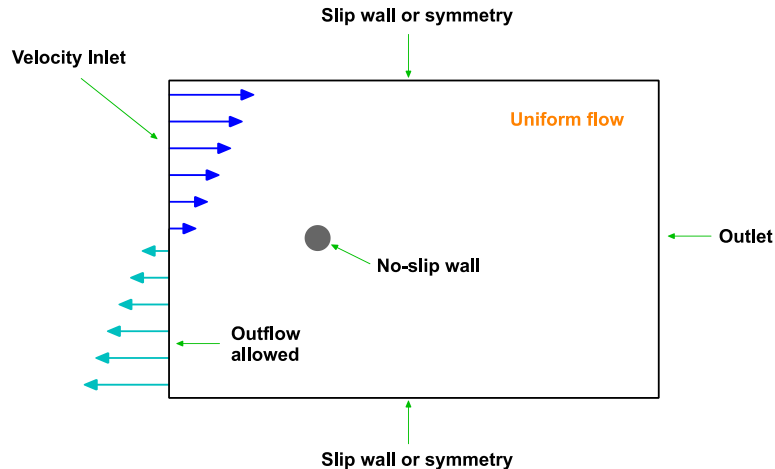
## A few words about initial conditions

- For initial conditions, you need to supply the initial information or initial state of your problem.
- This information can be a uniform value or a non-uniform value.
- You can apply the initial conditions to the whole domain or separated zones of the domain.
- Last but not least important, you must know the physics involved.



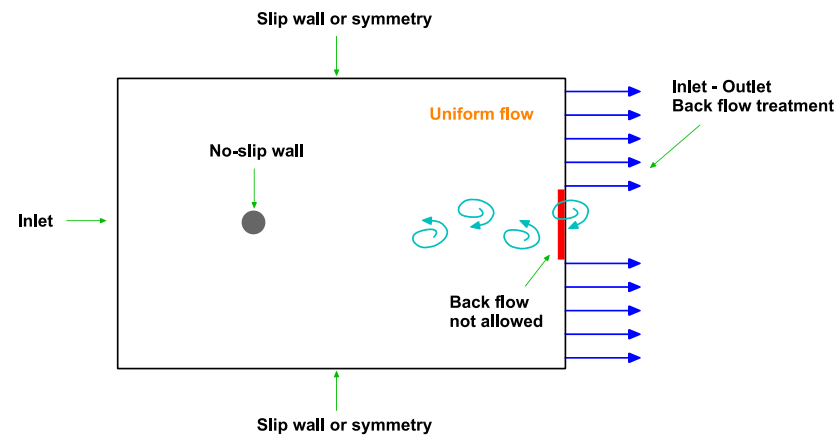
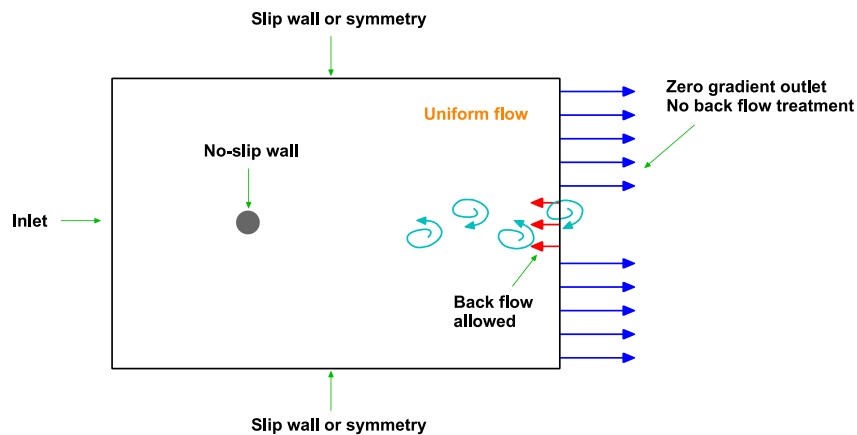
# Boundary conditions and initial conditions

- **Inlets and outlets boundary conditions:**
  - Inlets are for regions where inflow is expected; however, inlets might support outflow when a velocity profile is specified.
  - Pressure boundary conditions do not allow outflow at the inlets.
  - Velocity specified inlets work better with incompressible flows.
  - Pressure and mass flow inlets are suitable for compressible and incompressible flows.
  - Same concepts apply to outlets, which are regions where outflow is expected.



# Boundary conditions and initial conditions

- **Zero gradient and backflow boundary conditions:**
  - Zero gradient boundary conditions extrapolates the values from the domain. They require no information.
  - Zero gradient boundary conditions can be used at inlets, outlets, and walls.
  - Backflow boundary conditions provide a generic outflow/inflow condition, with specified inflow/outflow for the case of backflow.
  - In the case of a backflow outlet, when the flux is positive (out of domain) it applies a Neumann boundary condition (zero gradient), and when the flux is negative (into of domain), it applies a Dirichlet boundary condition (fixed value).
  - Same concept applies to backflow inlets.



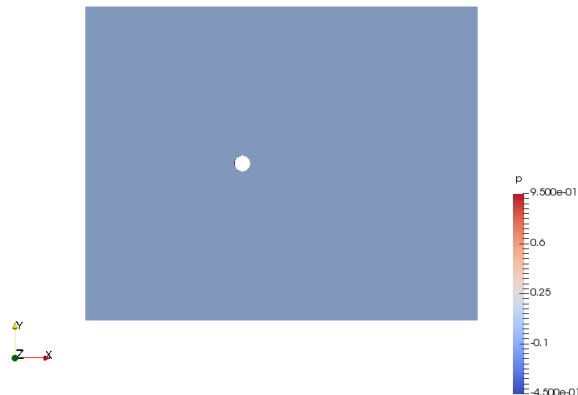
# Boundary conditions and initial conditions

- **On the outlet pressure boundary condition**
  - Some combinations of boundary conditions are very stable, and some are less reliable.
  - And some combinations of boundary conditions are unreliable, *e.g.*,
    - Inlet velocity at the inlet and pressure zero gradient at the outlet. This combination should be avoided because the static pressure level is not fixed.
  - Qualitatively speaking, the results are very different.
  - This simulation will eventually crash.



BCs 1. Inlet velocity and fixed outlet pressure  
[www.wolfdynamics.com/wiki/BC/aniBC1.gif](http://www.wolfdynamics.com/wiki/BC/aniBC1.gif)

Time: 0.000000



BCs 2. Inlet velocity and zero gradient outlet pressure  
[www.wolfdynamics.com/wiki/BC/aniBC2.gif](http://www.wolfdynamics.com/wiki/BC/aniBC2.gif)

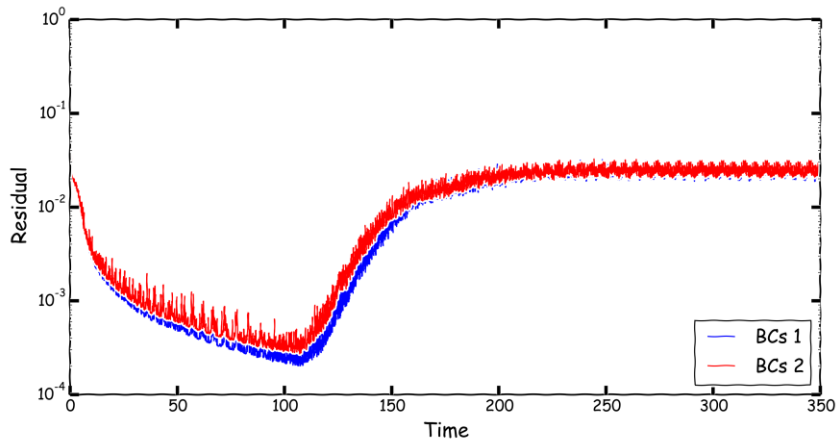
Time: 0.000000



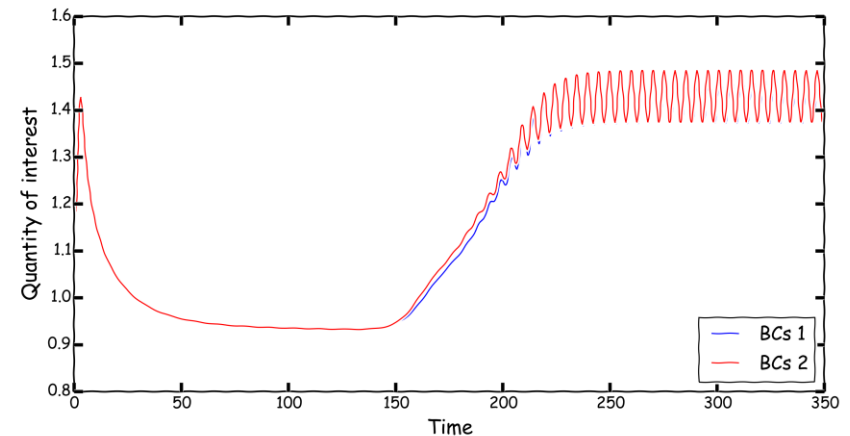
# Boundary conditions and initial conditions

- **On the outlet pressure boundary condition**
  - If you only rely on a QOI and the residuals, you will not see any major difference between the two cases with different outlet pressure boundary condition.
  - This is very misleading.
  - However, when you visualize the solution, you will realize that something is wrong. This is a case where pretty pictures can be used to troubleshoot the solution.
  - Quantitative speaking, the results are very similar.
  - However, this simulation will eventually crash.

Residual plot for pressure

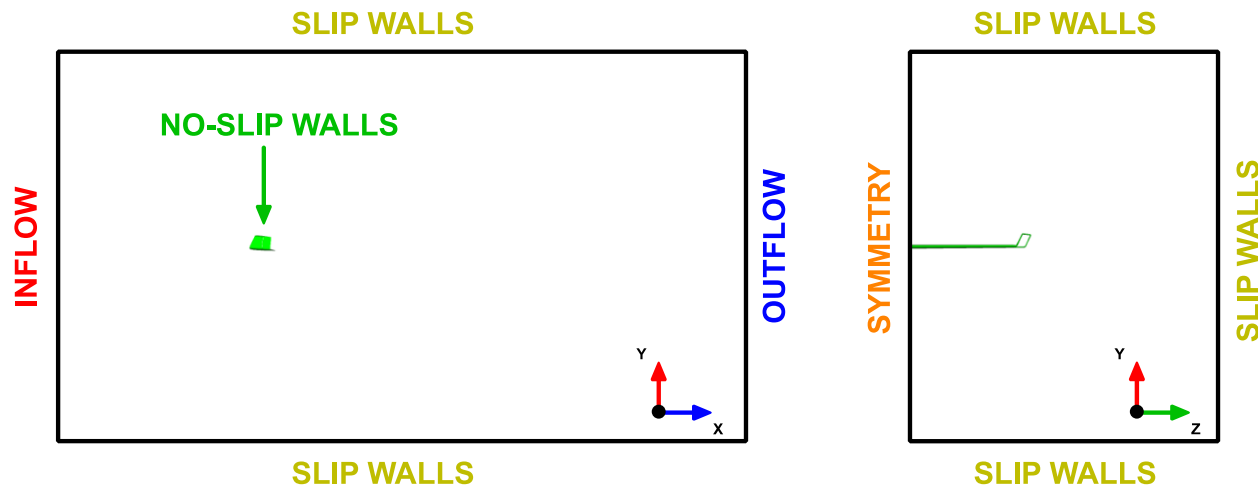


Quantity of interest – Force coefficient on the body



# Boundary conditions and initial conditions

- **Symmetry boundary conditions:**
  - Symmetry boundary conditions are a big simplification of the problem. However, they help to reduce mesh cell count.
  - Have in mind that symmetry boundary conditions only apply to **planar faces**.
  - To use symmetry boundary conditions, both the geometry and the flow field must be symmetric.
  - Mathematically speaking, setting a symmetry boundary condition is equivalent to:
    - Zero normal velocity at the symmetry plane and zero normal gradients of all variables at the symmetry plane.
  - Physically speaking, they are equivalent to slip walls.



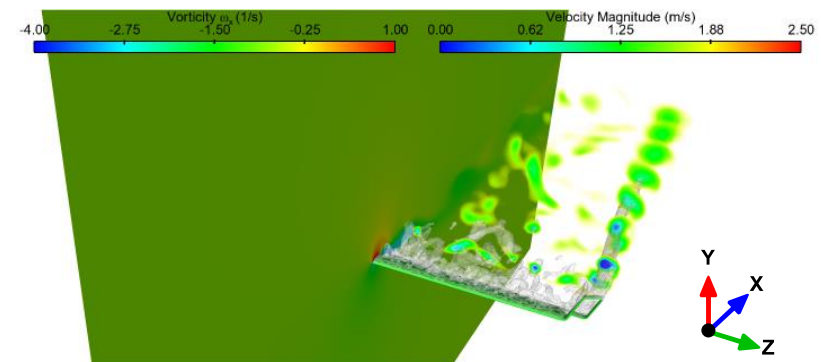
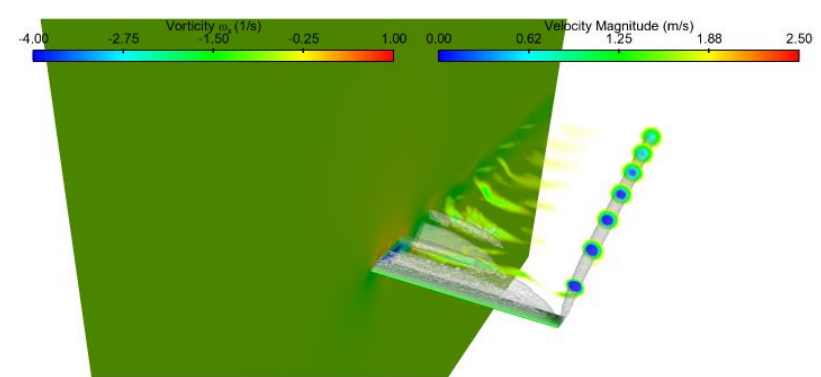
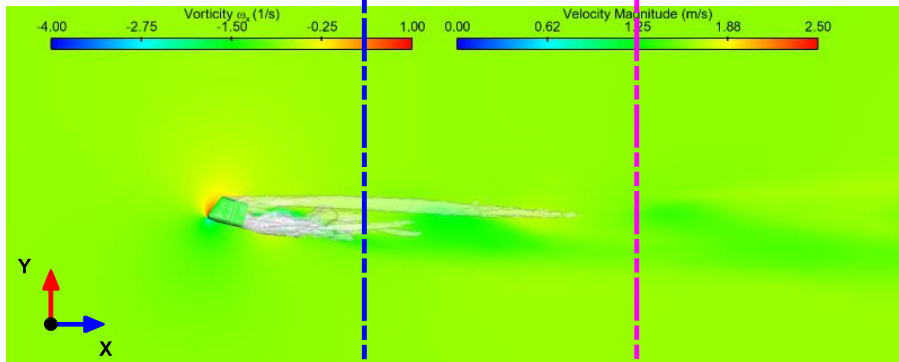
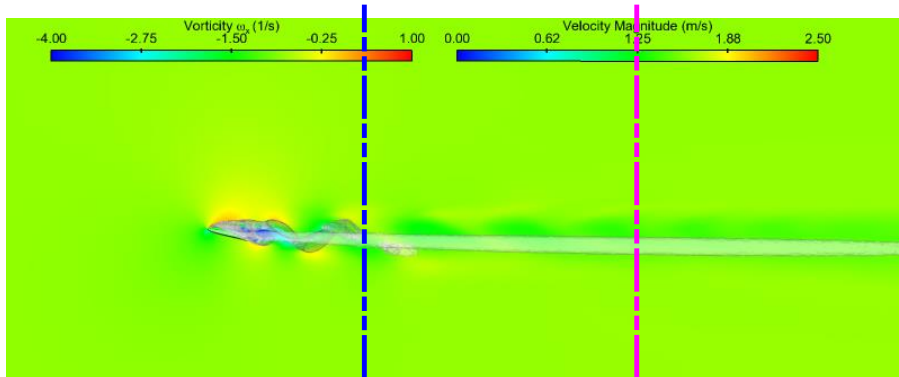
# Boundary conditions and initial conditions

- **Location of the outlet boundary condition:**
  - Place outlet boundary conditions as far as possible from recirculation zones or backflow conditions, by doing this you increase the stability.
  - Remember, backflow conditions requires special treatment.

Possible backflow

Might be OK

Far enough so the flow can be considered fully developed

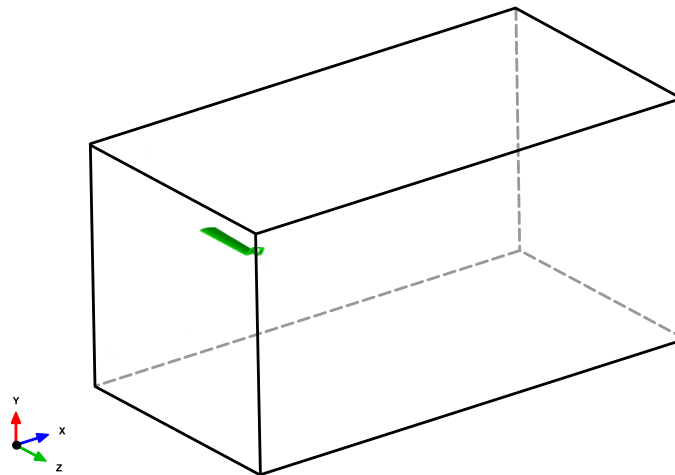
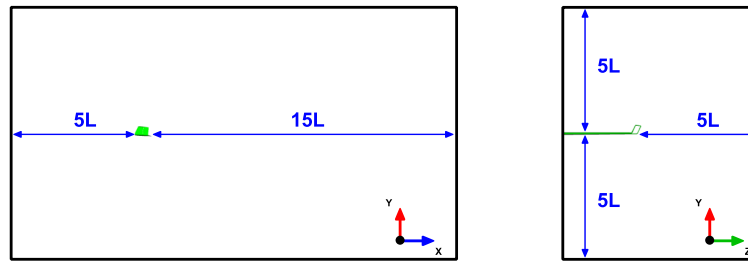




# Boundary conditions and initial conditions

- **Domain dimensions (when the dimensions are not known):**

- If you do not have any constrain in the domain dimensions, you can use as a general guideline the dimensions illustrated in the figure below, where  $L$  is a reference length (in this case,  $L$  is the wing chord).
- The values illustrated in the figure are on the conservative side, but if you want to play safe, multiply the values by two or three.
- Always verify that there are no significant gradients normal to any of the boundary patches. If there are, you should consider increasing the domain dimensions.



# Boundary conditions and initial conditions

## A few considerations and guidelines

- Boundary conditions and initial conditions need to be physically realistic.
- Poorly defined boundary conditions can have a significant impact on your solution.
- Initial conditions are as important as the boundary conditions.
- A good initial condition can improve the stability and convergence rate.
- On the other hand, unphysical initial conditions can slow down the convergence rate or can cause divergence.
  - And in some situations, can lead to a different solution. Very misleading.
- You need to define boundary conditions and initials conditions for every single variable you are solving.
- Setting the right boundary conditions is extremely important.
  - But have in mind that you need to understand the physics in order to set the right boundary conditions.
- Do not force the flow at the outlet, use a zero normal gradient for all flow variables and fix the pressure level.
  - The solver extrapolates the required information from the interior.

# Boundary conditions and initial conditions

## A few considerations and guidelines

- Be careful with backward flow at the outlets (flow coming back to the domain) and backward flow at inlets (reflection waves), they required special treatment.
- If possible, select inflow and outflow boundary conditions such that the flow either goes in or out normal to the boundaries.
- At outlets, use zero gradient boundary conditions only with incompressible flows and when you are sure that the flow is fully developed.
  - Remember, do not use zero gradient for the pressure.
- Outlets that discharge to the atmosphere can use a static pressure boundary condition.
  - This is interpreted as the static pressure of the environment into which the flow exhausts.
- Generally speaking, it is better to set the mass flow or total pressure at the outlet.
- Inlets that take flow into the domain from the atmosphere can use a total pressure boundary condition (e.g., open window).
- Mass flow inlets produce a uniform velocity profile at the inlet.

# Boundary conditions and initial conditions

## A few considerations and guidelines

- Pressure specified boundary conditions at inlets allow a natural velocity profile to develop.
- The required values of the boundary conditions and initial conditions depend on the equations you are solving, and physical models used, *e.g.*,
  - For incompressible and laminar flows, you will need to set only the velocity and pressure.
  - If you are solving a turbulent compressible flow you will need to set velocity, pressure, temperature and the turbulent variables.
  - For multiphase flows you will need to set the primitives variables for each phase. You will also need to initialize the phases.
  - If you are doing turbulent combustion or chemical reactions, you will need to define the species, reactions and turbulent variables.
- Minimize grid skewness, non-orthogonality, growth rate, and aspect ratio near the boundaries. You do not want to introduce diffusion errors early in the simulation, especially close to the inlets.
- Try to avoid large gradients in the direction normal to the boundaries and near inlets and outlets. That is to say, put your boundaries far away from where things are happening.

# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- ~~2. Important concepts to remember~~
- ~~3. The Finite Volume Method: An overview~~
- ~~4. Navier-Stokes equations and pressure-velocity coupling~~
- ~~5. On the CFL number~~
- ~~6. Unsteady and steady simulations~~
- ~~7. Understanding the residuals~~
- ~~8. Boundary conditions and initial conditions~~
- 9. The FVM in OpenFOAM: some implementation details and computational pointers**
- ~~10. Best standard practices – General guidelines~~
- ~~11. Final remarks~~

# The FVM in OpenFOAM

- We just addressed the theoretical background of the FVM method.
- We have seen that a lot of sophistication goes into mesh data bookkeeping, especially for unstructured meshes.
- We know that there are many space and time discretization schemes.
- We also know that at the end of the day we want a solution that is at least second order accurate.
- Also, as we are solving an initial boundary value problem (IBVP), we need to give boundary conditions and initial conditions (that needs to be realistic).
- Then, somehow OpenFOAM assembles a matrix of coefficients.
- And then it will crunch numbers using the linear solvers.
- So, the big question is: what does OpenFOAM do? And where can I choose the different options?
- We are going to review the whole process in this section.

# The FVM in OpenFOAM

## So, what does OpenFOAM do?

- It simply discretizes in space and time the governing equations in arbitrary unstructured meshes.
- Assembling in this way a large set of linear algebraic equations (LAE).
- It then solves this system of LAE using iterative linear solvers to find the solution of the transported quantities.
- Therefore, we need to give to OpenFOAM the following information:
  - Discretization of the solution domain or the mesh. This information is contained in the directory **constant/polyMesh**
  - Boundary conditions and initials conditions. This information is contained in the directory **0**
  - Physical properties such as density, gravity, diffusion coefficient, viscosity, etc. This information is contained in the directory **constant**
  - Physics involved, such as turbulence modeling, mass transfer, source terms, etc. This information is contained in the directories **constant** and/or **system**
  - How to discretize in space each term of the governing equations (diffusive, convective, gradient and source terms). This information is set in the *system/fvSchemes* dictionary.

# The FVM in OpenFOAM

## So, what does OpenFOAM do?

- It simply discretizes in space and time the governing equations in arbitrary unstructured meshes.
- Assembling in this way a large set of linear algebraic equations (LAE).
- It then solves this system of LAE using iterative linear solvers to find the solution of the transported quantities.
- Therefore, we need to give to OpenFOAM the following information:
  - How to discretize in time the semi-discrete governing equations. This information is set in the *system/fvSchemes* dictionary.
  - How to solve the linear system of linear algebraic equations (crunch numbers). This information is set in the *system/fvSolution* dictionary.
  - Set runtime parameters and general instructions on how to run the case (such as time step and maximum CFL number). This information is set in the *system/controlDict* dictionary.
  - Additionally, we may set sampling and monitors for post-processing. This information is set in the *system/fvSchemes* dictionary or in the sampling dictionaries located in the directory **system/**



# The FVM in OpenFOAM

## Are there default options in OpenFOAM?

- When you use commercial CFD applications, they will use the best possible options or default options (stable and accurate).
- Even if you choose the wrong options, the solver will do some black magic to stabilize the solution and get the best results.
- In OpenFOAM, such default options do not exist.
- It is to the user to choose the best options based on the theory.
  - Therefore, it is important to understand the theory.
- Hereafter, we are going to give you what we think are the best options.
- Which are based on what is found in commercial software, extensive validation, and experience.
- A small warning, do not take the options in the tutorials that come with OpenFOAM as the default or best options.
- If you go through the tutorials, you will realize that some of them uses upwind or do not do any kind of correction.
  - Remember, those tutorials are there just to show you how to setup a case.

## **Discretization methods**

# The FVM in OpenFOAM

## Where do we set all the discretization schemes in OpenFOAM?

`ddtSchemes` ←  $\frac{\partial \phi}{\partial t}$   
`{`  
`default backward;`  
`}`

`gradSchemes` ←  $\nabla \phi_P$   
`{`  
`default Gauss linear;`  
`grad(p) Gauss linear;`  
`}`

`divSchemes` ←  $\nabla \cdot (\mathbf{U} \phi)$   
`{`  
`default none;`  
`div(phi,U) Gauss linear;`  
`}`

`laplacianSchemes` ←  $\nabla \cdot \Gamma \nabla \phi$   
`{`  
`default Gauss linear orthogonal;`  
`}`

`interpolationSchemes` ←  $\begin{cases} \phi_f = f_x \phi_P + (1 - f_x) \phi_N \\ f_x = \frac{fN}{PN} = \frac{|\mathbf{x}_f - \mathbf{x}_N|}{|\mathbf{d}|} \end{cases}$   
`{`  
`default linear;`  
`}`

`snGradSchemes`  
`{`  
`default orthogonal; ←  $\mathbf{n}_f \cdot \nabla \phi_f$`   
`}`

- The *fvSchemes* dictionary contains the information related to the discretization schemes for the different terms appearing in the governing equations.
- The discretization schemes can be chosen in a term-by-term basis.
- The keyword **ddtSchemes** refers to the time discretization.
- The keyword **gradSchemes** refers to the gradient term discretization.
- The keyword **divSchemes** refers to the convective terms discretization.
- The keyword **laplacianSchemes** refers to the Laplacian terms discretization.
- The keyword **interpolationSchemes** refers to the method used to interpolate values from cell centers to face centers. It is unlikely that you will need to use something different from linear.
- The keyword **snGradSchemes** refers to the discretization of the surface normal gradients evaluated at the faces.
- Remember, if you want to know the options available for each keyword you can use the banana method.

# The FVM in OpenFOAM

## Time discretization schemes

- These are the time discretization schemes available in OpenFOAM:
  - **backward**
  - **bounded**
  - **CoEuler**
  - **CrankNicolson**
  - **Euler**
  - **localEuler**
  - **SLTS**
  - **steadyState**
- You will find the source code in the following directory:
  - `$WM_PROJECT_DIR/src/finiteVolume/finiteVolume/ddtSchemes`

# The FVM in OpenFOAM

## Time discretization schemes

- These are the time discretization schemes that you will use most of the times:
  - **steadyState**: for steady state simulations (implicit/explicit).
  - **Euler**: time dependent first order (implicit/explicit), bounded.
  - **backward**: time dependent second order (implicit), bounded/unbounded.
  - **CrankNicolson**: time dependent second order (implicit), bounded/unbounded.
- First order methods are bounded and stable, but diffusive.
- Second order methods are accurate, but they might become oscillatory.
- At the end of the day, we always want a second order accurate solution.
- If you keep the CFL less than one when using the Euler method, numerical diffusion is not that much.

# The FVM in OpenFOAM

## Time discretization schemes

- The **Crank-Nicolson** method as it is implemented in OpenFOAM, uses a blending factor.

```
ddtSchemes
{
    default      CrankNicolson  $\psi$ ;
}
```

- Setting  $\psi$  to 0 is equivalent to running a pure **Euler** scheme (robust but first order accurate).
- By setting the blending factor equal to 1 you use a pure **Crank-Nicolson** (accurate but oscillatory, formally second order accurate).
- If you set the blending factor to 0.5, you get something in between first order accuracy and second order accuracy, or in other words, you get the best of both worlds.
- A blending factor of 0.7-0.9 is safe to use for most applications (stable and accurate).

# The FVM in OpenFOAM

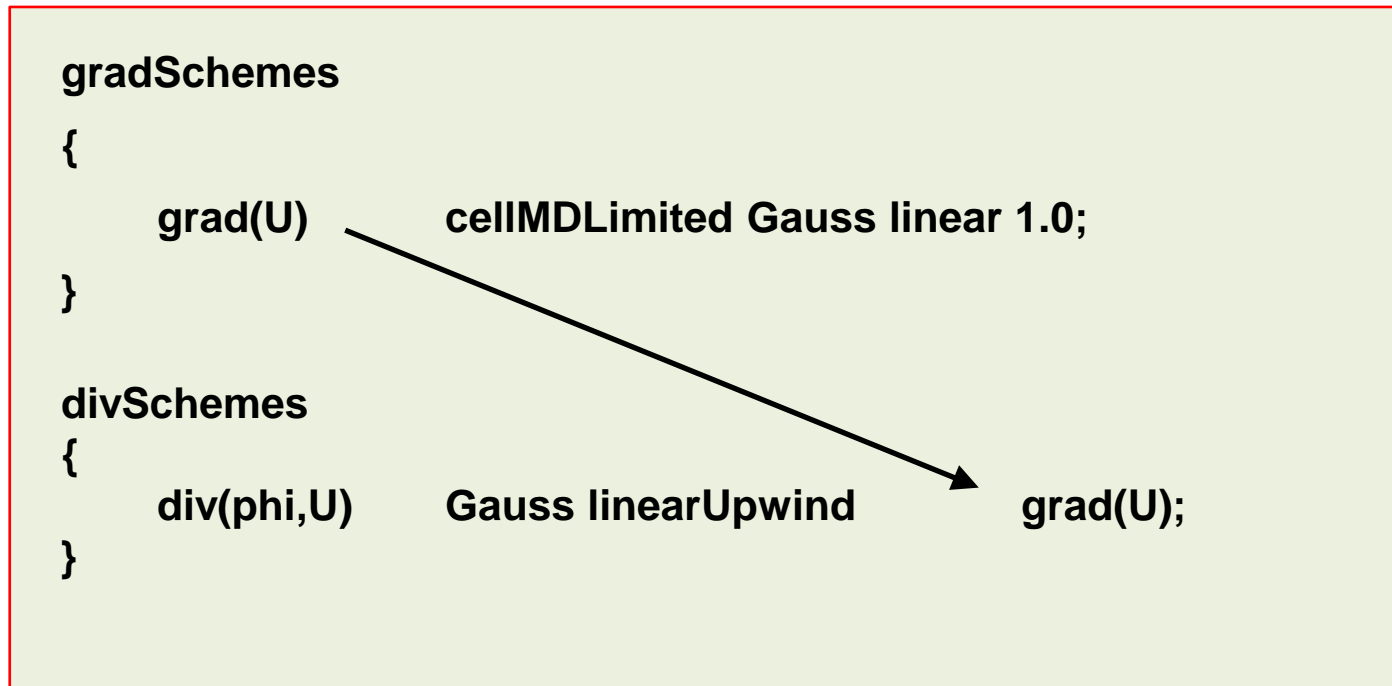
## Convective terms discretization schemes

- There are many convective terms discretization schemes available in OpenFOAM (more than 50 last time we checked).
- You will find the source code in the following directory:
  - `$WM_PROJECT_DIR/src/finiteVolume/interpolation/surfaceInterpolation`
- These are the convective discretization schemes that you will use most of the times:
  - **upwind**: first order accurate.
  - **linearUpwind**: second order accurate, bounded.
  - **linearUpwindV**: second order accurate, bounded, formulation for vector fields.
  - **linear**: second order accurate, unbounded.
  - **vanLeer**: TVD, second order accurate, bounded.
  - **Minmod**: TVD, second order accurate, bounded (alternative to **vanLeer**).
  - **limitedLinear**: second order accurate, unbounded, but more stable than pure linear. Recommended for LES simulations (kind of similar to the Fromm method).
  - **LUST**: blended 75% **linear** and 25% **linearUpwind** scheme.
- First order methods are bounded and stable but diffusive.
- Second order methods are accurate, but they might become oscillatory.
- At the end of the day, we always want a second order accurate solution.

# The FVM in OpenFOAM

## Convective terms discretization schemes

- When you use **linearUpwind** and **LUST** for **div(phi,U)**, you need to tell OpenFOAM how to compute the velocity gradient or **grad(U)**,



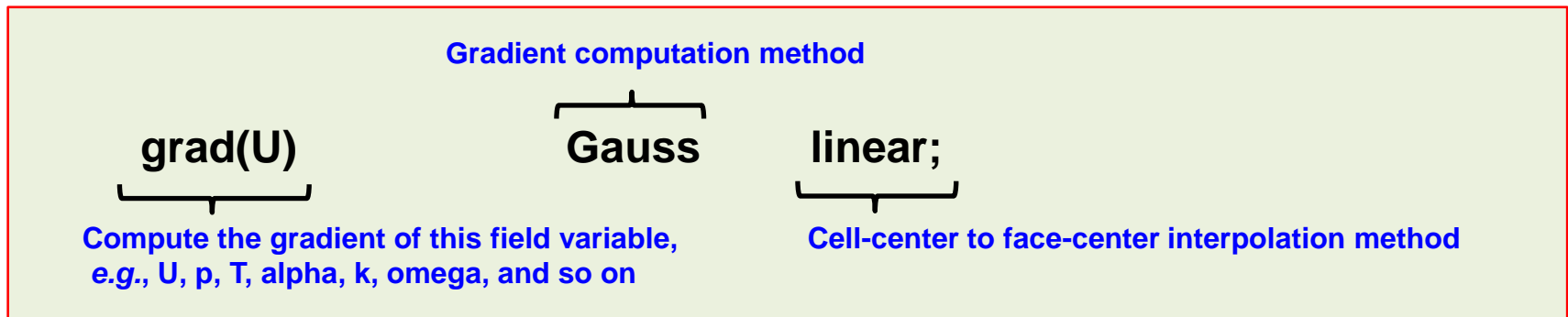
- Same applies for scalars (e.g., **k**, **epsilon**, **omega**, **T**, **e**, **h**) or other vector fields.



# The FVM in OpenFOAM

## Gradient terms discretization schemes

- These are the gradient discretization schemes available in OpenFOAM:
  - **edgeCellsLeastSquares**
  - **fourth**
  - **Gauss**
  - **leastSquares**
  - **pointCellsLeastSquares**
- All of them are at least second order accurate.
- Some of the gradient discretization methods will require information on how to interpolate the cell-centered value to the face-center, *e.g.*,



- You will find the source code in the following directory:
  - `$WM_PROJECT_DIR/src/finiteVolume/finiteVolume/gradSchemes`

# The FVM in OpenFOAM

## Gradient terms discretization schemes

- These are the gradient limiter schemes available in OpenFOAM:
  - **cellLimited**
  - **cellMDLimited**
  - **faceLimited**
  - **faceMDLimited**
- Gradient limiters will avoid over and under shoots on the gradient computations. This increases the stability of the method but will add diffusion due to clipping.
- You will find the source code in the following directory:
  - `$WM_PROJECT_DIR/src/finiteVolume/finiteVolume/gradSchemes/limitedGradSchemes`

# The FVM in OpenFOAM

## Gradient terms discretization schemes

- Additionally, you have the option to change the gradient limiter method.
- The following options are available:
  - **cubic**
  - **minmod**
  - **Venkatakrishnan**
- The default method is the **minmod**.
- You can use the **cubic** or **Venkatakrishnan** method only with the **cellLimited** option.
- You will find the source code in the following directory:
  - `$WM_PROJECT_DIR/src/finiteVolume/finiteVolume/gradSchemes/limitedGradSchemes/cellLimitedGrad/gradientLimiters`

# The FVM in OpenFOAM

## Gradient terms discretization schemes

- Additionally, you have the option to change the gradient limiter method.
- The following options are available:
  - **cubic**
  - **minmod**
  - **Venkatakrishnan**
- To use the **cubic** method, you need to define the following keyword:
  - **cellLimited<cubic>**
- To use the **Venkatakrishnan** method you need to define the following keyword:
  - **cellLimited<Venkatakrishnan>**
- Recall that the **cubic** and **Venkatakrishnan** are differentiable limiters, whereas the **minmod** is non-differentiable.

# The FVM in OpenFOAM

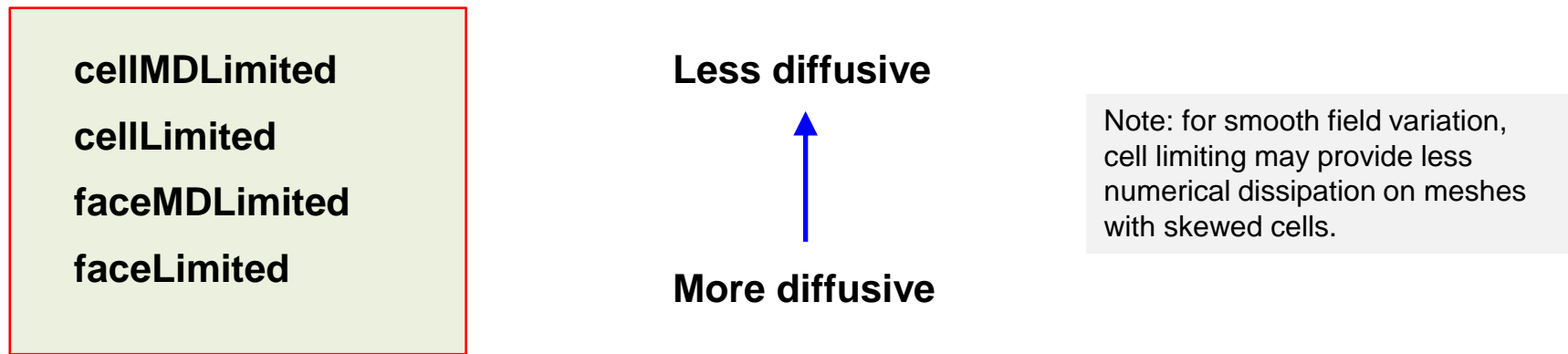
## Gradient terms discretization schemes

- These are the gradient discretization schemes that you will use most of the times:
  - **Gauss + interpolation method**
    - **Gauss linear** (Gauss cell-based)
    - **Gauss pointLinear** (Gauss node-based)
  - **leastSquares** (no interpolation method information required)
- These are the gradient limiter schemes that you will use most of the times:
  - **cellLimited** or **cellMDLimited**
- All of the gradient discretization schemes are at least second order accurate.
- It is recommended not to add too aggressive limiters to all field variables.
- Most of the times is fine to add limiters only for velocity (**U**) and the turbulent quantities (**k**, **omega**, **epsilon**, and so on).
- Avoid adding aggressive limiters to pressure (**p**), temperature (**T**), internal energy (**e**), volume-of-fraction (**alpha**), interface curvature (**nHat**); as they may add too much numerical diffusion.
- If you add too aggressive limiters to all field variables you will add numerical diffusion due to clipping, smear the solution, or stalled the residuals (in steady simulations).

# The FVM in OpenFOAM

## Gradient terms discretization schemes

- According to their diffusivity, the gradient limiter schemes available in OpenFOAM are classified as follows:



- Cell limiters will limit cell-to-cell values.
- Face limiters will limit cell-to-face values.
- The multi-directional (or multi-dimensional) limiters (**cellMDLimited** and **faceMDLimited**), will apply the limiter in each face direction separately (that is, only in the unbounded direction).
- The standard limiters (**cellLimited** and **faceLimited**), will apply the limiter to all components of the gradient.
- The default method is the Minmod.

# The FVM in OpenFOAM

## Gradient terms discretization schemes

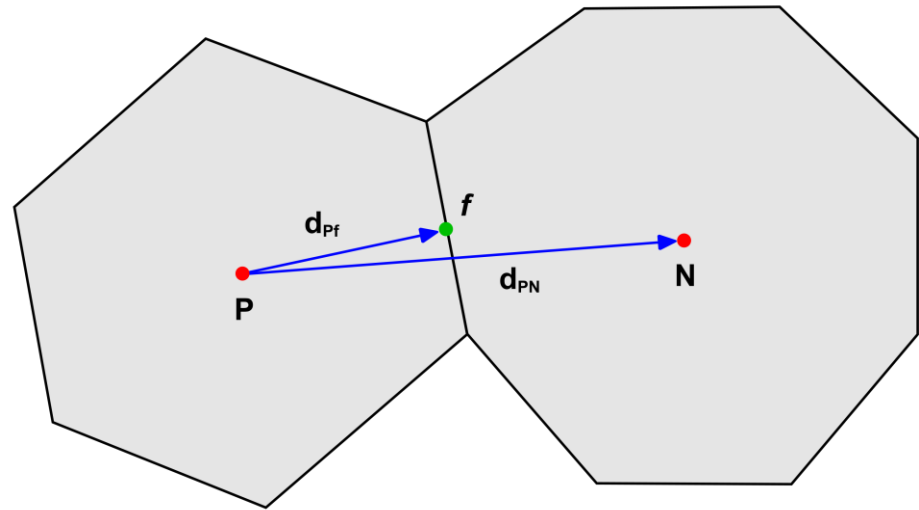
- Limiting direction:

- Cell-to-cell direction limiting,

$$\nabla \phi_P \cdot \mathbf{d}_{PN}$$

- Cell-to-face direction limiting,

$$\nabla \phi_P \cdot \mathbf{d}_{Pf}$$

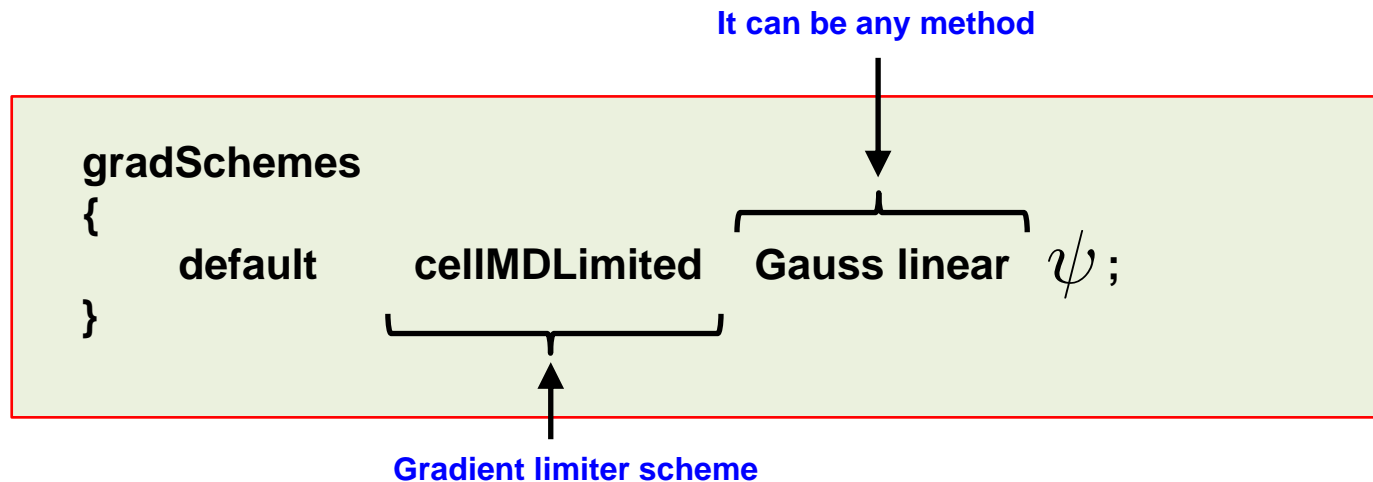


- Cell based limiters will limit cell-to-cell values. That is, in the direction  $\mathbf{d}_{PN}$ .
- Face based limiters will limit cell-to-face values. That is, in the direction  $\mathbf{d}_{Pf}$ .
- The more skewed the mesh is, the bigger the different between these methods.
- In good quality meshes both limiters will give 2<sup>nd</sup> order accuracy. However, in highly skewed meshes the face limiters might give 1<sup>st</sup> order accuracy.
- The method should be selected based in accuracy, smooth field variation, and the need of unnecessary limiting.

# The FVM in OpenFOAM

## Gradient terms discretization schemes

- The gradient limiter implementation in OpenFOAM, uses a blending factor  $\psi$



- Setting  $\psi$  to 0 is equivalent to turning off the gradient limiter. You gain accuracy but the solution might become unbounded.
- By setting the blending factor equal to 1 the limiter is always on. You gain stability but you give up accuracy (due to gradient clipping).
- If you set the blending factor to 0.5, you get the best of both worlds.
- You can use limiters with all gradient discretization schemes.



# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- These are the Laplacian terms discretization schemes available in OpenFOAM:
  - **corrected**
  - **faceCorrected**
  - **limited**
  - **linearFit**
  - **orthogonal**
  - **quadraticFit**
  - **uncorrected**
- You will find the source code in the following directory:
  - `$WM_PROJECT_DIR/src/finiteVolume/finiteVolume/snGradSchemes`

# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- These are the Laplacian terms discretization schemes that you will use most of the times:

- orthogonal:** mainly limited for hexahedral meshes with no grading (a perfect mesh). Second order accurate, bounded on perfect meshes, without non-orthogonal corrections.
- corrected:** for meshes with grading and non-orthogonality. Second order accurate, bounded depending on the quality of the mesh, with non-orthogonal corrections.
- limited:** for meshes with grading and non-orthogonality. Second order accurate, bounded depending on the quality of the mesh, with non-orthogonal corrections.
- uncorrected:** usually limited to hexahedral meshes with very low non-orthogonality. Second order accurate, without non-orthogonal corrections. Stable but more diffusive than the limited and corrected methods.

$$\longrightarrow \mathbf{S} \cdot (\nabla \phi)_f = |\mathbf{S}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}.$$

Can be computed using the over-relaxed approach

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}.$$

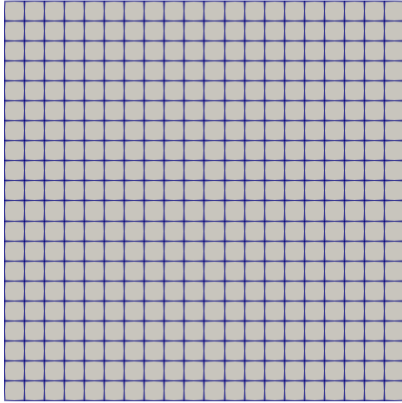
$$\longrightarrow \mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}.$$

Can be computed using the over-relaxed approach

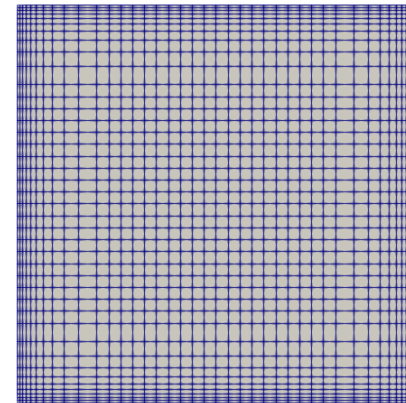
# The FVM in OpenFOAM

## Laplacian terms discretization schemes

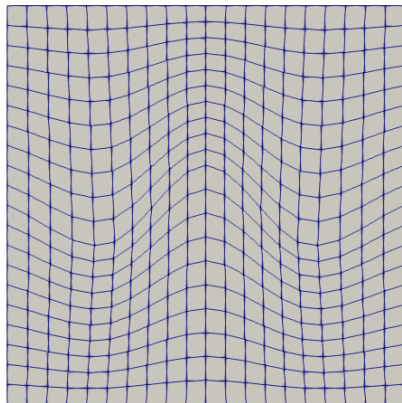
- According to the mesh, the Laplacian discretization can be chosen as follows:



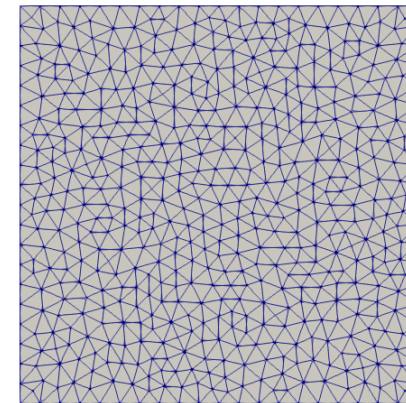
Perfect orthogonal mesh with no stretching  
**laplacianSchemes** → **orthogonal**



Orthogonal mesh with stretching  
**laplacianSchemes** → **limited 1** or **corrected**



Mesh with some degree of non-orthogonality (low to medium)  
**laplacianSchemes** → **limited 1** to **limited 0.5**

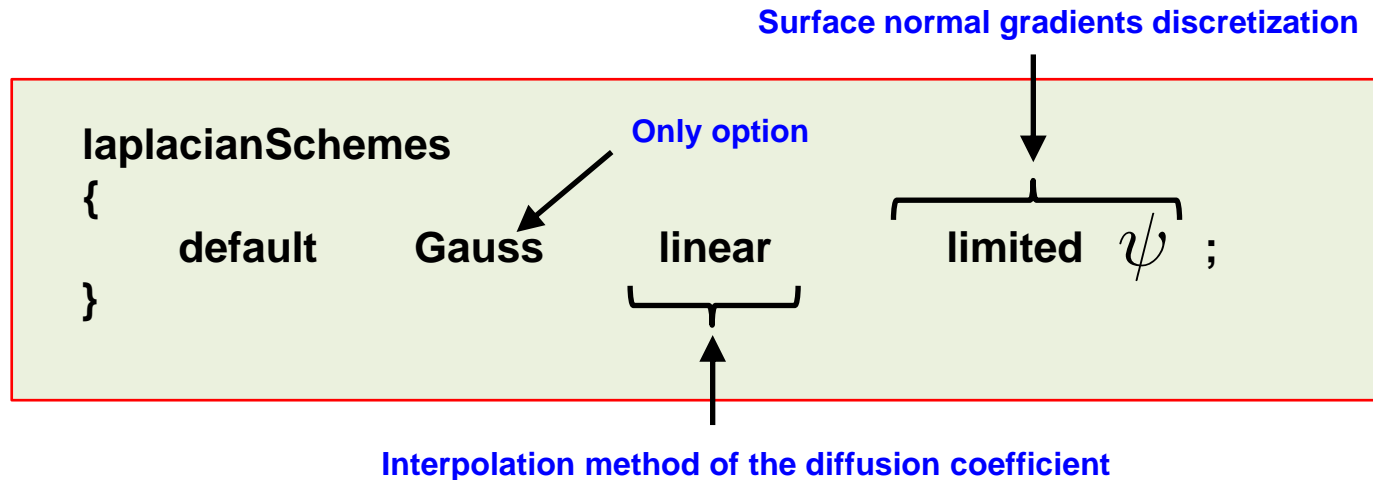


General unstructured meshes  
**laplacianSchemes** → **limited 0.5**

# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- The limited method uses a blending factor  $\psi$ .

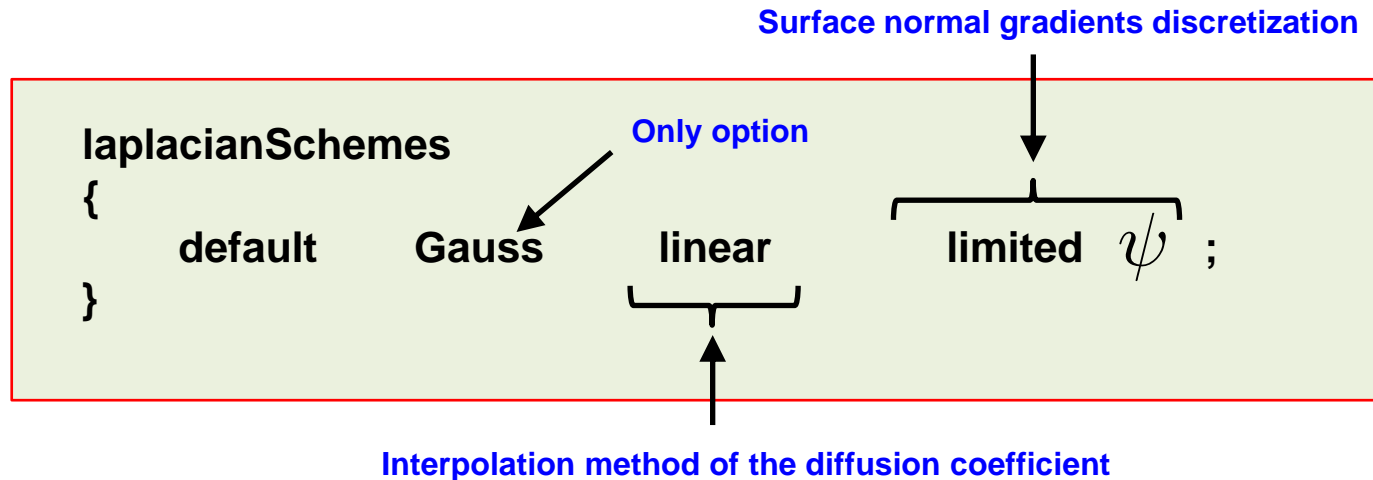


- Setting  $\psi$  to 1 is equivalent to using the **corrected** method.
  - The computation of the Laplacian on the non-orthogonal mesh depends on the orthogonal contribution (implicit contribution), and on the non-orthogonal contribution (explicit contribution).
  - You gain accuracy, but the solution might become unbounded.
- This approach is recommended for meshes with non-orthogonality less than 70 degrees.

# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- The limited method uses a blending factor  $\psi$ .

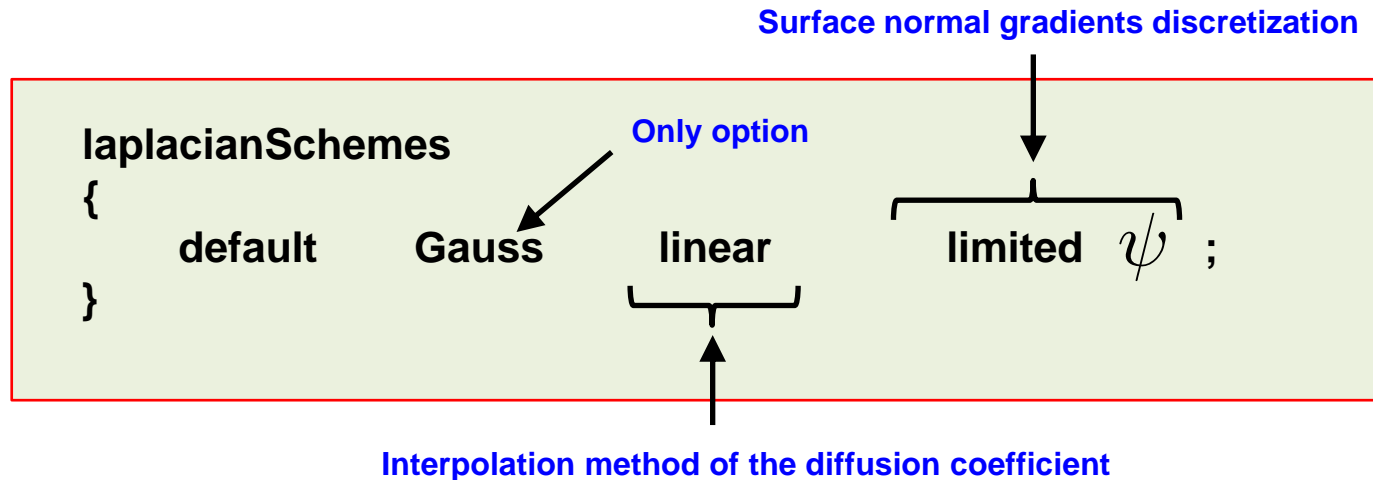


- Setting  $\psi$  to 0 is equivalent to using the **uncorrected** method.
  - The computation of the Laplacian on the non-orthogonal mesh depends only on the orthogonal contribution.
  - You give up accuracy with the potential benefit of gaining some stability.
- This approach is recommended for very good quality meshes.
- That is, meshes with non-orthogonality less than 60 degrees.
- Rarely you will use this approach.

# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- The limited method uses a blending factor  $\psi$ .

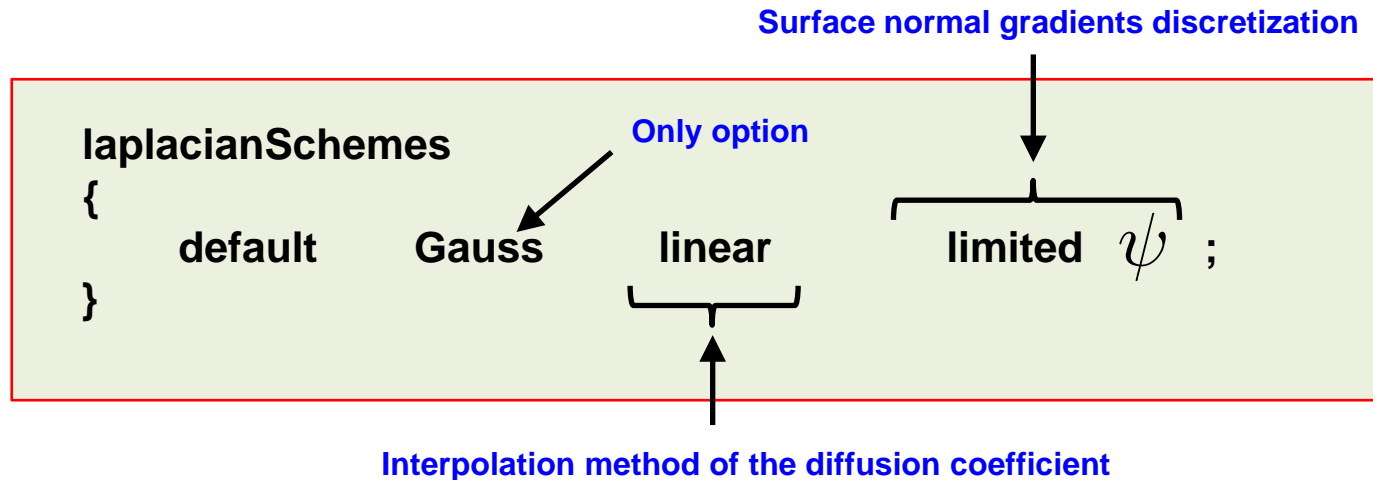


- By setting  $\psi$  to 0.5 you will get the best of both worlds.
  - The computation of the Laplacian on the non-orthogonal mesh depends on the orthogonal contribution (implicit contribution), and on the non-orthogonal contribution (explicit contribution).
  - However, the non-orthogonal contribution is limited so that it does not exceed the orthogonal part.
  - The limiting is proportional to the blending coefficient used.
  - You give up accuracy but gain stability.

# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- The limited method uses a blending factor  $\psi$ .



- Final guidelines:
  - For meshes with non-orthogonality less than 70, you can set the blending factor to 1.
  - For meshes with non-orthogonality between 70 and 85, you can set the blending factor to 0.5
    - This is the method recommended for industrial unstructured meshes.**
  - For meshes with non-orthogonality more than 85, it is better to get a better mesh.
    - But if you want to use that mesh, you can set the blending factor between 0.333 and 0.5.
    - You should also increase the number of non-orthogonal corrections.
  - If you are doing LES or DES simulations, use a blending factor of 1.
    - This means that you need good meshes.

# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- Just to make it clear, the blending factor  $\psi$  is used to avoid the non-orthogonal contribution exceeding the orthogonal part, that is,

$$\text{non-orthogonal contribution} \leq \text{orthogonal contribution}$$

- And recall that by using the over-relaxed approach, the Laplacian term (or diffusive flux) is computed (and corrected) as follows,

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}} .$$

Implicit contribution

Explicit contribution

$$\mathbf{S} = \Delta_{\perp} + \mathbf{k} \longrightarrow \mathbf{k} = \mathbf{S} - \Delta_{\perp} \quad \Delta_{\perp} = \frac{\mathbf{d}}{\mathbf{d} \cdot \mathbf{S}} |\mathbf{S}|^2$$




# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- Just to make it clear, the blending factor  $\psi$  is used to avoid the non-orthogonal contribution exceeding the orthogonal part, that is, non-orthogonal contribution  $\leq$  orthogonal contribution.

The blending factor works as a limiter acting on this term (non-orthogonal contribution)

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}.$$



Implicit contribution

Explicit contribution

- Then, the amount of correction applied to the non-orthogonal contribution is proportional to the blending coefficient  $\psi$  used in the **limited** approach.


$$\underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}} \leq \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} \times \underbrace{\text{Limiter - Blending coefficient}}_{\psi}$$

# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- **On the need of limiting the non-orthogonal contribution.**
  - In meshes with large non-orthogonality, the explicit term can lead to unboundedness and eventually divergence.
  - To avoid unboundedness, a limiting is applied so that the non-orthogonal contribution never exceeds the orthogonal contribution.
  - This limiting is local, similar to the treatment done for the connective terms when using slope limiters and TVD schemes.
  - The explicit contribution is added to the RHS of the linear system (source term), so if this term becomes too large it will lead to convergence problems.
  - If the non-orthogonal contribution is large, it becomes harder to guarantee diagonal dominance of the matrix of coefficient; therefore, the Scarborough criterion might not be satisfied.

The blending factor works as a limiter acting on this term (non-orthogonal contribution)

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}.$$


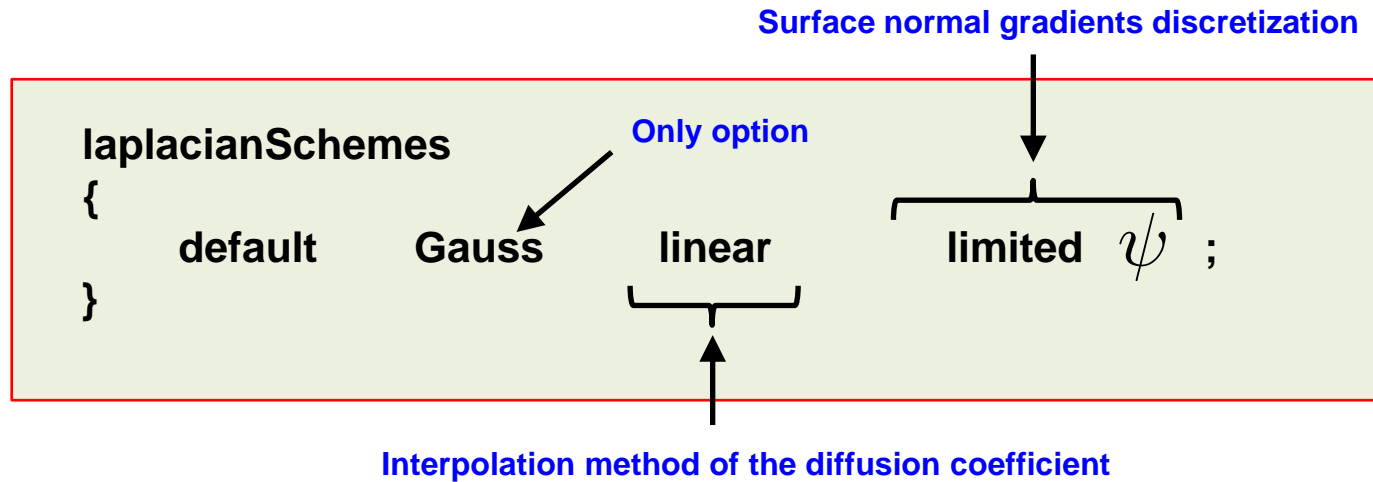
Implicit contribution

Explicit contribution

# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- The limited method uses a blending factor  $\psi$ .



- It is unlikely that you will need to use something different from linear to interpolate the diffusion coefficient.
- If that situation arises (e.g., if you are dealing with CHT where you have different diffusion coefficients in each region), the following options are valid:
  - cubic
  - harmonic
  - linear
  - midPoint
  - pointLinear
  - reverseLinear

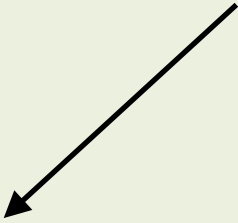
# The FVM in OpenFOAM

## Laplacian terms discretization schemes

- The surface normal gradients terms usually use the same method as the one chosen for the Laplacian terms.
- For instance, if you are using the **limited 1** method for the Laplacian terms, you can use the same method for **snGradSchemes**:

```
laplacianSchemes
{
    default          Gauss linear      limited 1;
}

snGradSchemes
{
    default          limited 1;
}
```




# The FVM in OpenFOAM

**What method should I use?**

# The FVM in OpenFOAM

## Recommended setup for most cases

```
ddtSchemes
{
    default          CrankNicolson 0; //0-0.333
}
gradSchemes
{
    default          cellLimited Gauss linear 0.5;
    grad(U)          cellLimited Gauss linear 1;
}
divSchemes
{
    default          none;
    div(phi,U)       Gauss linearUpwindV grad(U);
    div(phi,omega)    Gauss linearUpwind default;
    div(phi,k)        Gauss linearUpwind default;
    div((nuEff*dev(T(grad(U)))) Gauss linear;
}
laplacianSchemes
{
    default          Gauss linear limited 1;
}
interpolationSchemes
{
    default          linear;
}
snGradSchemes
{
    default          limited 1;
}
```

- **This setup is recommended for most of the cases.** 
- It is very similar to the default method you will find in commercial solvers.
- In overall, this setup is second order accurate and fully bounded.
- According to the quality of your mesh, you will need to change the blending factor of the **laplacianSchemes** and **snGradSchemes** keywords.
- To keep time diffusion to a minimum, use a CFL number less than 2, and preferably below 1.
- If during the simulation the turbulence quantities become unbounded, you can safely change the discretization scheme to upwind. After all, turbulence is diffusion.
- For gradient discretization the **leastSquares** method is more accurate. But we have found that it is a little bit oscillatory in tetrahedral meshes.

# The FVM in OpenFOAM

## Recommended setup for most cases

```
ddtSchemes
{
    default Euler; //0-0.333
}
gradSchemes
{
    default cellLimited Gauss linear 0.5;
    grad(U) cellLimited Gauss linear 1;
}
divSchemes
{
    default none;
    div(phi,U) Gauss linearUpwind grad(U);
    div(phi,omega) Gauss upwind;
    div(phi,k) Gauss upwind;
    div((nuEff*dev(T(grad(U)))) Gauss linear;
}
laplacianSchemes
{
    default Gauss linear limited 0.5;
}
interpolationSchemes
{
    default linear;
}
snGradSchemes
{
    default limited 0.5;
}
```

- And if the previous setup is becoming a little bit oscillatory, you can try this variant.



- In overall, this setup still is second order accurate and fully bounded, except for the turbulence terms.
- However, this is not a big problem because turbulence is a diffusive process.
- To keep time diffusion to a minimum, use a CFL number less than 2, and preferably below 1.

# The FVM in OpenFOAM

## A very accurate but oscillatory numerics

```
ddtSchemes
{
    default backward;
}
gradSchemes
{
    default Gauss leastSquares;
}
divSchemes
{
    default none;
    div(phi,U) Gauss linear;
    div(phi,omega) Gauss linear;
    div(phi,k) Gauss linear;
    div((nuEff*dev(T(grad(U)))) Gauss linear;
}
laplacianSchemes
{
    default Gauss linear limited 1;
}
interpolationSchemes
{
    default linear;
}
snGradSchemes
{
    default limited 1;
}
```

- If you are looking for more accuracy, you can use this method.
- In overall, this setup is second order accurate but oscillatory.
- Use this setup with LES simulations or laminar flows with no complex physics.
- Use this method with good quality meshes.
- According to the quality of your mesh, you will need to change the blending factor of the **laplacianSchemes** and **snGradSchemes** keywords.



# The FVM in OpenFOAM

## An accurate and a little bit more stable numerics

```
ddtSchemes
{
    default          CrankNicolson 0.7;
}
gradSchemes
{
    default          cellMDLimited Gauss linear 0.5;
}
divSchemes
{
    default          none;
    div(phi,U)       Gauss linear;
    div(phi,omega)    Gauss limitedLinear 1;
    div(phi,k)        Gauss limitedLinear 1;
    div((nuEff*dev(T(grad(U)))) Gauss linear;
}
laplacianSchemes
{
    default          Gauss linear limited 1;
}
interpolationSchemes
{
    default          linear;
}
snGradSchemes
{
    default          limited 1;
}
```

- If you are looking for more accuracy, you can use this method.
- In overall, this setup is second order accurate.
- In the presence of strong gradients, it might oscillate a little bit.
- This setup is recommended for cases with no complex physics.
- Use this method with good quality meshes.
- According to the quality of your mesh, you will need to change the blending factor of the **laplacianSchemes** and **snGradSchemes** keywords.

# The FVM in OpenFOAM

## Still accurate (but starting to become diffusive) numerics

```
ddtSchemes
{
    default          CrankNicolson 0.333
}
gradSchemes
{
    default          cellLimited Gauss linear 0.333;
    grad(U)          cellLimited Gauss linear 1;
}
divSchemes
{
    default          none;
    div(phi,U)       Gauss linearUpwindV grad(U);
    div(phi,omega)    Gauss linearUpwind default;
    div(phi,k)        Gauss linearUpwind default;
    div((nuEff*dev(T(grad(U)))) Gauss linear;
}
laplacianSchemes
{
    default          Gauss linear limited 1;
}
interpolationSchemes
{
    default          linear;
}
snGradSchemes
{
    default          limited 1;
}
```

- In overall, this setup is second order accurate.
- It is more diffusive than the previous one.
- This setup is partially bounded, we re using gradient limiters only for **U**.
- Here, we are giving up accuracy to gain stability.
- According to the quality of your mesh, you will need to change the blending factor of the **laplacianSchemes** and **snGradSchemes** keywords.

# The FVM in OpenFOAM

## A very stable but too diffusive numerics

```
ddtSchemes
{
    default Euler;
}
gradSchemes
{
    default cellLimited Gauss linear 0.5;
    grad(U) cellLimited Gauss linear 1;
}
divSchemes
{
    default none;
    div(phi,U) Gauss upwind;
    div(phi,omega) Gauss upwind;
    div(phi,k) Gauss upwind;
    div((nuEff*dev(T(grad(U)))) Gauss linear;
}
laplacianSchemes
{
    default Gauss linear limited 0.5;
}
interpolationSchemes
{
    default linear;
}
snGradSchemes
{
    default limited 0.5;
}
```

- If you are looking for extra stability, you can use this method.
- This setup is very stable but too diffusive.
- This setup is first order in space and time.
- You can use this setup to start the solution in the presence of bad quality meshes or strong discontinuities.
- Remember, you can start using a first order method and then switch to a second order method.
- According to the quality of your mesh, you will need to change the blending factor of the **laplacianSchemes** and **snGradSchemes** keywords.
- You can use this method for troubleshooting. If the solution diverges, you better check boundary conditions, physical properties, and so on.
- **Start robustly, end with accuracy.**

# The FVM in OpenFOAM

**Pressure velocity coupling  
SIMPLE and PISO loops**

# The FVM in OpenFOAM

- In OpenFOAM, you will find segregated pressure-based solvers.
- The following methods are available:
  - **SIMPLE** (Semi-Implicit Method for Pressure-Linked Equations)
  - **SIMPLEC** (SIMPLE Corrected/Consistent)
  - **PISO** (Pressure Implicit with Splitting Operators)
- Additionally, you will find something called **PIMPLE**, which is a hybrid between **SIMPLE** and **PISO**.
  - Also known as iterative PISO or **PISO-ITA**.
- The **PISO-ITA** formulation can give you more accuracy and stability when using very large time-steps, pseudo-transient simulations, or when dealing with complex physics.
- The standard **PISO**, is also known as **PISO-NITA** or non-iterative **PISO**.

# The FVM in OpenFOAM

- In OpenFOAM, the **PISO** and **PIMPLE** methods are formulated for unsteady simulations.
- Whereas the **SIMPLE** and **SIMPLEC** methods are formulated for steady simulations.
- If conserving time is not a priority, you can use the **PIMPLE** method in pseudo transient mode.
- The pseudo transient **PIMPLE** method is more stable than the **SIMPLE** method, but it has a higher computational cost.
- Depending on the method and solver you are using, you will need to define a specific sub-dictionary in the dictionary file *fvSolution*.
- For instance, if you are using the **PISO** method, you will need to specify the **PISO** sub-dictionary.
- And depending on the method, each sub-dictionary will have different entries.
- You will find the solvers in the following directory:
  - **\$WM\_PROJECT\_DIR/applications/solvers**

# The FVM in OpenFOAM

- As already seen, the **SIMPLE** and **PISO** methods are used to deal with the coupling of the pressure and velocity equations (P-V coupling).
- In the original governing equations, the continuity and momentum equations are decoupled, that is, there is no direct pressure link.
- Therefore, we need to use some mathematical tricks to deal with this decoupling.
- In the **SIMPLE** and **PISO** methods, we mathematically manipulate the starting equations, so the pressure now appears in both equations.
  - In this way there is a direct link between the governing equations.
- It is worth stressing that the manipulated equations are equivalent to the original equations.
  - There is no loss of generality.
- When using the **SIMPLE** and **PISO** methods, we use the velocity obtained in the momentum equation to compute the pressure using the newly derived pressure equation, and then correct the velocity with the new pressure value.

# The FVM in OpenFOAM

## On the origins of the methods – Useful references

- **SIMPLE**

- S. V. Patankar and D. B. Spalding, “A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows”, Int. J. Heat Mass Transfer, 15, 1787-1806 (1972).

- **SIMPLEC or SIMPLE consistent**

- J. P. Van Doormaal and G. D. Raithby, “Enhancements of the SIMPLE method for predicting incompressible fluid flows”, Numerical Heat Transfer, 7, 147-163 (1984).

- **PISO**

- R. I. Issa, “Solution of the implicitly discretized fluid flow equations by operator-splitting”, J. Comput. Phys., 62, 40-65 (1985).



# The FVM in OpenFOAM

## On the origins of the methods – Useful references

- **PIMPLE**

- Unknown origins outside OpenFOAM ecosystem (we are referring to the semantics).
- It is equivalent to **PISO** with outer iterations (iterative time-advancement of the solution).
- Useful reference (besides **PISO** reference):
  - I. E. Barton, “Comparison of SIMPLE and PISO-type algorithms for transient flows, Int. J. Numerical methods in fluids, 26,459-483 (1998).
  - P. Oliveira and R. I. Issa, “An improved piso algorithm for the computation of buoyancy-driven flows”, Numerical Heat Transfer, 40, 473-493 (2001).

- **Rhie-Chow interpolation**

- C. M. Rhie and W. L. Chow, “Numerical study of the turbulent flow past an airfoil with trailing edge separation”, AIAA Journal, Vol. 21, 1525-1532 (1983).

# The FVM in OpenFOAM

## Equations used in the SIMPLE and PISO loops

- The pressure equation is derived starting from the momentum equation,

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nu \nabla^2 \mathbf{U} = -\nabla p$$

- Then, by taking the divergence of the momentum equation and setting  $\nabla \cdot \mathbf{U} = 0$ , we obtain,

$$\cancel{\nabla \cdot \left( \frac{\partial \mathbf{U}}{\partial t} \right)} + \nabla \cdot (\nabla \cdot (\mathbf{U}\mathbf{U})) - \cancel{\nabla \cdot (\nu \nabla^2 \mathbf{U})} = -\nabla^2 p$$

- Then, the final form of the pressure equation is as follows,

$$\nabla^2 p = \nabla \cdot \mathbf{H}(\mathbf{U}) \quad \text{where} \quad \mathbf{H}(\mathbf{U}) = -\nabla \cdot (\mathbf{U}\mathbf{U})$$

# The FVM in OpenFOAM

## Equations used in the SIMPLE and PISO loops

- In the pressure-based approach, the actual equations that are being solved are,

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nu \nabla^2 \mathbf{U} = -\nabla p$$

This system of equations is equivalent to the original Navier-Stokes equations.

$$\nabla^2 p = \nabla \cdot \mathbf{H}(\mathbf{U}) \quad \text{where} \quad \mathbf{H}(\mathbf{U}) = -\nabla \cdot (\mathbf{U}\mathbf{U})$$

- The previous equations are solved in a given domain, with boundary conditions **BCs**, and initial condition **ICs**.
- In this set of equations, continuity  $\nabla \cdot \mathbf{U} = 0$  is enforced while deriving the pressure equation (referred to as pressure-Poisson equation) and in all boundaries of the domain.
- We use these equations because in the original incompressible Navier-Stokes equations, pressure does not appear in the continuity equation, so is not possible to link the equations.
- Therefore, we derive an alternative set of equations where pressure appears (albeit in the form of a gradient and large pressure gradients may cause stability and accuracy problems).
- So now we can use the velocity obtained in the momentum equation (momentum predictor step) to compute the pressure using the pressure-Poisson equation (pressure corrector step), and then correct the velocity with the new pressure value (momentum corrector step).
- This is referred to as pressure-velocity coupling (P-V coupling).

# The FVM in OpenFOAM

## Equations used in OpenFOAM SIMPLE and PISO loops

- The equations used in the loops implemented in OpenFOAM are divided by  $A$ .
- The matrix  $A$  contains the diagonal coefficients of the momentum equations corresponding to the **SIMPLE** or **PISO** loops.
- By dividing by  $A$ , makes the equations more convergent.
- In the momentum equation, add and subtract the term  $A\mathbf{U}$ ,

$$\underbrace{\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nu \nabla^2 \mathbf{U} - A\mathbf{U} + A\mathbf{U}}_{-\mathbf{H}(\mathbf{U}) \text{ contains all terms except } \nabla p} = -\nabla p$$

- Then, divide by  $A$ , take divergence and apply  $\nabla \cdot \mathbf{U} = 0$

$$-\nabla \cdot \left( \frac{\mathbf{H}(\mathbf{U})}{A} \right) + \cancel{\nabla \cdot \mathbf{U}}^0 = -\nabla \cdot \frac{1}{A} \nabla p$$

# The FVM in OpenFOAM

## Equations used in OpenFOAM SIMPLE and PISO loops

- Then, the pressure equation is expressed as follows (pay attention that is divided by  $A$ ),

$$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left( \frac{\mathbf{H}(\mathbf{U})}{A} \right)$$

- The momentum corrector (also divided by  $A$ ), is expressed as follows,

$$\mathbf{U} = \frac{\mathbf{H}(\mathbf{U})}{A} - \frac{1}{A} \nabla p$$

- Notice that the momentum corrector equation is obtained from equation,

$$\underbrace{\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nu \nabla^2 \mathbf{U} - A\mathbf{U} + A\mathbf{U}}_{-\mathbf{H}(\mathbf{U}) \text{ contains all terms except } \nabla p} = -\nabla p$$

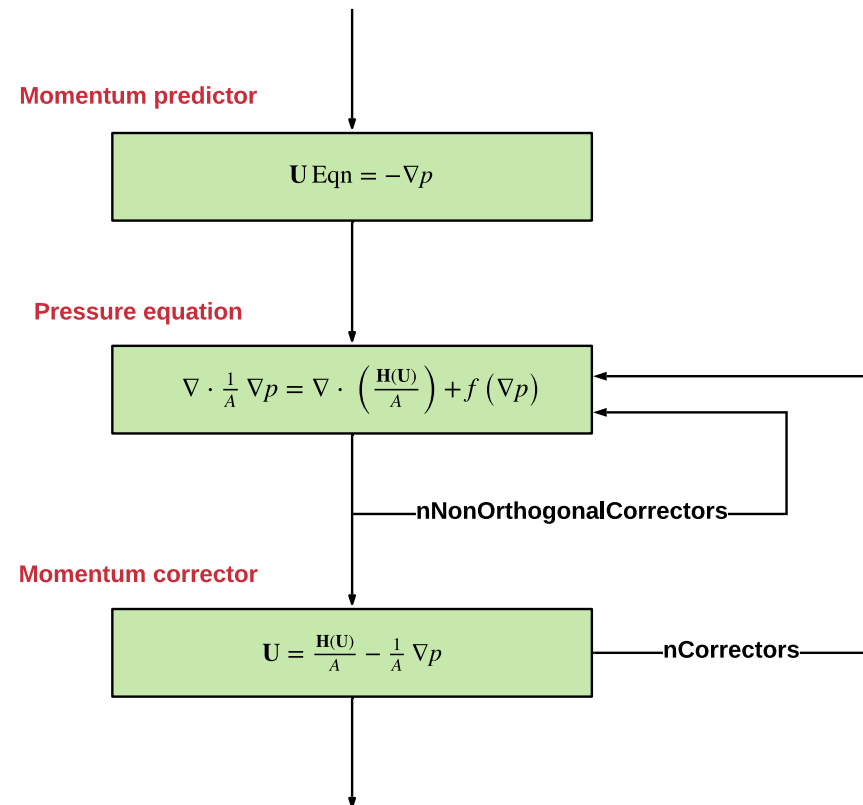
# The FVM in OpenFOAM

## Equations used in OpenFOAM SIMPLE and PISO loops

- As we already know, mesh non-orthogonality introduces secondary gradients into the pressure equation (the term  $f(\nabla p)$  in the equation).
- To reduce any error introduced by secondary gradients, we need to correct the pressure equation.
- That is, we solve for pressure and then we correct it, obtaining in this way better approximations.
- This is controlled using the **nNonOrthogonalCorrectors** keyword.
- After correcting  $\mathbf{U}$  (momentum corrector), we can substitute the new value in the pressure equation and solve again.
- This is controlled by the keyword **nCorrectors** (total passes through pressure and momentum corrector equations).
- Notice that mesh non-orthogonality and skewness also introduces secondary gradients in the energy equation,  $f(\nabla T)$ .
- The energy equation tends to be more sensitive to secondary gradients than the pressure equation. Therefore, is recommended to do more corrections.

$$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left( \frac{\mathbf{H}(\mathbf{U})}{A} \right) + f(\nabla p)$$

$$\mathbf{H}(\mathbf{U}) = -\nabla \cdot (\mathbf{U}\mathbf{U})$$



# The FVM in OpenFOAM

## The SIMPLE sub-dictionary

- This sub-dictionary is located in the dictionary file *fvSolution*.
- It controls the options related to the **SIMPLE** pressure-velocity coupling method.
- The **SIMPLE** method only makes one correction.
- An additional correction to account for mesh non-orthogonality is available when using the **SIMPLE** method. The number of non-orthogonal correctors is specified by the **nNonOrthogonalCorrectors** keyword.
- The number of non-orthogonal correctors is chosen according to the mesh quality.
- For orthogonal meshes you can use 0 non-orthogonal corrections. However, it is strongly recommended to do at least 1 non-orthogonal correction (this helps stabilizing the solution).
- For non-orthogonal meshes, it is recommended to do at least 1 correction.

**SIMPLE**

{

→ **nNonOrthogonalCorrectors** 1;

}

# The FVM in OpenFOAM

## The SIMPLE sub-dictionary

- You can use the optional keyword **consistent** to enable or disable the **SIMPLEC** method.
- This option is disabled by default.
- In the **SIMPLEC** method, the cost per iteration is marginally higher but the convergence rate is better so the number of iterations can be reduced.
- The **SIMPLEC** method relaxes the pressure in a consistent manner and additional relaxation of the pressure is not generally necessary.
- In addition, convergence of the **p-U** system is better and still is reliable with less aggressive relaxation of the momentum equation.

**SIMPLE**

{

**consistent**    **yes;**  
**nNonOrthogonalCorrectors**    **1;**

}



# The FVM in OpenFOAM

## The SIMPLE sub-dictionary

- These are the typical (or industry standard) under-relaxation factors for the **SIMPLE** and **SIMPLEC** methods.
- Remember the under-relaxation factors are problem dependent.

### SIMPLE

```
relaxationFactors
{
    fields
    {
        p          0.3;
    }
    equations
    {
        U          0.7;
        k          0.7;
        omega      0.7;
    }
}
```

### SIMPLEC

```
relaxationFactors
{
    fields
    {
        p          0.9;
    }
    equations
    {
        p          0.9;
        U          0.9;
        k          0.9;
        omega      0.9;
    }
}
```

Usually there is no need to under-relax pressure; however, it is advisable.

# The FVM in OpenFOAM

## The SIMPLE sub-dictionary

- If you are planning to use the **SIMPLEC** method, we recommend you use under-relaxation factors that are little bit smaller than the commonly recommended values.
- If during the simulation you still have some stability problems, try to reduce all the values to 0.5.
- Remember the under-relaxation factors are problem dependent.
- It is also recommended to start the simulation with low values (about 0.5), and then increase the values slowly up to 0.7 or 0.9 (for faster convergence).
- For complex physics or when the solution diverges with no reason, set all URF values to 0.7 or even lower.

### SIMPLEC

```
relaxationFactors
{
    fields
    {
        p          0.7;
    }
    equations
    {
        p          0.7;
        U          0.7;
        k          0.7;
        omega      0.7;
    }
}
```

# The FVM in OpenFOAM

## The SIMPLE sub-dictionary

- The **SIMPLE** and **SIMPLEC** methods require the definition of under-relaxation factors (URF).
- The URF control the change of the field variables from iteration to iteration.
- If you do not define URF they will be switch-off, therefore, you will not use under-relaxation and is likely that the solution will diverge.
- In OpenFOAM, setting the URF values equal to 1 is not equivalent to turning them off.
- URF values equal to one will make the linear system more diagonally dominant.
- To know what field variables can be under-relaxed, go to the solver directory or model directory (e.g., turbulence models), and type in the terminal:
  - `$> grep -rn "relax()"`
- All variables reported by this command requires the definition of under-relaxation factors.
- So far we have addressed the **SIMPLE** method. Have in mind that you can also use URF with the **PISO** and **PIMPLE** methods.
- However, if time accuracy is important to you, set all URF equal to 1 or add URF that are not too low (usually industry standard values are fine).
- If you under-relax transient solvers, it is strongly recommended to conduct a time convergence study by using different URF and time-steps values to be sure that you are not losing time accuracy.

# The FVM in OpenFOAM

## The SIMPLE loop in OpenFOAM

```
fvVectorMatrix UEqn
```

```
(
    fvm::ddt(U) + fvm::div(phi, U) - fvm::laplacian(nu, U)
);
```

```
solve(UEqn == -fvc::grad(p));
```

```
fvScalarMatrix pEqn
```

```
(
    fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
);
```

```
U = HbyA - rAU*fvc::grad(p);
```

This is an excerpt of the actual source code of the solver

Momentum equation without the pressure gradient term

Under-relax U Eqn

Momentum predictor

Pressure equation

Under-relax p

Update flux  $\phi = S_f \cdot \left[ \frac{H(U)_f}{A_f} - \left( \frac{1}{A} \right)_f (\nabla p)_f \right]$

Momentum corrector

Solve additional transport equations

Start  
Iterative marching

U Eqn

$\partial U / \partial t + \nabla \cdot (UU) - \nu \nabla^2 U$

U Eqn =  $-\nabla p$

$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left( \frac{H(U)}{A} \right) + f(\nabla p)$

nNonOrthogonalCorrectors

Non-orthogonal corrections loop

$U = \frac{H(U)}{A} - \frac{1}{A} \nabla p$

SIMPLE loop  
convergence?

No

SIMPLE loop

Yes

End  
Iterative marching

Iterative marching

# The FVM in OpenFOAM

## The PISO sub-dictionary

- This sub-dictionary is located in the dictionary file *fvSolution*.
- It controls the options related to the **PISO** pressure-velocity coupling method.
- The **PISO** method requires at least one correction (**nCorrectors**).
- For good accuracy and stability (specially in unstructured meshes), it is recommended to use at least 2 **nCorrectors**.
- An additional correction to account for mesh non-orthogonality is available when using the **PISO** method. The number of non-orthogonal correctors is specified by the **nNonOrthogonalCorrectors** keyword.
- The number of non-orthogonal correctors is chosen according to the mesh quality.
- For orthogonal meshes you can use 0 non-orthogonal corrections. However, it is strongly recommended to do at least 1 non-orthogonal correction (this helps stabilizing the solution).
- For non-orthogonal meshes, it is recommended to do at least 1 correction.

**PISO**

{

nCorrectors 2;

nNonOrthogonalCorrectors 1;

}

# The FVM in OpenFOAM

## The PISO sub-dictionary

- You can use the optional keyword **momentumPredictor** to enable or disable the momentum predictor step.
- The momentum predictor (**momentumPredictor**) helps in stabilizing the solution as it computes better approximations for the velocity.
- It is clear that this will add an extra computational cost, which most of the times is negligible.
- In most of the solvers, this option is enabled by default.
- It is recommended to use this option for highly convective flows.
  - Flows with high Reynolds number ( $Re > 10000$ ) or with large Peclet numbers ( $Pe > 10$ ).
- If you are working with low Reynolds flows or creeping flows, it is recommended to turn it off.

PISO

{

momentumPredictor yes;

nCorrectors 2;

nNonOrthogonalCorrectors 1;

}

# The FVM in OpenFOAM

## The PISO sub-dictionary

- Note that when you enable the option **momentumPredictor**, you will need to define the linear solvers for the variables **.\*Final** (we are using regex notation).
- You need to set the extra **.\*Final** linear solvers for all transported variables except pressure.
- The pressure **pFinal** linear solver needs to be defined always.
- In our experience, the benefits of the **momentumPredictor** in unsteady solvers are not quite clear.
- And in particular if you are using the **PISO-ITA** approach.
- If you are using the **PISO-NITA** approach, we recommend you use this option at the beginning of the simulation and then turn it off.
  - Remember to always monitor the solution for oscillations when doing these modifications on-the-fly.

PISO

{

momentumPredictor yes;

nCorrectors 2;

nNonOrthogonalCorrectors 1;

}

# The FVM in OpenFOAM

## The PISO loop in OpenFOAM (PISO with non-iterative marching – NITA – )

- It is recommended to switch-off the momentum predictor option for creeping flows or low convection flows (low Peclet number).
- If you enable this option in creeping flows or low convection flows, it is recommended to do at least two nCorrectors.

```
fvVectorMatrix UEqn
(
    fvm::ddt(U) + fvm::div(phi, U) - fvm::laplacian(nu, U)
);
```

```
solve(UEqn == -fvc::grad(p));
```

```
fvScalarMatrix pEqn
(
    fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
);
```

```
U = HbyA - rAU*fvc::grad(p);
```

Momentum equation without the pressure gradient term

Under-relax U Eqn

Momentum predictor

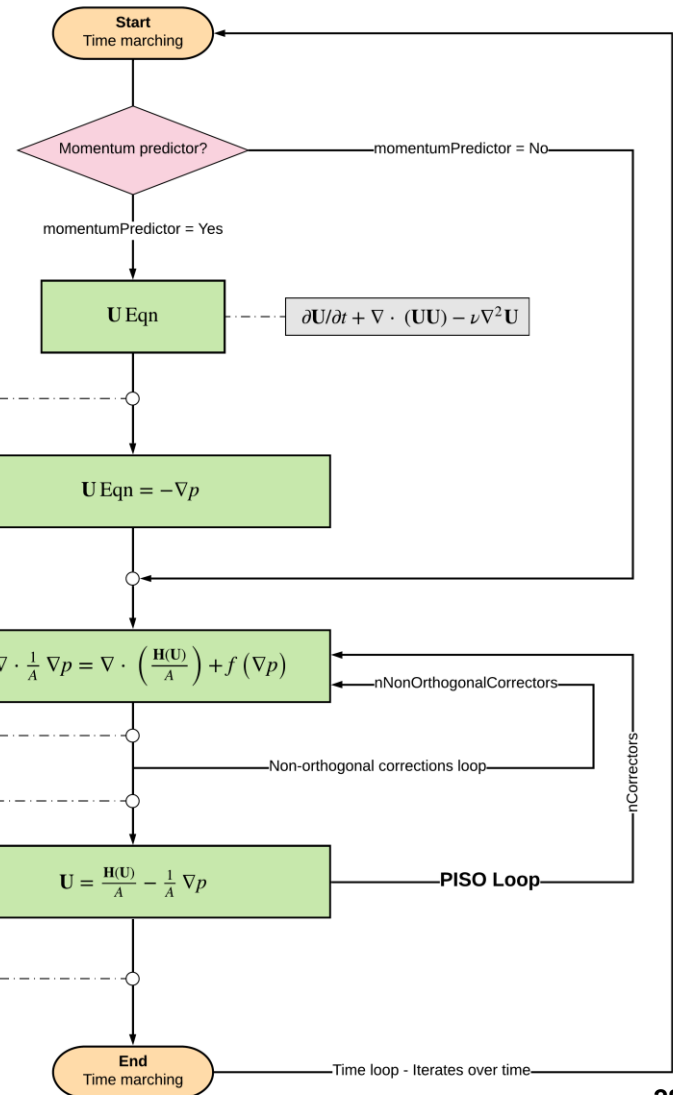
Pressure equation

Under-relax p

Update flux  $\phi = S_f \cdot \left[ (H/A)_f - (1/A)_f (\nabla p)_f \right]$

Momentum corrector

Solve additional transport equations



This is an excerpt of the actual source code of the solver



# The FVM in OpenFOAM

## The PIMPLE sub-dictionary

- This sub-dictionary is located in the dictionary file *fvSolution*. It controls the options related to the **PIMPLE** pressure-velocity coupling method.
- The **PIMPLE** method works very similar to the **PISO** method.
- In fact, setting the keyword **nOuterCorrectors** to 1 is equivalent to running using the **PISO** method.
- The keyword **nOuterCorrectors** controls a loop outside the **PISO** loop.
- To gain more stability, especially when using large time-steps or when dealing with complex physics (combustion, chemical reactions, shock waves, and so on), you can use more outer correctors (**nOuterCorrectors**).
  - Usually between 2 and 5 corrections for computational efficiency.
- Have in mind that increasing the number of **nOuterCorrectors** will highly increase the computational cost.

### PIMPLE

{

momentumPredictor yes;

nOuterCorrectors 1;

nCorrectors 2;

nNonOrthogonalCorrectors 1;

}

# The FVM in OpenFOAM

## The PIMPLE sub-dictionary

- You can use the optional keyword **momentumPredictor** to enable or disable the momentum predictor step.
- The momentum predictor (**momentumPredictor**) helps in stabilizing the solution as it computes better approximations for the velocity.
- It is clear that this will add an extra computational cost, which most of the times is negligible.
- In most of the solvers, this option is enabled by default.
- It is recommended to use this option for highly convective flows.
  - Flows with high Reynolds number ( $Re > 10000$ ) or with large Peclet numbers ( $Pe > 10$ ).
- If you are working with low Reynolds flows or creeping flows, it is recommended to turn it off.
- If you enable this option when working with low Reynolds flow or creeping flows, it is recommended to do at least two nCorrectors.

### PIMPLE

{

momentumPredictor yes;

nOuterCorrectors 1;

nCorrectors 2;

nNonOrthogonalCorrectors 1;

}

# The FVM in OpenFOAM

## The PIMPLE sub-dictionary

- Note that when you enable the option **momentumPredictor**, you will need to define the linear solvers for the variables **.\*Final** (we are using regex notation).
- You need to set the extra **.\*Final** linear solvers for all transported variables except pressure.
- The pressure **pFinal** linear solver needs to be defined always.
- In our experience, the benefits of switching-off the **momentumPredictor** in unsteady solvers are not quite clear.
- And in particular if you are using the **PISO-ITA** approach (**PIMPLE** in OpenFOAM).
- With the **PIMPLE** family of solver, we recommended to always switch it on and to do at least two **nOuterCorrectors**.

**PIMPLE**

{

**momentumPredictor**    **yes;**

**nOuterCorrectors**    **2;**

**nCorrectors**    **2;**

**nNonOrthogonalCorrectors**    **1;**

}

# The FVM in OpenFOAM

## The PIMPLE sub-dictionary

- You can use under-relaxation factors (URF) with the **PISO** family of solvers, namely,
  - **PISO-NITA** and **PISO-ITA** (**PIMPLE** in OpenFOAM).
- By using URF, you will gain more stability in time dependent solutions (as they control the amount of change of field variables within the time-step).
- However, if you use too low URF values, your solution might not be time-accurate anymore.
- You can use the same or larger URF values as those for steady simulation.
- We recommend to always use URF factors.
- In particular, if you are using the **PIMPLE** family of solvers with large CFL numbers (large time-steps).

**PIMPLE**

{

momentumPredictor yes;

nOuterCorrectors 1;

nCorrectors 2;

nNonOrthogonalCorrectors 1;

}

# The FVM in OpenFOAM

## The PIMPLE sub-dictionary

- You can use the following guidelines to define the URF with the **PIMPLE** family of solvers:

```
relaxationFactors
{
    //Nothing in here
}
```

- URF switch-off.

```
relaxationFactors
{
    fields
    {
        "*" 1.0;
    }
    equations
    {
        "*" 1.0;
    }
}
```

- URF set to ensure diagonally dominance.
- The wildcard `*` will apply the URF factors to all fields (including `*Final`).

# The FVM in OpenFOAM

## The PIMPLE sub-dictionary

- You can use the following guidelines to define the URF with the **PIMPLE** family of solvers:

```
relaxationFactors
{
    fields
    {
        "p.*"          0.3;
    }
    equations
    {
        "U.*"          0.7;
        "k.*"          0.7;
        "omega.*"      0.7;
    }
}
```

- Recommended URF values with the **PIMPLE** method (**SIMPLE** formulation).
- The wildcard .\* will apply the URF factors to all fields (including .\*Final).

```
relaxationFactors
{
    fields
    {
        "p.*"          0.7;
    }
    equations
    {
        "p.*"          0.7;
        "U.*"          0.7;
        "k.*"          0.7;
        "omega.*"      0.7;
    }
}
```

- Recommended URF values with the **PIMPLE** method (**SIMPLEC** formulation).
- The wildcard .\* will apply the URF factors to all fields (including .\*Final).

# The FVM in OpenFOAM

## The PIMPLE sub-dictionary

- You can use the following guidelines to define the URF with the **PIMPLE** family of solvers:

```
relaxationFactors
{
    fields
    {
        p                0.7;
        pFinal           1.0;
    }
    equations
    {
        U                0.7;
        UFinal           1.0;
        k                0.7;
        kFinal           1.0;
        omega            0.7;
        omegaFinal       1.0;
    }
}
```

- You can also apply the URF in a selective way.
- That is, you can use different URF for the intermediate field variables (*i.e.*, **p**, **U**, and so on) and the final field variables (*i.e.*, **pFinal**, **UFinal**, and so on).
- It is not compulsory to use URF with transient solvers.
- Nonetheless, using URF with transient solvers will improve stability as they increase diagonal dominance.
- Remember, if you use too low URF you will lose time accuracy.
- Therefore, it is strongly recommended to conduct a time convergence study by using different URF and time-steps values.
- In general, we recommend to set all URF equal to 1 (to improve stability) or using the industry standard (for extra stability).

# The FVM in OpenFOAM

## The PIMPLE loop in OpenFOAM (PISO with iterative marching – ITA –)

- It is recommended to switch-off the momentum predictor option for creeping flows or low convection flows (low Peclet number).
- If you enable this option in creeping flows or low convection flows, it is recommended to do at least two nCorrectors.

```
fvVectorMatrix UEqn
```

```
(
    fvm::ddt(U) + fvm::div(phi, U) - fvm::laplacian(nu, U)
);
```

```
solve(UEqn == -fvc::grad(p));
```

```
fvScalarMatrix pEqn
```

```
(
    fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
);
```

```
U = HbyA - rAU*fvc::grad(p);
```

This is an excerpt of the actual source code of the solver

Momentum equation without the pressure gradient term

Under-relax U Eqn

Momentum predictor

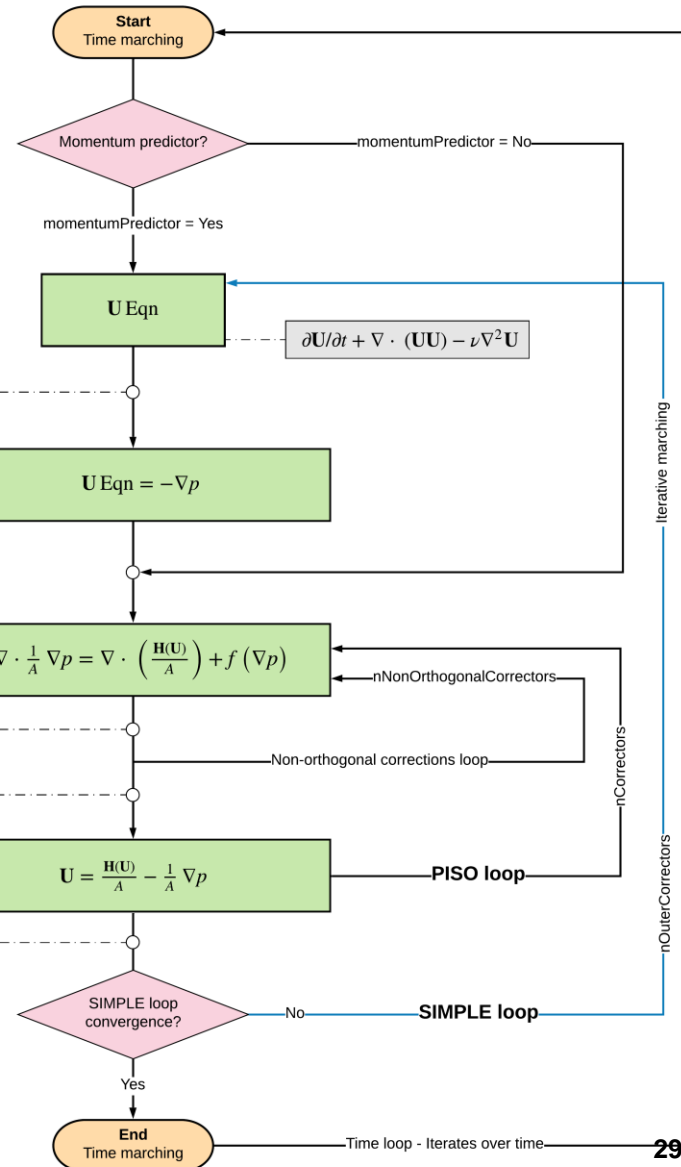
Pressure equation

Under-relax p

Update flux  $\phi = S_f \cdot \left[ \left( \frac{H(U)}{A} \right)_f - \left( \frac{U(A)}{A} \right)_f (\nabla p)_f \right]$

Momentum corrector

Solve additional transport equations

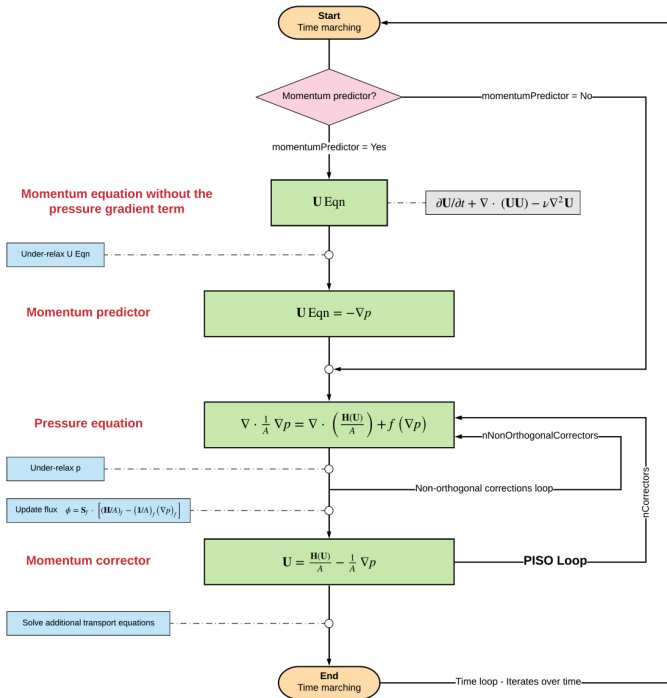




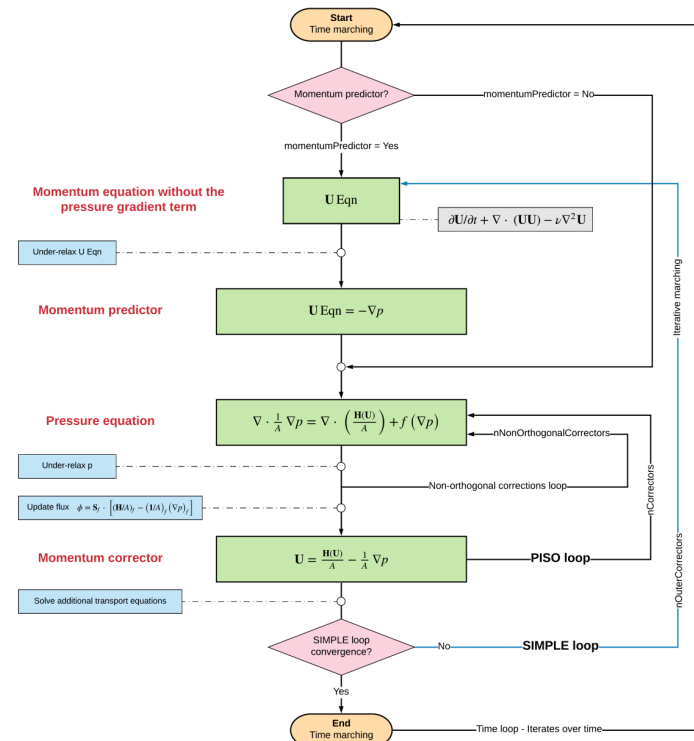
# The FVM in OpenFOAM

## Comparison of PISO with non-iterative time-advancement (PISO-NITA) against PISO with Iterative time-advancement (PISO-ITA)

- The main difference between both methods is the outer loop present in the **PISO-ITA**.
- This outer loop gives more stability and allow the use of very large time-steps (CFL numbers).
- The recommended CFL number of the **PISO-NITA** is below 2 (for good accuracy and stability).



**PISO-NITA**



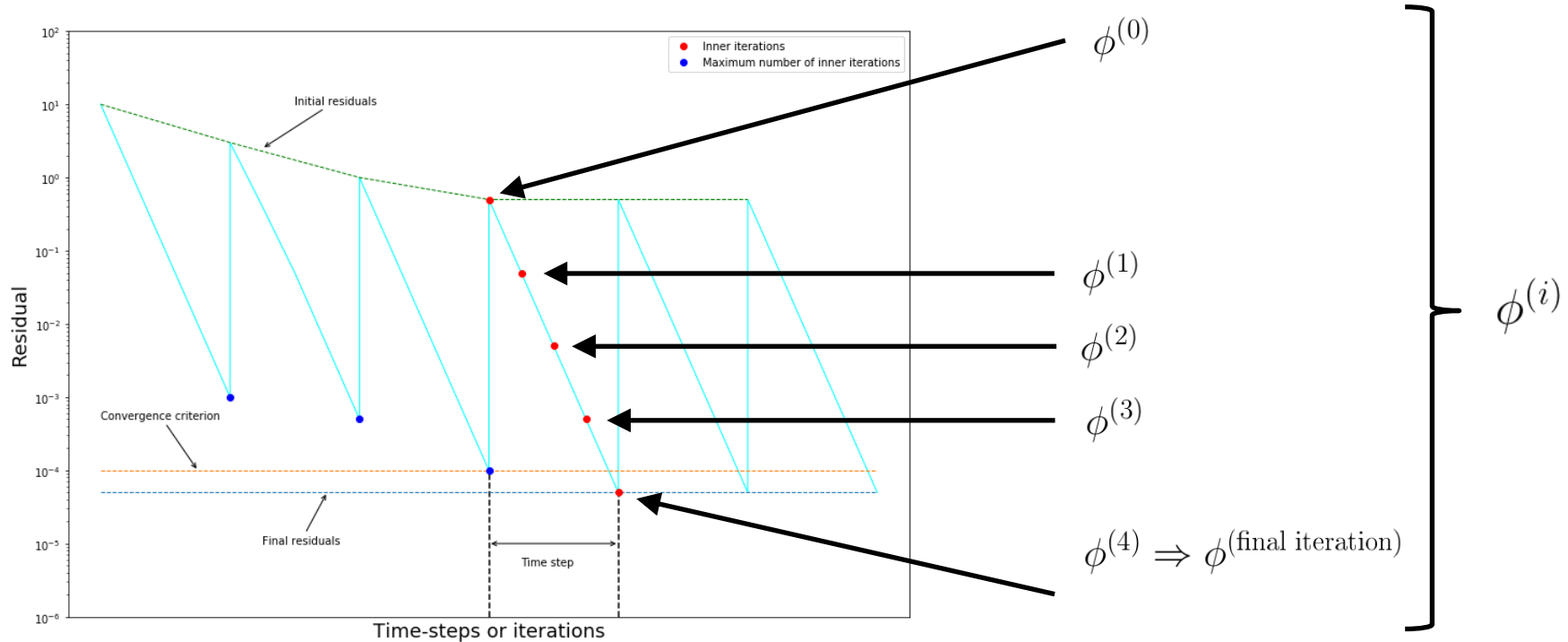
**PISO-ITA (PIMPLE in OpenFOAM)**

## **Linear solvers – Crunching numbers**

# The FVM in OpenFOAM

## Linear solvers

- To get a better idea of how iterative methods work, and what are initial residuals and final residuals, let us take another look at a residual plot.

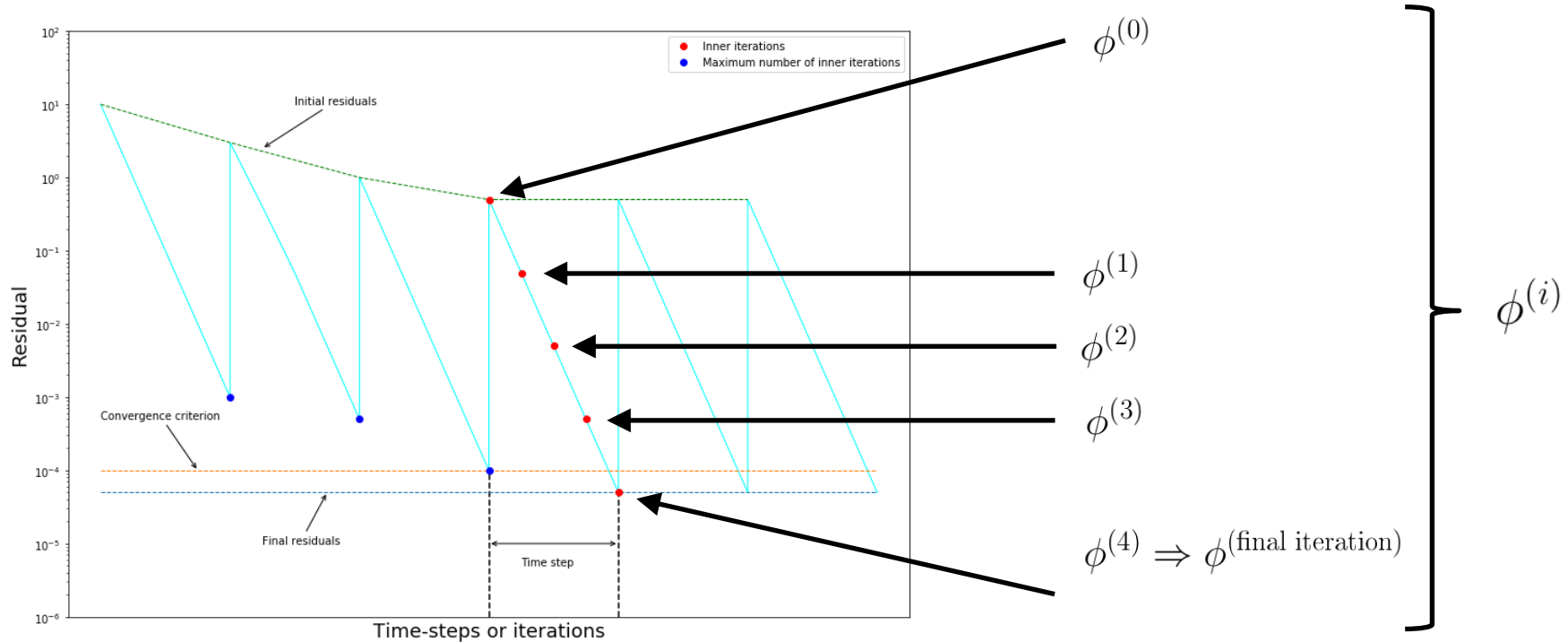


- $\phi^{(0)}$  is the initial guess used to start the iterative solver.
- If the following condition is fulfilled  $|\mathbf{A}\phi^i - \mathbf{b}| \leq |\mathbf{r}|$  (where  $\mathbf{r}$  is the convergence criterion or tolerance), the linear solver will stop iterating and will advance to the next time-step.
- By working in an iterative way, every single iteration  $\phi^{(i)}$  is a better approximation of the previous iteration  $\phi^{(i-1)}$ .

# The FVM in OpenFOAM

## Linear solvers

- To get a better idea of how iterative methods work, and what are initial residuals and final residuals, let us take another look at a residual plot.



- Remember, you can also do many correctors within a single time-step.
- Sometimes the linear solver might stop before reaching the predefined convergence criterion because it has reached the maximum number of iterations, you should be careful of this because we are talking about unconverged iterations.

# The FVM in OpenFOAM

## Linear solvers

```
solvers ←
{
  p
  {
    solver      PCG;
    preconditioner DIC;
    tolerance   1e-06;
    relTol      0;
  }
  pFinal
  {
    $p;
    relTol 0;
  }
  U
  {
    solver      PBiCGStab;
    preconditioner DILU;
    tolerance   1e-08;
    relTol      0;
  }
}

PISO ←
{
  nCorrectors 2;
  nNonOrthogonalCorrectors 1;
}
```

- The equation solvers, tolerances, and algorithms are controlled from the sub-dictionary **solvers** located in the *fvSolution* dictionary file.
- In the dictionary file *fvSolution*, and depending on the solver you are using you will find the additional sub-dictionaries **PISO**, **PIMPLE**, and **SIMPLE**, which will be described later.
- In this dictionary is where we are telling OpenFOAM how to crunch numbers.
- The **solvers** sub-dictionary specifies each linear-solver that is used for each equation being solved.
- If you forget to define a linear-solver, OpenFOAM will let you know what are you missing.
- The syntax for each entry within the **solvers** sub-dictionary uses a keyword that is the word relating to the variable being solved in the particular equation and the options related to the linear-solver.

# The FVM in OpenFOAM

## Linear solvers

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance        1e-06;
        relTol           0;
    }
    pFinal
    {
        $p;
        relTol 0;
    }
    U
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance        1e-08;
        relTol           0;
    }
}

PISO
{
    nCorrectors 2;
    nNonOrthogonalCorrectors 1;
}
```

The diagram illustrates the configuration of linear solvers for different variables in OpenFOAM. It shows three solver entries: 'p' (pressure), 'pFinal' (final pressure correction), and 'U' (velocity). Each entry specifies a solver, a preconditioner, an absolute tolerance, and a relative tolerance. The 'p' and 'pFinal' entries use the PCG solver with the DIC preconditioner, while the 'U' entry uses the PBiCGStab solver with the DILU preconditioner. The diagram uses brackets and arrows to group the settings for each variable and point them to the corresponding variable name in the list of solvers.

- In this generic case, to solve the pressure (**p**) we are using the **PCG** method with the **DIC** preconditioner, an absolute **tolerance** equal to 1e-06 and a relative tolerance **relTol** equal to 0.
- The entry **pFinal** refers to the final pressure correction (notice that we are using macro syntax), and we are using a relative tolerance **relTol** equal to 0.
- To solve the velocity field (**U**) we are using the **PBiCGStab** method with the **DILU** preconditioner, an absolute **tolerance** equal to 1e-08 and a relative tolerance **relTol** equal to 0.
- The linear solvers will iterate until reaching any of the tolerance values set by the user or reaching a maximum value of iterations (optional entry).
- FYI, solving for the velocity is relatively inexpensive, whereas solving for the pressure is expensive.
- The pressure equation is particularly important as it governs mass conservation.
- If you do not solve the equations accurately enough (tolerance), the physics might be wrong.
- Selection of the tolerance is of paramount importance.

# The FVM in OpenFOAM

## Linear solvers

```
solvers
{
  p
  {
    solver      PCG;
    preconditioner DIC;
    tolerance   1e-06;
    relTol      0;
  }
  pFinal
  {
    $p;
    relTol 0;
  }
  U
  {
    solver      PCG;
    preconditioner DILU;
    tolerance   1e-08;
    relTol      0;
  }
}

PISO
{
  nCorrectors 2;
  nNonOrthogonalCorrectors 1;
}
```

- The linear solvers distinguish between symmetric matrices and asymmetric matrices.
- The symmetry of the matrix depends on the structure of the equation being solved.
- Pressure is a symmetric matrix and velocity is an asymmetric matrix.
- If you use the wrong linear solver, OpenFOAM will complain and will let you know what options are valid.
- In the following error screen, we are using a symmetric solver for an asymmetric matrix,

-> FOAM FATAL IO ERROR :

Unknown asymmetric matrix solver PCG

Valid asymmetric matrix solvers are :

```
4
(
  BICCG
  GAMG
  P
  smoothSolver
)
```

# The FVM in OpenFOAM

## Linear solvers

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
    }
    pFinal
    {
        $p;
        relTol 0;
    }
    U
    {
        solver          PBiCGStab;
        preconditioner   DILU;
        tolerance        1e-08;
        relTol           0;
    }
}

PISO
{
    nCorrectors 2;
    nNonOrthogonalCorrectors 1;
}
```

- The linear solvers are iterative, *i.e.*, they are based on reducing the equation residual over a succession of solutions.
- The residual is a measure of the error in the solution so that the smaller it is, the more accurate the solution.
- More precisely, the residual is evaluated by substituting the current solution into the equation and taking the magnitude of the difference between the left- and right-hand sides (L2-norm).

$$|\mathbf{A}\phi^k - \mathbf{b}| = |\mathbf{r}^k|$$

- It is also normalized to make it independent of the scale of the problem being analyzed.

$$\text{Residual} = \frac{|\mathbf{r}|}{\text{Normalization factor}} < \text{Tolerance}$$



# The FVM in OpenFOAM

## Linear solvers

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
    }
    pFinal
    {
        $p;
        relTol 0;
    }
    U
    {
        solver          PBiCG;
        preconditioner   DILU;
        tolerance        1e-08;
        relTol           0;
        minIter          3;
        maxIter          100;
    }
}

PISO
{
    nCorrectors 2;
    nNonOrthogonalCorrectors 1;
}
```

- Before solving an equation for a particular field, the initial residual is evaluated based on the current values of the field.
- After each solver iteration the residual is re-evaluated. The solver stops if either of the following conditions are reached:
  - The residual falls below the solver tolerance, **tolerance**.
  - The ratio of current to initial residuals falls below the solver relative tolerance, **relTol**.
  - The number of iterations exceeds a maximum number of iterations, **maxIter**.
- The solver tolerance should represent the level at which the residual is small enough that the solution can be deemed sufficiently accurate.
- The keyword **maxIter** is optional and the default value is 1000.
- The user can also define the minimum number of iterations using the keyword **minIter**. This keyword is optional, and the default value is 0.

# The FVM in OpenFOAM

## Linear solvers

- These are the linear solvers available in OpenFOAM:
  - **GAMG** → Multigrid solver
  - **PBiCG** → Newton-Krylov solver
  - **PBiCGStab** → Newton-Krylov solver
  - **PBiCICG** → Newton-Krylov solver – Coupled (only for asymmetric matrices)
  - **PBiCCCG** → Newton-Krylov solver – Coupled (only for asymmetric matrices)
  - **PCICG** → Newton-Krylov solver
  - **PCG** → Newton-Krylov solver
  - **smoothSolver** → Relaxation iterative solver
  - **diagonalSolver** → Explicit solver (back substitution)
- You will find the source code of the linear solvers in the following directory:
  - `$WM_PROJECT_DIR/src/OpenFOAM/matrices/lduMatrix/solvers`
  - `$WM_PROJECT_DIR/src/OpenFOAM/matrices/LduMatrix/Solvers`

# The FVM in OpenFOAM

## Linear solvers

- These are the preconditioners available in OpenFOAM:
  - **diagonal**
  - **DIC**
  - **DILU**
  - **FDIC**
  - **GAMG**
  - **noPreconditioner**
- The preconditioners are mainly used in conjunction with the conjugate gradient solvers (Newton-Krylov solvers), and it is highly advisable to use them as they accelerate the solution.
- You will find the source code in the following directory:
  - `$WM_PROJECT_DIR/src/OpenFOAM/matrices/lduMatrix/preconditioners`
  - `$WM_PROJECT_DIR/src/OpenFOAM/matrices/LduMatrix/Preconditioners`

# The FVM in OpenFOAM

## Linear solvers

- These are the smooth solvers (or relaxation solvers) available in OpenFOAM:
  - **DIC**
  - **DICGaussSeidel**
  - **DILU**
  - **DILUGaussSeidel**
  - **FDIC**
  - **GaussSeidel**
  - **nonBlockingGaussSeidel**
  - **symGaussSeidel**
- The smooth solvers (**smoothSolver**) require the smoother (actual solver) to be specified.
- When using the smooth solvers, the user can optionally specify the number of sweeps by using the **nSweeps** keyword.
- You will find the source code in the following directory:
  - `$WM_PROJECT_DIR/src/OpenFOAM/matrices/lduMatrix/smoother`
  - `$WM_PROJECT_DIR/src/OpenFOAM/matrices/LduMatrix/Smoothers`

# The FVM in OpenFOAM

## Linear solvers

- As you can see, when it comes to linear solvers there are many options and combinations available in OpenFOAM.
- When it comes to choosing the linear solvers, there is no written theory.
- It is problem and hardware dependent (type of the mesh, physics involved, processor cache memory, network connectivity, partitioning method, and so on).
- Most of the times using the **GAMG** method (geometric-algebraic multi-grid), is the best choice for symmetric matrices (e.g., pressure).
- The **GAMG** method should converge fast (less than 100 iterations). If it is taking more iterations, try to change the some of the solver options (pre-sweeps, post-sweeps, agglomeration, and so on).
- And if it is taking too long or it is unstable, use the **PCG** solver (Newton-Krylov) with a good preconditioner.
- When running with many cores (more than 1000), using the **PCG** might be a better choice.

# The FVM in OpenFOAM

## Linear solvers

- For asymmetric matrices, the **PBiCGStab** method with **DILU** preconditioner is a good choice.
- The **smoothSolver** solver with smoother **GaussSeidel**, also performs very well.
- If the **PBiCGStab** method with **DILU** preconditioner mysteriously crashed with an error related to the preconditioner, use the **smoothSolver** or change the preconditioner.
- But in general, the **PBiCGStab** solver should be faster than the **smoothSolver** solver.
- Remember, asymmetric matrices are assembled from the velocity (**U**), and the transported quantities (**k**, **omega**, **epsilon**, **e**, **T**, and so on).
- Usually, computing the velocity and the transported quantities is inexpensive and fast, so it is a good idea to use a tight tolerance ( $1e-8$ ) for these fields.
- The diagonal solver is used for back-substitution, for instance, when computing density using the equation of state (we know **p**, **h**, **e** and **T**).

# The FVM in OpenFOAM

## Linear solvers

- A few comments on the linear solver residuals.
  - Residuals are not a direct indication that you are converging to the right solution.
  - The first time-steps the solution might not converge, this is acceptable.
  - Also, you might need to use a smaller time-step during the first iterations to maintain solver stability.
  - If the solution is not converging, try to reduce the time-step size.
  - Also, it is highly advisable to do at least one iteration (**minIter** keyword), but we recommend to do at least 3 iterations.

Time = 50

Courant Number mean: 0.044365026 max: 0.16800273

smoothSolver: Solving for Ux, Initial residual = 1.0907508e-09, Final residual = 1.0907508e-09, No Iterations 0

smoothSolver: Solving for Uy, Initial residual = 1.4677462e-09, Final residual = 1.4677462e-09, No Iterations 0

DICPCG: Solving for p, Initial residual = 1.0020944e-06, Final residual = 1.0746895e-07, No Iterations 1

time step continuity errors : sum local = 4.0107145e-11, global = -5.0601748e-20, cumulative = 2.637831e-18

ExecutionTime = 4.47 s ClockTime = 5 s

fieldMinMax minmaxdomain output:

min(p) = -0.37208345 at location (0.025 0.975 0.5)

max(p) = 0.77640927 at location (0.975 0.975 0.5)

min(U) = (0.00028445255 -0.00028138799 0) at location (0.025 0.025 0.5)

max(U) = (0.00028445255 -0.00028138799 0) at location (0.025 0.025 0.5)

}  
↑  
Residuals

# The FVM in OpenFOAM

## Linear solvers

- So how do we set the tolerances?
- The pressure equation (symmetric matrix) is particularly important, so we should resolve it accurately. Solving the pressure equation is the expensive part of the whole iterative process.
- For the pressure equation you can start the simulation with a **tolerance** equal to **1e-6** and **relTol** equal to **0.01**. After a while you change these values to **1e-6** and **0.0**, respectively.
- If the solver is too slow, you can change the convergence criterion to **1e-4** and **relTol** equal to **0.05**. You usually will do this during the first iterations.

### Loose tolerance

```
p
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-6;
    relTol           0.01;
}
```

### Tight tolerance

```
p
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-6;
    relTol           0.0;
}
```



# The FVM in OpenFOAM

## Linear solvers

- For the velocity field (**U**) and the transported quantities (asymmetric matrices), you can use the following criterion.
- Solving for these variables is relatively inexpensive, so you can start right away with a tight tolerance.

### Loose tolerance

```
U
{
    solver          PBiCGStab;
    preconditioner  DILU;
    tolerance       1e-8;
    relTol          0.01;
}
```

### Tight tolerance

```
U
{
    solver          PBiCGStab;
    preconditioner  DILU;
    tolerance       1e-8;
    relTol          0.0;
}
```

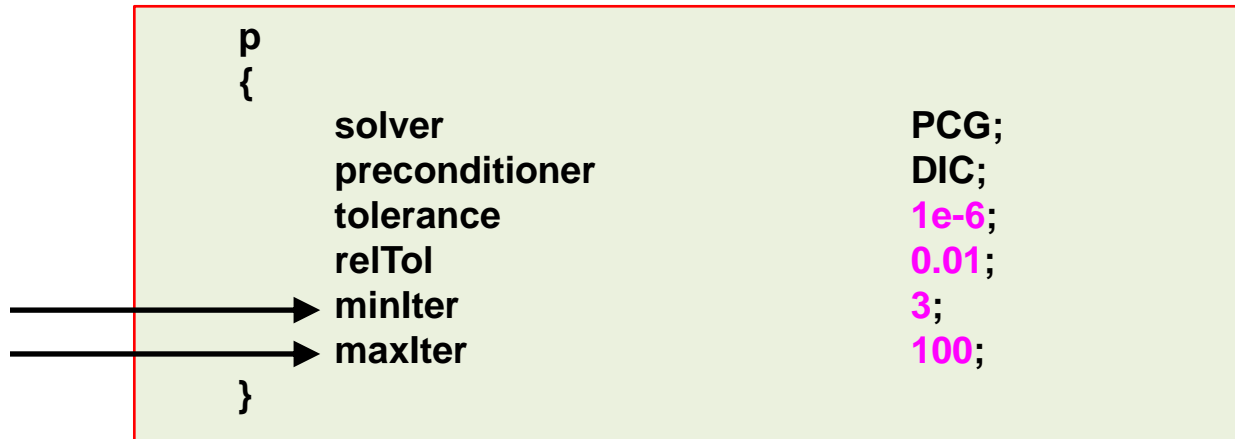
# The FVM in OpenFOAM

## Linear solvers

- It is also a good idea to set the minimum number of iterations (**minIter**) to 3.
- If your solver is doing too many iterations, you can set the maximum number of iterations (**maxIter**).
- But be careful, if the solver reaches the maximum number of iterations it will stop, we are talking about unconverged iterations.
- Setting the maximum number of iterations is especially useful during the first time-steps where the linear solver takes longer to converge.
- You can set **minIter** and **maxIter** in all symmetric and asymmetric linear solvers.

```
p
{
    solver
    preconditioner
    tolerance
    relTol
    minIter
    maxIter
}
```

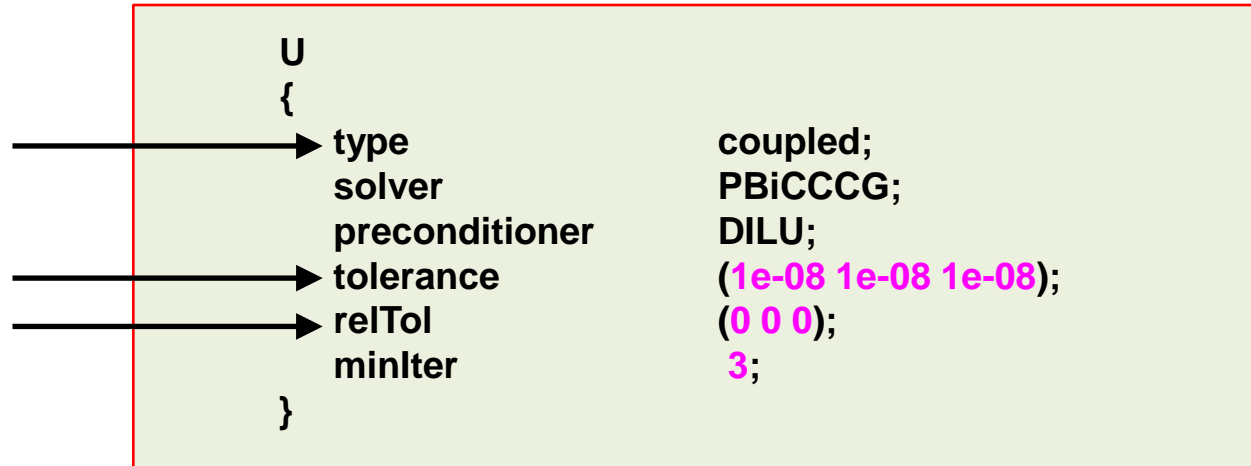
PCG;  
DIC;  
1e-6;  
0.01;  
3;  
100;

A diagram showing a configuration block for a linear solver named 'p'. The block is enclosed in a red border and contains a list of settings: solver, preconditioner, tolerance, relTol, minIter, and maxIter. To the right of these settings are the corresponding values: PCG;, DIC;, 1e-6;, 0.01;, 3;, and 100;. Two horizontal arrows point from the left towards the minIter and maxIter settings, highlighting them.

# The FVM in OpenFOAM

## Linear solvers

- For the velocity field ( $\mathbf{U}$ ), the default option is the conventional segregated linear solver. That is, you first solve for velocity component  $\mathbf{X}$ , then velocity component  $\mathbf{Y}$ , and finally velocity component  $\mathbf{Z}$ .
- You can get some improvement in terms of stability and turn around time by using the coupled matrix solver for vectors and tensors, *i.e.*, if you are solving the velocity field, you solve all the velocity components at once.
- To select the coupled solver, you need to set the solver type to **coupled**.
- In the coupled matrix solver, you set tolerance as a vector (absolute and relative).



# The FVM in OpenFOAM

## Linear solvers

- When you use the **PISO** or **PIMPLE** method, you need to set the tolerance for the pressure final corrector step, that is, **pFinal**.
- By proceeding in this way, you can put all the computational effort only in the last corrector step (**pFinal**). For example, you can use the following solver and tolerance criterion for all the intermediate corrector steps (**p**), then in the final corrector step (**pFinal**) you tight the solver tolerance.
- Have in mind that the pressure equation is particularly important, so we should resolve it accurately.
- We recommend to use a tight tolerance for the intermediate and final corrector steps of the pressure linear solvers (**p** and **pFinal**).

### Loose tolerance for p for intermediate corrector steps (p)

```
p
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-6;
    relTol           0.01;
}
```

### Tight tolerance for final corrector step (pFinal)

```
pFinal
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-6;
    relTol           0.0;
}
```

# The FVM in OpenFOAM

## Linear solvers

- When you use the **PISO** or **PIMPLE** method, you need to set the tolerance for the pressure final corrector step, that is, **pFinal**.
- By proceeding in this way, you can put all the computational effort only in the last corrector step (**pFinal**).
- For all the intermediate corrector steps (**p**), you can use a more relaxed convergence criterion.
- If you proceed in this way, it is recommended to do at least 2 corrector steps (**nCorrectors**).

```
Courant Number mean: 0.10556573 max: 0.65793603
deltaT = 0.00097959184
Time = 10
```

```
PIMPLE: iteration 1
```

```
DILUPBiCG: Solving for Ux, Initial residual = 0.0024649332, Final residual = 2.3403547e-09, No Iterations 4
```

```
DILUPBiCG: Solving for Uy, Initial residual = 0.0044355904, Final residual = 1.8966277e-09, No Iterations 4
```

```
DILUPBiCG: Solving for Uz, Initial residual = 0.010100894, Final residual = 1.4724403e-09, No Iterations 4
```

```
GAMG: Solving for p, Initial residual = 0.018497918, Final residual = 0.00058090899, No Iterations 3
```

```
GAMG: Solving for p, Initial residual = 0.00058090857, Final residual = 2.5748489e-05, No Iterations 5
```

```
time step continuity errors : sum local = 1.2367812e-09, global = 2.8865505e-11, cumulative = 1.057806e-08
```

```
GAMG: Solving for p, Initial residual = 0.00076032002, Final residual = 2.3965621e-05, No Iterations 3
```

```
GAMG: Solving for p, Initial residual = 2.3961044e-05, Final residual = 6.3151172e-06, No Iterations 2
```

```
time step continuity errors : sum local = 3.0345314e-10, global = -3.0075104e-12, cumulative = 1.0575052e-08
```

```
DILUPBiCG: Solving for omega, Initial residual = 0.00073937735, Final residual = 1.2839908e-10, No Iterations 4
```

```
DILUPBiCG: Solving for k, Initial residual = 0.0018291502, Final residual = 8.5494234e-09, No Iterations 3
```

```
ExecutionTime = 29544.18 s ClockTime = 29600 s
```

**p**

**p**

**pFinal**

1

2

nCorrectors

# The FVM in OpenFOAM

## Linear solvers

- If the **momentumPredictor** is enabled (by default in most solvers), you will need to set the tolerance for the final corrector step of the rest of the transported variables.
  - Namely **UFinal**, **kFinal**, **omegaFinal**, **hFinal**, and so on.
- By proceeding in this way, you can put all the computational effort only in the last corrector step (**.\*Final**).
- For all the intermediate corrector steps, you can use a more relaxed convergence criterion.
- Have in mind that solving these variables is relatively inexpensive, so you can start right away with a tight tolerance.

### Loose tolerance for p for intermediate corrector steps (U)

```
U
{
    solver          PBiCGStab;
    preconditioner  DILU;
    tolerance       1e-6;
    relTol          0.01;
}
```

### Tight tolerance for final corrector step (UFinal)

```
UFinal
{
    solver          PBiCGStab;
    preconditioner  DILU;
    tolerance       1e-6;
    relTol          0.0;
}
```

# The FVM in OpenFOAM

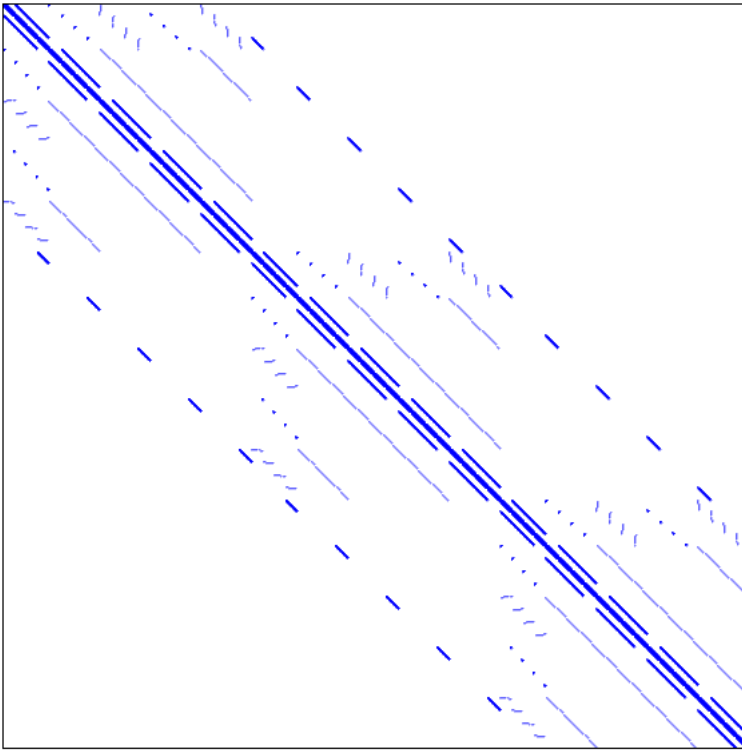
## Linear solvers

- As we are solving a sparse matrix, the more diagonal the matrix is, the best the convergence rate will be.
- So, it is highly advisable to use the utility `renumberMesh` before running the simulation.
  - `$> renumberMesh -overwrite`
- The utility `renumberMesh` can dramatically increase the speed of the linear solvers, specially during the initial iterations.
- You will find the source code and the master dictionary in the following directory:
  - `$WM_PROJECT_DIR/applications/utilities/mesh/manipulation/renumberMesh`

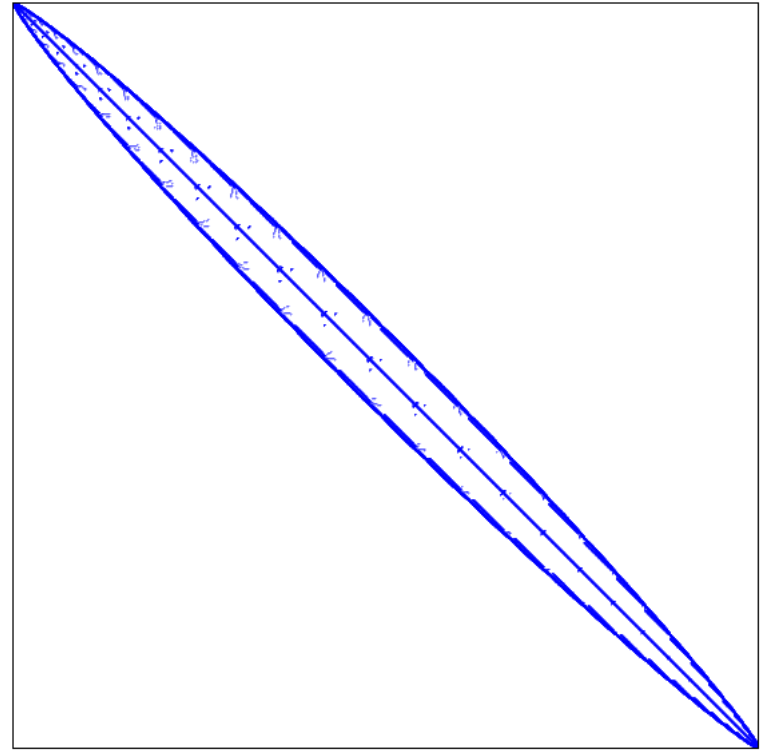
# The FVM in OpenFOAM

## Linear solvers

- The idea behind reordering is to make the matrix more diagonally dominant, therefore, speeding up the iterative solver.



Matrix structure plot before reordering



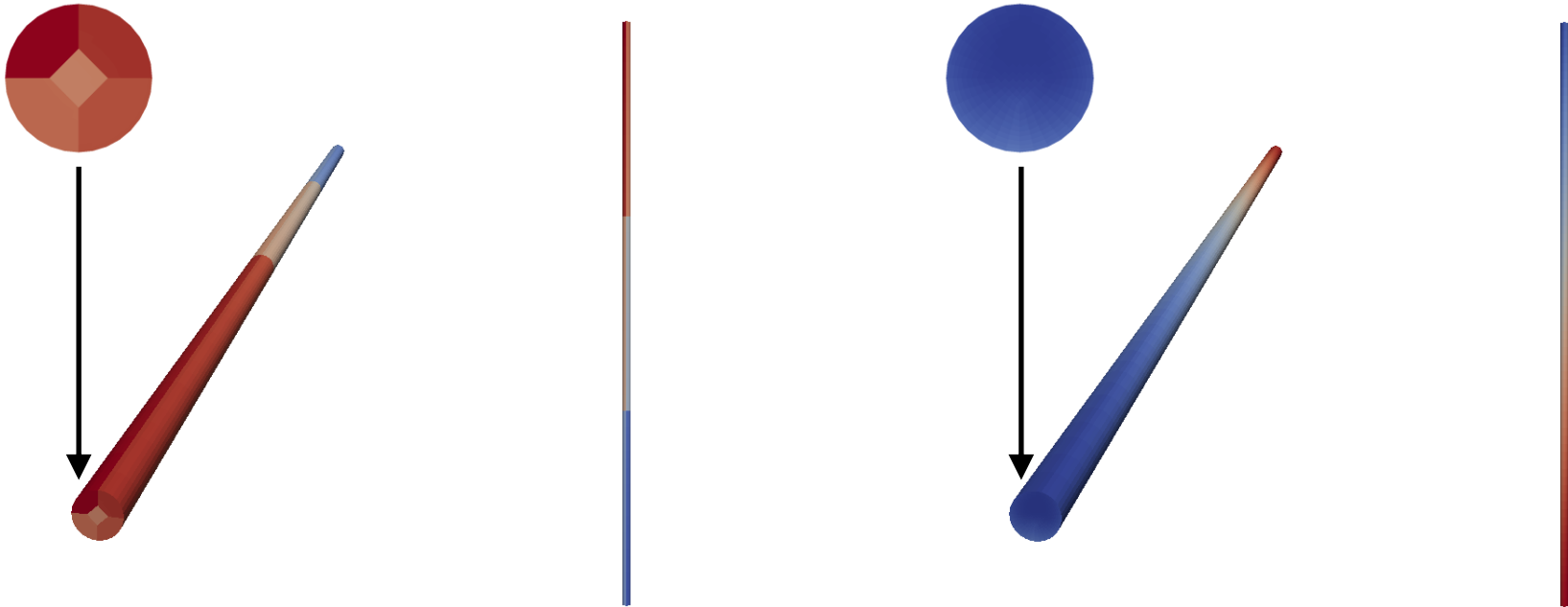
Matrix structure plot after reordering

**Note:**  
This is the actual pressure matrix from an OpenFOAM model case



# The FVM in OpenFOAM

## Linear solvers



**3D straight pipe mesh using an O-grid topology**

- Cells ordering before (left figure) and after (right figure) using `renumberMesh`. The colors represent the position of neighbor cells in the matrix.
- In the left figure the cells in each block are close together but each block is separated. The linear solver will perform poorly in this case.
- In the right figure the cells are all neighbors in the sparse matrix, the matrix is very diagonal. The linear solver will perform very well in this case.

# The FVM in OpenFOAM

## On the multigrid solvers

- The development of multigrid solvers (**GAMG** in OpenFOAM), together with the development of high-resolution TVD schemes and parallel computing, are among the most remarkable achievements of the history of CFD.
- Most of the time using the **GAMG** linear solver is fine. However, if you see that the linear solver is taking too long to converge or is converging in more than 100 iterations, it is better to use the **PCG** linear solver.
- Particularly, we have found that the **GAMG** linear solver in OpenFOAM does not perform very well when you scale your computations to more than 500 processors.
- Also, we have found that for some multiphase cases the **PCG** method outperforms the **GAMG**.
- But again, this is problem and hardware dependent.
- As you can see, you need to always monitor your simulations (stick to the screen for a while). Otherwise, you might end-up using a solver that is performing poorly. And this translate in increased computational time and costs.

# The FVM in OpenFOAM

## On the multigrid solvers tolerances

- If you go for the **GAMG** linear solver for symmetric matrices (e.g., pressure), the following tolerances are acceptable for most of the cases.

### Loose tolerance for p

```
p
{
    solver          GAMG;
    tolerance        1e-6;
    relTol           0.01;
    smoother         GaussSeidel;
    nPreSweeps        0;
    nPostSweeps       2;
    cacheAgglomeration on;
    agglomerator       faceAreaPair;
    nCellsInCoarsestLevel 100;
    mergeLevels       1;
    minIter           3;
}
```

### Tight tolerance for pFinal

```
pFinal
{
    solver          GAMG;
    tolerance        1e-6;
    relTol           0;
    smoother         GaussSeidel;
    nPreSweeps        0;
    nPostSweeps       2;
    cacheAgglomeration on;
    agglomerator       faceAreaPair;
    nCellsInCoarsestLevel 100;
    mergeLevels       1;
    minIter           3;
}
```

#### NOTE:

The GAMG parameters are not optimized, that is up to you.  
Most of the times is safe to use the default parameters.

# The FVM in OpenFOAM

## On the solvers tolerances for symmetric matrices

- If you do not use the **GAMG** solver for symmetric matrices (e.g., pressure), you can use the **PCG** solver with the **DIC** preconditioner and the following tolerances,

### Loose tolerance for p

```
p
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-6;
    relTol           0.01;
    minlter          3;
}
```

### Tight tolerance for pFinal

```
pFinal
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-6;
    relTol           0.0;
    minlter          3;
}
```

- Again, the choice of the solver and preconditioner/smoothen is problem and hardware dependent.

# The FVM in OpenFOAM

## On the solvers tolerances for symmetric matrices

- Yet another efficient linear solver for pressure is the following.

### Loose tolerance for p

```
p
{
    solver          PCG;
    preconditioner
    {
        preconditioner GAMG;
        tolerance      1e-06;
        relTol         0;
    }
    tolerance        1e-6;
    relTol            0.01;
    minIter           3;
}
```

### Tight tolerance for pFinal

```
pFinal
{
    solver          PCG;
    preconditioner
    {
        preconditioner GAMG;
        tolerance      1e-06;
        relTol         0;
    }
    tolerance        1e-6;
    relTol            0.01;
    minIter           3;
}
```

- In this case, the preconditioner is using the GAMG method with the default parameters, which might not be the optimal ones.
- Again, the choice of the solver and preconditioner/smoothen is problem and hardware dependent.
- Most of the times, this is our choice of linear solver or pressure.

# The FVM in OpenFOAM

## On the solvers tolerances for asymmetric matrices

- Most of the times solving asymmetric matrices is inexpensive (**U**, **k**, **omega**, and so on). Therefore, you can start right away with a tight tolerance.
- You can use either of the following linear solvers.

### Tight tolerance

```
U
{
    solver          PBiCGStab;
    preconditioner  DILU;
    tolerance       1e-8;
    relTol          0.0;
    minIter         3;
}
```

### Tight tolerance

```
U
{
    solver          smoothSolver;
    preconditioner  GaussSeidel;
    tolerance       1e-8;
    relTol          0.0;
    minIter         3;
}
```

- Again, the choice of the solver and preconditioner/smoothers is problem and hardware dependent.

# The FVM in OpenFOAM

## Linear solvers tolerances – Steady simulations

- The previous tolerances are fine for unsteady solvers.
- For extremely coupled problems you might need to have tighter tolerances.
- You can use the same tolerances for steady solvers.
- However, it might be a good idea to use tighter tolerances.
- You can also set the convergence controls based on residuals of fields. The controls are specified in the **residualControls** sub-dictionary of the dictionary file *fvSolution*.

```
SIMPLE
{
    nNonOrthogonalCorrectors 2;

    residualControl ←
    {
        p 1e-4;
        U 1e-4; } ← Residual control for every
    }                field variable you are solving
}
```

# The FVM in OpenFOAM

**Linear solvers benchmarking study of a model case**



# The FVM in OpenFOAM

## Linear solvers benchmarking of a model case

- In the following benchmarking study, we will use the incompressible solver `icoFoam`, and the compressible solver `rhoPimpleFoam`.
  - `icoFoam` is a transient solver for incompressible, laminar flow of Newtonian fluids.
  - `rhoPimpleFoam` is a transient solver for laminar or turbulent flow of compressible fluids for HVAC and similar applications.
- The physics to be addressed is that of a laminar flow in a 3D straight pipe.
- For this benchmarking study, we will use several combinations of linear solvers and preconditioners.
- As you can see, this very simple physics can have very different performance using different linear solvers.
- Also, to accelerate the convergence rate we will renumber the mesh (reduce sparsity).
- To renumber the mesh, we will use the utility `renumberMesh`.

# The FVM in OpenFOAM

## Linear solvers benchmarking of a model case

Case	Linear solver for P	Preconditioner or smoother	MR	Time	QOI
IC1	PCG	FDIC	NO	278	2.8265539
IC2	smoothSolver	symGaussSeidel	NO	2070	2.8271198
IC3	ICCG	GAMG	NO	255	2.8265538
IC4	GAMG	GaussSeidel	NO	1471	2.8265538
IC5	PCG	GAMG-GaussSeidel	NO	302	2.8265538
IC6	GAMG	GaussSeidel	YES	438	2.8265539
IC7	PCG	FDIC	YES	213	2.8265535
IC8	PCG	GAMG-GaussSeidel	YES	283	2.8265538
IC9	ICCG	GAMG	YES	261	2.8265538
IC10	PCG	DIC	NO	244	2.8265539
IC11	PCG	FDIC	NO	138	2.1934228

**Solver used** = icoFoam – Incompressible case

**MR** = matrix reordering (renumberMesh)

**QOI** = quantity of interest. In this case the maximum velocity at the outlet (m/s)

**TIME** = clock time (seconds)

Remember to monitor a QOI to verify the goodness of the solution

# The FVM in OpenFOAM

## Linear solvers benchmarking of a model case

Case	Linear solver for P	Preconditioner or smoother	MR	Time	QOI
IC1	PCG	FDIC	NO	278	2.8265539
IC4	GAMG	GaussSeidel	NO	1471	2.8265538
IC9	ICCG	GAMG	YES	261	2.8265538

**Solver used =** icoFoam

Case	Linear solver for P	Preconditioner or smoother	MR	Time	QOI
C1	PCG	FDIC	NO	214	2.8271341
C2	GAMG	GaussSeidel	NO	895	2.8271357
C3	ICCG	GAMG	YES	562	2.8271357

**Solver used =** rhoPimpleFoam

**MR** = matrix reordering (`renumberMesh`)

**QOI** = quantity of interest. In this case the maximum velocity at the outlet (m/s)

**TIME** = clock time (seconds)

# The FVM in OpenFOAM

## Linear solvers benchmarking of a model case

- In cases IC1-IC10, we used the following tolerances for **p**,

```
tolerance      1e-6;  
relTol         0.01;
```

- And for **pFinal** we used the following solver and tolerances,


```
pFinal  
{  
    $p;  
    tolerance      1e-6;  
    relTol         0;  
}
```

# The FVM in OpenFOAM

## Linear solvers benchmarking of a model case

- In IC11 (which is the fastest case), we used the following tolerances for **p**,

```
tolerance      1e-6;  
relTol         0.01;  
maxIter        1;
```



- And for **pFinal** we used the following solver and tolerances,

```
pFinal  
{  
    $p;  
    tolerance    1e-6;  
    relTol       0;  
}
```

# The FVM in OpenFOAM

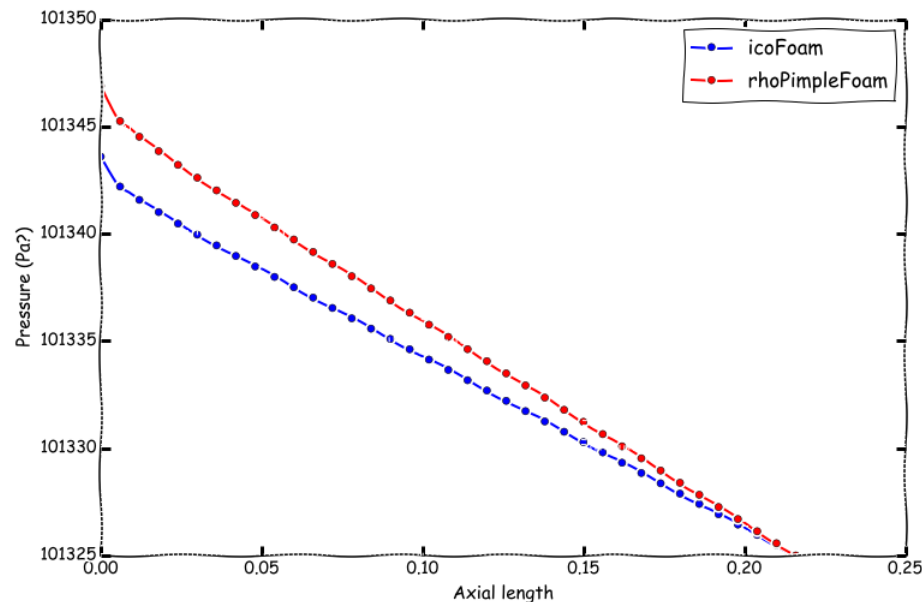
## Linear solvers benchmarking of a model case

- Notice that in case IC11 by setting the maximum number of iterations (**maxIter**) for **p** and **pFinal**, we managed to accelerated the iterative process.
- However, we were also forcing the linear solver not to converge.
- As you can see, the solver converged to the wrong solution.
- A good advice (and totally free), always monitor a quantity of interest (QOI) and do not to rely only on the residuals.
- In this simple case, you can see the importance of setting the proper tolerances.
- We can also see that some linear solvers perform better than others. This is problem and hardware dependent.
- Also, by using the utility `renumberMesh`, we managed to speed-up the iterative process.
- This is evident for the **GAMG** solver, where the speed-up was about 3 times. For the other solvers the speed-up was negligible.

# The FVM in OpenFOAM

## Linear solvers benchmarking of a model case

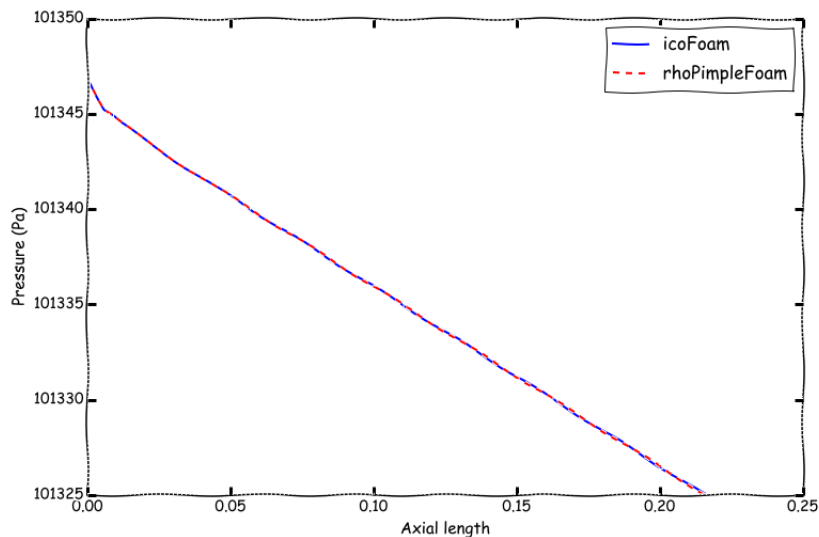
- Let us compare the pressure drop obtained with `icoFoam` and `rhoPimpleFoam`.
- At the end of the day, both solvers should give us very similar results.
- Remember, in `icoFoam` we work with relative pressure.
- After adding the reference pressure (101325 Pa) to the output pressure (pressure density or pressure divided density), we obtain the following results (which are not very encouraging).



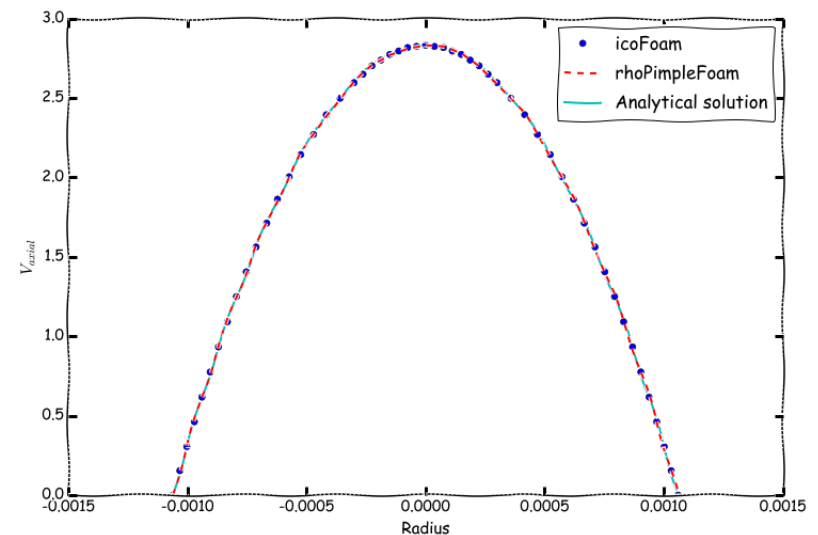
# The FVM in OpenFOAM

## Linear solvers benchmarking of a model case

- Let us compare the pressure drop obtained with `icoFoam` and `rhoPimpleFoam`.
- However, after correcting the output pressure by multiplying the pressure by the reference density, we get the following results (a perfect match).
- Remember, the pressure output of the incompressible solvers is the pressure divided by the reference density (pressure density), that is,  $p/\rho$



Pressure drop along the pipe axis



Velocity profile at the outlet



## **Unsteady and steady Simulations**

# The FVM in OpenFOAM

## How to run unsteady simulations in OpenFOAM?

- Select the time step. The time-step must be chosen in such a way that it resolves the time-dependent features and maintains solver stability.
- Select the time discretization scheme.
- Set the tolerance (absolute and/or relative) of the linear solvers.
- Monitor the CFL number.
- Monitor the stability and boundedness of the solution.
- Monitor a quantity of interest.
- And of course, you need to save the solution with a given frequency.
- Have in mind that unsteady simulations generate a lot of data.
- End time of the simulation?, it is up to you.
- In the *controlDict* dictionary you need to set runtime parameters and general instructions on how to run the case (such as time step and maximum CFL number). You also set the saving frequency.
- In the *fvSchemes* dictionary you need to set the time discretization scheme.
- In the *fvSolution* dictionary you need to set the linear solvers.
- Also, you will need to set the number of corrections of the velocity-pressure coupling method used (e.g., **PISO** or **PIMPLE**), this is done in the *fvSolution* dictionary.
- Additionally, you may set **functionObjects** in the *controlDict* dictionary. The **functionObjects** are used to do sampling, probing and co-processing while the simulation is running.

# The FVM in OpenFOAM

## How to run unsteady simulations in OpenFOAM?

```
startFrom latestTime;

startTime 0; ←

stopAt endTime;

endTime 10; ←

deltaT 0.0001; ←

writeControl runTime;

writeInterval 0.1; ←

purgeWrite 0;

writeFormat ascii;

writePrecision 8;

writeCompression off;

timeFormat general;

timePrecision 6;

runTimeModifiable yes; ←

adjustTimeStep yes;
maxCo 2.0;
maxDeltaT 0.001;
```

- The *controlDict* dictionary contains runtime simulation controls, such as, start time, end time, time step, saving frequency and so on. Most of the entries are self-explanatory.
- This generic case starts from time 0 (**startTime**), and it will run up to 10 seconds (**endTime**).
- It will write the solution every 0.1 seconds (**writeInterval**) of simulation time (**runTime**).
- The time step of the simulation is 0.0001 seconds (**deltaT**).
- It will keep all the solution directories (**purgeWrite**).
- It will save the solution in ascii format (**writeFormat**) with a precision of 8 digits (**writePrecision**).
- And as the option **runTimeModifiable** is on (**yes**), we can modify all these entries while we are running the simulation.

# The FVM in OpenFOAM

## How to run unsteady simulations in OpenFOAM?

```
startFrom    latestTime;

startTime    0;

stopAt       endTime;

endTime      10;

deltaT       0.0001;

writeControl  runTime;

writeInterval 0.1;

purgeWrite   0;

writeFormat   ascii;

writePrecision 8;

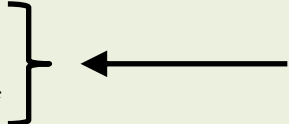
writeCompression off;

timeFormat    general;

timePrecision 6;

runTimeModifiable yes;

adjustTimeStep yes;
maxCo          2.0;
maxDeltaT      0.001;
```



- In this generic case, the solver supports adjustable time-step (**adjustTimeStep**).
- The option **adjustTimeStep** will automatically adjust the time step to achieve the maximum desired courant number (**maxCo**) or time-step size (**maxDeltaT**).
- When any of these conditions is reached, the solver will stop scaling the time-step size.
- Remember, the first time-step of the simulation is done using the value defined with the keyword **deltaT** and then it is automatically scaled (up or down), to achieve the desired maximum values (**maxCo** and **maxDeltaT**).
- It is recommended to start the simulation with a low time-step in order to let the solver scale-up the time-step size.
- The feature **adjustTimeStep** is only present in the **PIMPLE** family solvers, but it can be added to any solver by modifying the source code.
- If you are planning to use large time steps (CFL much higher than 1), it is recommended to do at least 3 correctors steps (**nCorrectors**) in **PISO/PIMPLE** loop, and at least 2 outer correctors in the **PIMPLE** loop.

# The FVM in OpenFOAM

## How to run unsteady simulations in OpenFOAM?

```
startFrom    latestTime;

startTime    0;

stopAt       endTime;

endTime      10;

deltaT       0.0001;

writeControl adjustableRunTime; ←

writeInterval 0.1;

purgeWrite   0;

writeFormat  ascii;

writePrecision 8;

writeCompression off;

timeFormat   general;

timePrecision 6;

runTimeModifiable yes;

adjustTimeStep yes;
maxCo          2.0;
maxDeltaT      0.001; } ←
```

- A word of caution about adjustable time-step (**adjustTimeStep**).
- This option will automatically adjust the time step to achieve the maximum desired courant number (**maxCo**) or time-step size (**maxDeltaT**).
- If the **maxDeltaT** condition is not reached, the solver will adapt the time-step to achieve the target **maxCo**, and as the time-step is not fixed this might introduce spurious oscillations in the solution.
- It is recommended to use this option at the beginning of the simulation and as soon as the solution stabilizes try fixed the time-step.
- Also, try to avoid using adjustable time step together with the option **adjustableRunTime**.
- The option **adjustableRunTime** will adjust the time-step to save the solution at the precise write intervals, and this might introduce numerical oscillations due to the fact that the time-step is changing.
- Also, the fact that you are using an adaptive time-step can have a negative effect when doing signal analysis.

# The FVM in OpenFOAM

## How to run unsteady simulations in OpenFOAM?

```
ddtSchemes ←  $\frac{\partial \phi}{\partial t}$ 
{
    default backward;
}

gradSchemes
{
    default Gauss linear;
    grad(p) Gauss linear;
}

divSchemes
{
    default none;
    div(phi,U) Gauss linear;
}

laplacianSchemes
{
    default Gauss linear orthogonal;
}

interpolationSchemes
{
    default linear;
}

snGradSchemes
{
    default orthogonal;
}
```

- The *fvSchemes* dictionary contains the information related to time discretization and spatial discretization schemes.
- In this generic case we are using the **backward** method for time discretization (**ddtSchemes**).
- This scheme is second order accurate but oscillatory.
- The parameters can be changed on-the-fly.

# The FVM in OpenFOAM

## How to run unsteady simulations in OpenFOAM?

```
solvers
{
  p
  {
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-06;
    relTol           0;
  }

  pFinal
  {
    $p;
    relTol 0;
  }

  "U.*"
  {
    solver          smoothSolver;
    smoother         symGaussSeidel;
    tolerance        1e-08;
    relTol           0;
  }
}

PIMPLE
{
  nOuterCorrectors 1;
  nCorrectors       2;
  nNonOrthogonalCorrectors 1;
}
```

- The *fvSolution* dictionary contains the instructions of how to solve each discretized linear equation system.
- As for the *controlDict* and *fvSchemes* dictionaries, the parameters can be changed on-the-fly.
- In this generic case, to solve the pressure (**p**) we are using the **PCG** method with the **DIC** preconditioner, an absolute **tolerance** equal to 1e-06 and a relative tolerance **relTol** equal to 0.
- The entry **pFinal** refers to the final pressure correction (notice that we are using macro syntax), and we are using a relative tolerance **relTol** equal to 0.
- To solve **U** and **UFinal** (**U.\*** using regex), we are using the **smoothSolver** method with an absolute **tolerance** equal to 1e-08 and a relative tolerance **relTol** equal to 0.
- The solvers will iterate until reaching any of the tolerance values set by the user or reaching a maximum value of iterations (optional entry).
- FYI, solving for the velocity is relatively inexpensive, whereas solving for the pressure is expensive.

# The FVM in OpenFOAM

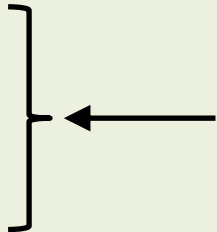
## How to run unsteady simulations in OpenFOAM?

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
    }

    pFinal
    {
        $p;
        relTol 0;
    }

    "U.*"
    {
        solver          smoothSolver;
        smoother         symGaussSeidel;
        tolerance        1e-08;
        relTol           0;
    }
}

PIMPLE
{
    nOuterCorrectors 1;
    nCorrectors      2;
    nNonOrthogonalCorrectors 1;
}
```



- The *fvSolution* dictionary also contains the **PIMPLE** and **PISO** sub-dictionaries.
- The **PIMPLE** sub-dictionary contains entries related to the pressure-velocity coupling method (the **PIMPLE** method).
- Setting the keyword **nOuterCorrectors** to 1 is equivalent to running using the **PISO** method.
- Remember, you need to do at least one **PISO** loop (**nCorrectors**).
- To gain more stability, especially when using large time-steps, you can use more outer correctors (**nOuterCorrectors**).
- Adding corrections increase the computational cost (**nOuterCorrectors** and **nCorrectors**).
- In this generic case, we are using 1 outer correctors (**nOuterCorrectors**), 2 inner correctors or **PISO** correctors (**nCorrectors**), and 1 correction due to non-orthogonality (**nNonOrthogonalCorrectors**).
- If you are using large time steps (CFL much higher than 1), it is recommended to do at least 3 correctors steps (**nCorrectors**) in **PISO/PIMPLE** loop.



# The FVM in OpenFOAM

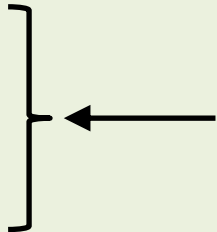
## How to run unsteady simulations in OpenFOAM?

```
solvers
{
  p
  {
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-06;
    relTol           0;
  }

  pFinal
  {
    $p;
    relTol 0;
  }

  U
  {
    solver          smoothSolver;
    smoother         symGaussSeidel;
    tolerance        1e-08;
    relTol           0;
  }
}

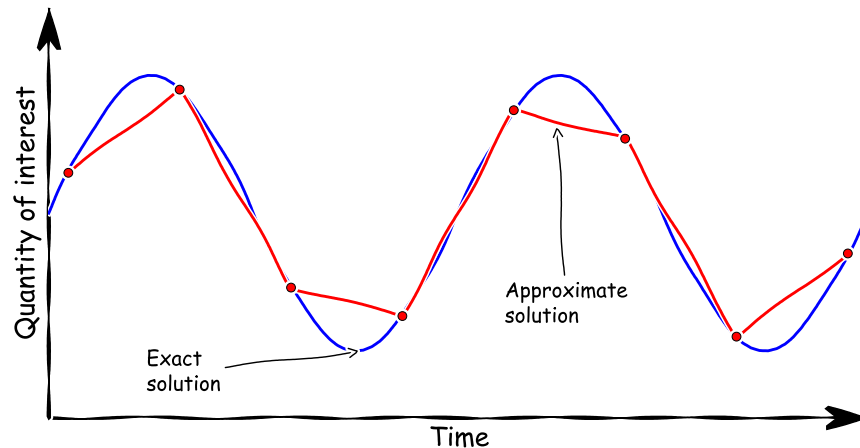
PISO
{
  nCorrectors 2;
  nNonOrthogonalCorrectors 1;
}
```



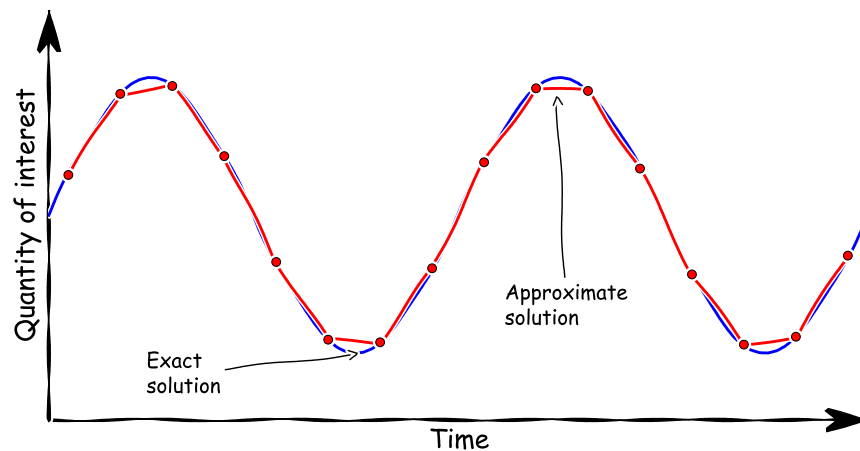
- If you use the **PISO** method for pressure-velocity coupling, you will need to define the **PISO** sub-dictionary.
- In this generic case we are doing two **PISO** corrections and one orthogonal correction.
- You need to do at least one **PISO** loop (**nCorrectors**).
- If you are planning to use large time steps (CFL much higher than 1), it is recommended to do at least 3 correctors steps (**nCorrectors**) in **PISO/PIMPLE** loop, and at least 2 outer correctors in the **PIMPLE** loop.

# The FVM in OpenFOAM

- Remember, when running unsteady simulations, the time-step must be chosen in such a way that it resolves the time-dependent features and maintains solver stability.



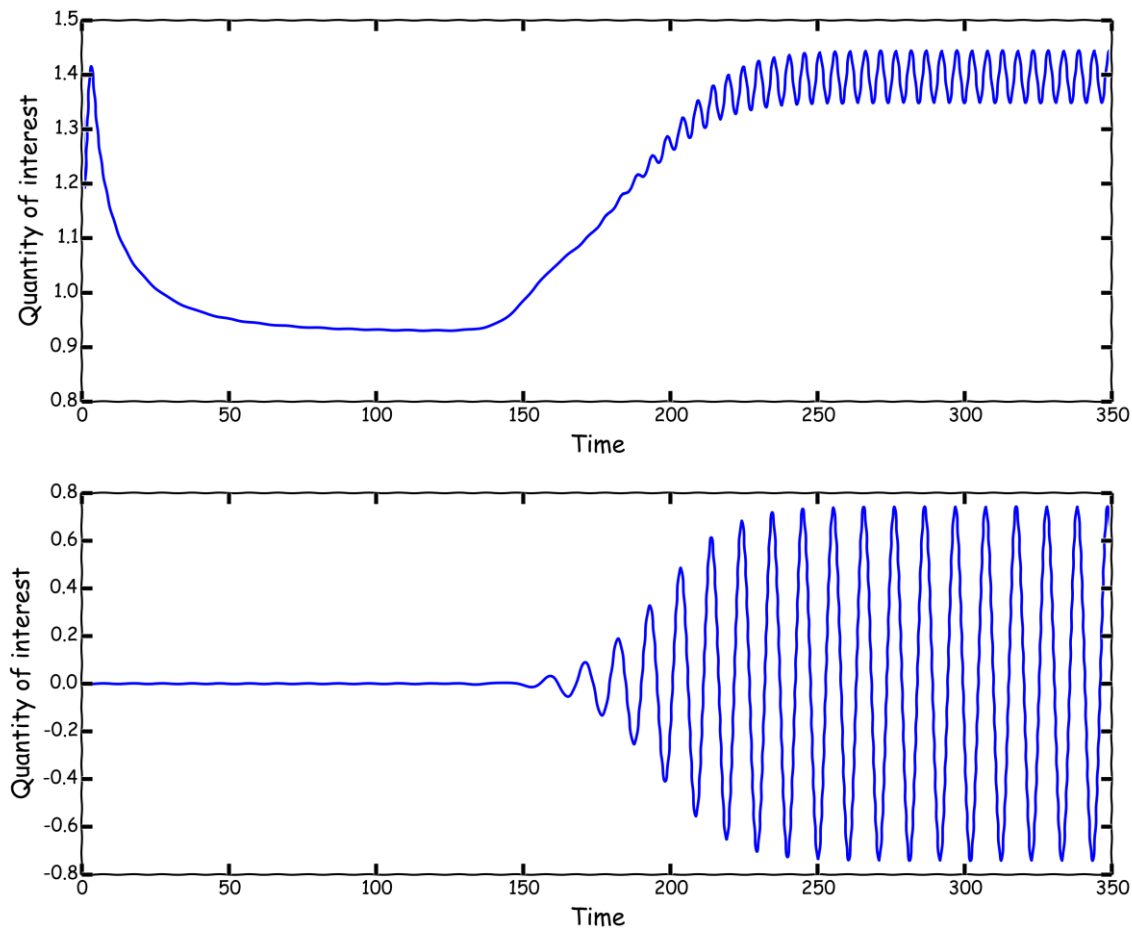
When you use large time steps you do not resolve well the physics



By using a smaller time step you resolve better the physics and you gain stability

# The FVM in OpenFOAM

- When running unsteady simulations, it is highly advisable to monitor a quantity of interest.
- The quantity of interest can fluctuate in time, this is an indication of unsteadiness.



# The FVM in OpenFOAM

## What about steady simulations?

- First of all, steady simulations are a big simplification of reality.
- Steady simulations is a trick used by CFDers to get fast outcomes with results that might be even more questionable.
- Remember, most of the flows you will encounter are unsteady so be careful of this hypothesis.
- In steady simulations, we made two assumptions:
  - We ignore unsteady fluctuations. That is, we neglect the time derivative in the governing equations.
  - We perform time averaging when dealing with stationary turbulence (RANS modeling)
- The advantage of steady simulations is that they require low computational resources, give fast outputs, and are easier to post-process and analyze.
- To do so, you need to use the appropriate solver and use the right discretization scheme.

# The FVM in OpenFOAM

## What about steady simulations?

- As you are not solving the time derivative, you do not need to set the time step.
- However, you need to tell OpenFOAM how many iterations you would like to run.
- You can also set the residual controls (**residualControl**), in the *fvSolution* dictionary file.
  - You set the **residualControl** in the **SIMPLE** sub-dictionary.
- If you do not set the residual controls, OpenFOAM will run until reaching the maximum number of iterations (**endTime**).
- You also will need to set the under-relaxation factor (**relaxationFactors**).
  - The iterative marching in steady solvers is controlled using under-relaxation factors.

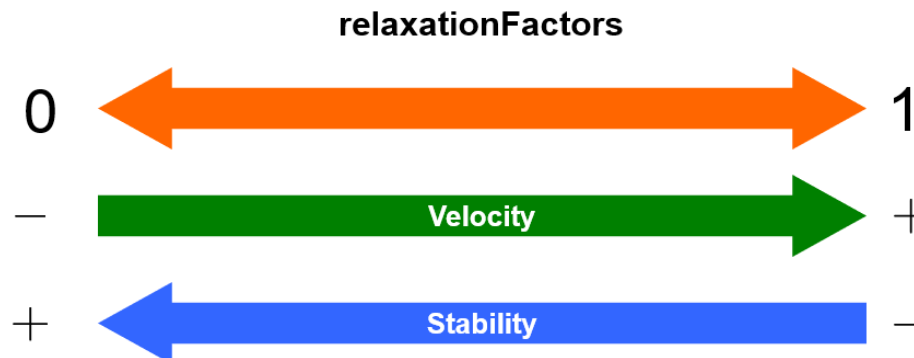
# The FVM in OpenFOAM

## What about steady simulations?

- You also need to set the under-relaxation factors.
- The under-relaxation factors control the change of the variable  $\phi$ .

$$\phi_P^n = \phi_P^{n-1} + \alpha(\phi_P^{n*} - \phi_P^{n-1})$$

- If  $\alpha < 1$  we are using under-relaxation.
- Under-relaxation is a feature typical of steady solvers using the **SIMPLE** method.
- If you do not set the under-relaxation factors, OpenFOAM will use the default hard-wired values (1.0 for all field variables or no under-relaxation).
- The under-relaxation factors are bounded between 0 and 1.
- Selecting the under-relaxation factors, it is kind of equivalent to selecting the right time step.



# The FVM in OpenFOAM

## What about steady simulations?

- Under-relaxation can be implicit (**equation** in OpenFOAM) or explicit (**field** in OpenFOAM).

$$\frac{a_P \phi}{\alpha} = \sum_N a_N \phi_N + b + \frac{1 - \alpha}{\alpha} a_P \phi_{n-1}$$

Implicit URF

$$\phi = \phi_{n-1} + \alpha \Delta \phi$$

Explicit URF

- You can relate URF to the CFL number as follows,

$$CFL = \frac{\alpha}{1 - \alpha}$$

$$\alpha = \frac{CFL}{1 + CFL}$$

- A small CFL number is equivalent to small URF.

# The FVM in OpenFOAM

## How to run steady simulations in OpenFOAM?

- In the *controlDict* dictionary you need to set runtime parameters and general instructions on how to run the case (such as the number of iterations to run).
- Remember to set also the saving frequency.
- In the *fvSchemes* dictionary you need to set the time discretization scheme, for steady simulations it must be **steadyState**.
- In the *fvSolution* dictionary you need to set the linear solvers, under-relaxation factors, and residual controls.
- Also, you will need to set the number of corrections of the velocity-pressure coupling method used (e.g., **SIMPLE** or **SIMPLEC**), this is done in the *fvSolution* dictionary.
- Additionally, you may set **functionObjects** in the *controlDict* dictionary.
- The **functionObjects** are used to do sampling, probing and co-processing while the simulation is running.



# The FVM in OpenFOAM

## How to run steady simulations in OpenFOAM?

- The under-relaxation factors (URF) control the change of the variable  $\phi$ .

$$\phi_P^n = \phi_P^{n-1} + \alpha(\phi_P^{n*} - \phi_P^{n-1})$$

- Under-relaxation is a feature typical of steady solvers using the **SIMPLE** family of methods.
- These are the URF commonly used with **SIMPLE** and **SIMPLEC** (industry standard),

SIMPLE		SIMPLEC		Pressure Usually does not require under-relaxing
<b>p</b>	<b>0.3;</b>	<b>p</b>	<b>1;</b>	
<b>U</b>	<b>0.7;</b>	<b>U</b>	<b>0.9;</b>	
<b>k</b>	<b>0.7;</b>	<b>k</b>	<b>0.9;</b>	
<b>omega</b>	<b>0.7;</b>	<b>omega</b>	<b>0.9;</b>	

- According to the physics involved you will need to add more under-relaxation factors.
- Finding the right URF involved experience and some trial and error.
- Selecting the URF it is kind of equivalent to selecting the right time step.
- Many times, steady simulations diverge because of wrongly chosen URF.

# The FVM in OpenFOAM

## How to run steady simulations in OpenFOAM?

```
startFrom    latestTime;  
startTime    0; ←  
stopAt       endTime;  
endTime      10000; ←  
deltaT       1; ←  
writeControl  runTime;  
writeInterval 100; ←  
purgeWrite   10; ←  
writeFormat  ascii;  
writePrecision 8;  
writeCompression off;  
timeFormat   general;  
timePrecision 6;  
runTimeModifiable yes; ←
```

- The *controlDict* dictionary contains runtime simulation controls, such as, start time, end time, time step, saving frequency and so on. Most of the entries are self-explanatory.
- As we are doing a steady simulation, let us talk about iterations instead of time (seconds).
- This generic case starts from iteration 0 (**startTime**), and it will run up to 10000 iterations (**endTime**).
- It will write the solution every 100 iterations (**writeInterval**) of simulation time (**runTime**).
- It will advance the solution one iteration at a time (**deltaT**).
- It will keep the last 10 saved solutions (**purgeWrite**).
- It will save the solution in ascii format (**writeFormat**) with a precision of 8 digits (**writePrecision**).
- And as the option **runTimeModifiable** is on (**true**), we can modify all these entries while we are running the simulation.

# The FVM in OpenFOAM

## How to run steady simulations in OpenFOAM?

```
ddtSchemes
{
    default    steadyState; ←  $\frac{\partial \phi}{\partial t}$ 
}

gradSchemes
{
    default    Gauss linear;
    grad(p)    Gauss linear;
}

divSchemes
{
    default    none;
    div(phi,u) bounded Gauss linear;
}

laplacianSchemes
{
    default    Gauss linear orthogonal;
}

interpolationSchemes
{
    default    linear;
}

snGradSchemes
{
    default    orthogonal;
}
```

- The *fvSchemes* dictionary contains the information related to time discretization and spatial discretization schemes.
- In this generic case and as we are interested in using a steady solver, we are using the **steadyState** method for time discretization (**ddtSchemes**).
- It is not a good idea to switch between steady and unsteady schemes on-the-fly.
- For steady state cases, the bounded form can be applied to the divSchemes, in this case, div(phi,U) **bounded** Gauss linear.
- This adds a linearized, implicit source contribution to the transport equation of the form,

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) - (\nabla \cdot \mathbf{u})\mathbf{u} = \nabla \cdot (\Gamma \nabla \mathbf{u}) + S$$

- This term removes a component proportional to the continuity error.
- This acts as a convergence aid to tend towards a bounded solution as the calculation proceeds.
- At convergence, this term becomes zero and does not contribute to the final solution.

# The FVM in OpenFOAM

## How to run steady simulations in OpenFOAM?

```
ddtSchemes
{
    default    steadyState; ←  $\frac{\partial \phi}{\partial t}$ 
}

gradSchemes
{
    default    Gauss linear;
    grad(p)    Gauss linear;
}

divSchemes
{
    default    none;
    div(phi,u) bounded Gauss linear;
}

laplacianSchemes
{
    default    Gauss linear orthogonal;
}

interpolationSchemes
{
    default    linear;
}

snGradSchemes
{
    default    orthogonal;
}
```

- The keyword **bounded** in the divergence discretization schemes, seems to be related to an attempt to recast the non-conservative formulation of the equations without enforcing the divergence-free condition.
- The bounded scheme targets the material derivative of the governing equations.
- When using the bounded scheme, we are actually solving the following form of the equations,

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \underbrace{\nabla \cdot (\rho \mathbf{u} \mathbf{u}) - \rho \mathbf{u} (\nabla \cdot \mathbf{u})}_{\nabla \cdot (\mathbf{u} \mathbf{u}) - \mathbf{u} \nabla \cdot \mathbf{u} = \mathbf{u} \cdot \nabla \mathbf{u}} = \nabla \cdot (\rho \Gamma \nabla \mathbf{u}) + S$$

$$\nabla \cdot (\mathbf{u} \mathbf{u}) - \mathbf{u} \nabla \cdot \mathbf{u} = \mathbf{u} \cdot \nabla \mathbf{u}$$

# The FVM in OpenFOAM

## How to run steady simulations in OpenFOAM?

```
ddtSchemes
{
    default    steadyState; ←  $\frac{\partial \phi}{\partial t}$ 
}

gradSchemes
{
    default    Gauss linear;
    grad(p)    Gauss linear;
}

divSchemes
{
    default    none;
    div(phi,U) bounded Gauss linear;
}

laplacianSchemes
{
    default    Gauss linear orthogonal;
}

interpolationSchemes
{
    default    linear;
}

snGradSchemes
{
    default    orthogonal;
}
```

- The use of the bounded scheme, it is a recommendation.
- In our personal experience, using the bounded scheme sometimes helps.
- In particular when dealing with compressible flows or complex physical modeling.
- Sometimes bounding some quantities can lead to convergence problems (very problem dependent).
- The bounded scheme is a mathematical trick that not always helps.
- From the physical point of view, the extra term added when using the bounded scheme, has no support or justification.
- You can find more information at the following link,
  - <https://openfoam.org/release/2-2-0/numerics-boundedness/>
  - <https://www.openfoam.com/documentation/guides/latest/doc/guide-schemes-divergence.html#sec-schemes-divergence-special-cases-steady-state>

# The FVM in OpenFOAM

## How to run steady simulations in OpenFOAM?

```
solvers
{
  p
  {
    solver          PCG;
    preconditioner  DIC;
    tolerance       1e-06;
    relTol          0;
  }

  U
  {
    solver          smoothSolver;
    smoother        symGaussSeidel;
    tolerance       1e-08;
    relTol          0;
  }
}

SIMPLE
{
  nNonOrthogonalCorrectors 2;

  residualControl
  {
    p 1e-4;
    U 1e-4;
  }
}
```

- The *fvSolution* dictionary contains the instructions of how to solve each discretized linear equation system.
- As for the *controlDict* and *fvSchemes* dictionaries, the parameters can be changed on-the-fly.
- In this generic case, to solve the pressure (**p**) we are using the **PCG** method with the **DIC** preconditioner, an absolute **tolerance** equal to 1e-06 and a relative tolerance **relTol** equal to 0.
- To solve **U** we are using the **smoothSolver** method with an absolute **tolerance** equal to 1e-08 and a relative tolerance **relTol** equal to 0.
- The solvers will iterate until reaching any of the tolerance values set by the user or reaching a maximum value of iterations (optional entry).

# The FVM in OpenFOAM

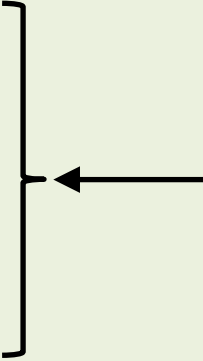
## How to run steady simulations in OpenFOAM?

```
solvers
{
  p
  {
    solver          PCG;
    preconditioner  DIC;
    tolerance       1e-06;
    relTol          0;
  }

  U
  {
    solver          smoothSolver;
    smoother        symGaussSeidel;
    tolerance       1e-08;
    relTol          0;
  }
}

SIMPLE
{
  nNonOrthogonalCorrectors 2;

  residualControl
  {
    p 1e-4;
    U 1e-4;
  }
}
```



- The *fvSolution* dictionary also contains the **SIMPLE** sub-dictionary .
- The **SIMPLE** sub-dictionary contains entries related to the pressure-velocity coupling method (the **SIMPLE** method).
- Increasing the number of **nNonOrthogonalCorrectors** corrections will add more stability but at a higher computational cost.
- Remember, **nNonOrthogonalCorrectors** is used to improve the gradient computation due to mesh quality.
- In this generic case, we are doing 2 corrections due to non-orthogonality (**nNonOrthogonalCorrectors**).
- The **SIMPLE** sub-dictionary also contains convergence controls based on residuals of fields. The controls are specified in the **residualControls** sub-dictionary.
- The user needs to specify a tolerance for one or more solved fields and when the residual for every field falls below the corresponding residual, the simulation terminates.
- If you do not set the **residualControls**, the solver will iterate until reaching the maximum number of iterations set in the *controlDict* dictionary.

# The FVM in OpenFOAM

## How to run steady simulations in OpenFOAM?

```
relaxationFactors
```

```
{
```

```
  fields ←
```

```
{
```

```
  p 0.3;
```

```
}
```

```
  equations ←
```

```
{
```

```
  U 0.7;
```

```
}
```

```
}
```

- The *fvSolution* dictionary also contains the **relaxationFactors** sub-dictionary.
- The **relaxationFactors** sub-dictionary which controls under-relaxation, is a technique used for improving stability when using steady solvers.
- Under-relaxation works by limiting the amount which a variable changes from one iteration to the next, either by modifying the solution matrix and source (**equations** keyword) prior to solving for a field or by modifying the field directly (**fields** keyword).
- An optimum choice of under-relaxation factors is one that is small enough to ensure stable computation but large enough to move the iterative process forward quickly.



# The FVM in OpenFOAM

## How to run steady simulations in OpenFOAM?

- Typical under-relaxation factors for the **SIMPLE** and **SIMPLEC** methods in OpenFOAM.
- Remember the under-relaxation factors are problem dependent.

### SIMPLE

```
relaxationFactors
```

```
{
```

```
  fields
```

```
  {
```

```
    p
```

```
    0.3;
```

```
  }
```

```
  equations
```

```
  {
```

```
    U
```

```
    0.7;
```

```
    k
```

```
    0.7;
```

```
    omega
```

```
    0.7;
```

```
  }
```

```
}
```

Explicit under-relaxation

Implicit under-relaxation

### SIMPLEC

```
relaxationFactors
```

```
{
```

```
  fields
```

```
  {
```

```
    p
```

```
    0.9;
```

```
  }
```

```
  equations
```

```
  {
```

```
    p
```

```
    0.9;
```

```
    U
```

```
    0.9;
```

```
    k
```

```
    0.9;
```

```
    omega
```

```
    0.9;
```

```
  }
```

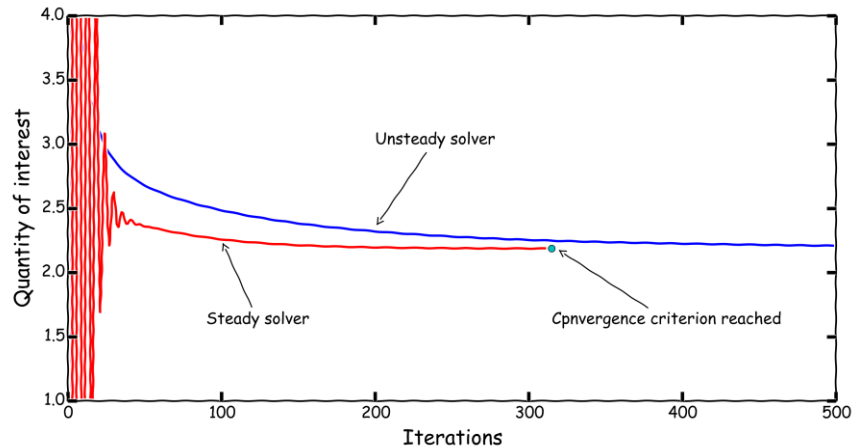
```
}
```

# The FVM in OpenFOAM

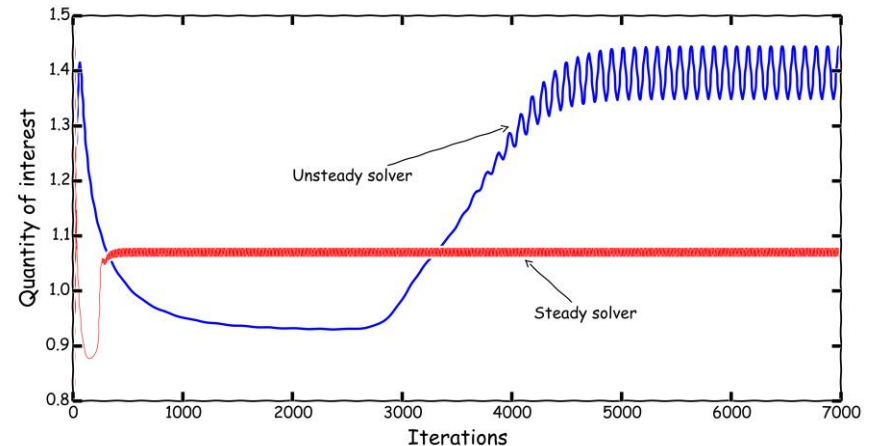
## Steady simulations vs. Unsteady simulations

- Steady simulations require less computational power than unsteady simulations.
- They are also much faster than unsteady simulations.
- But sometimes they do not converge to the right solution.
- They are easier to post-process and analyze (you just need to take a look at the last saved solution).
- You can use the solution of an unconverged steady simulation as initial conditions for an unsteady simulation.
- Remember, steady simulations are not time accurate, therefore we can not use them to compute time statistics or compute the shedding frequency

**Steady solution QOI**



**unsteady solution QOI**



# The FVM in OpenFOAM

**How to control the CFL number**

# The FVM in OpenFOAM

## How to control the CFL number

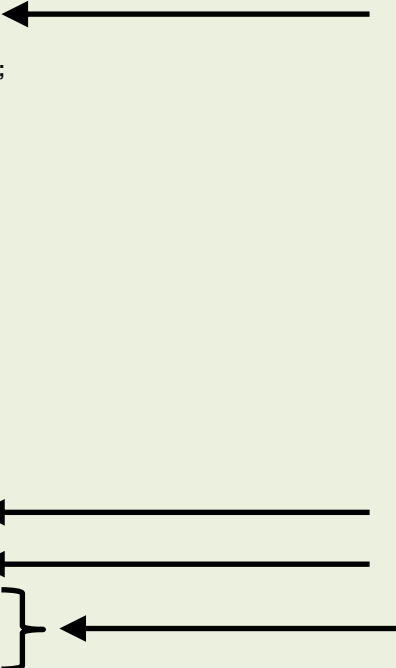
```
application      pimpleFoam;
startFrom        latestTime;
startTime        0;
stopAt           endTime;
endTime          10;
deltaT           0.0001;
writeControl      runtime;
writeInterval     0.1;
purgeWrite        0;
writeFormat       ascii;
writePrecision    8;
writeCompression off;
timeFormat        general;
timePrecision     6;
runTimeModifiable yes;
adjustTimeStep    yes;
maxCo             2.0;
maxDeltaT         0.001;
```

- You can control the CFL number by changing the mesh cell size or changing the time-step size.
- The easiest way is by changing the time-step size.
- If you refine the mesh, and you would like to have the same CFL number as the base mesh, you will need to decrease the time-step size.
- On the other side, if you coarse the mesh and you would like to have the same CFL number as the base mesh, you will need to increase the time-step size.
- The keyword **deltaT** controls the time-step size of the simulation (0.0001 seconds in this generic case).
- If you use a solver that supports adjustable time-step (**adjustTimeStep**), you can set the maximum CFL number and maximum allowable time-step using the keywords **maxCo** and **maxDeltaT**, respectively.

# The FVM in OpenFOAM

## How to control the CFL number

```
application      pimpleFoam;
startFrom        latestTime;
startTime        0;
stopAt           endTime;
endTime          10;
deltaT           0.0001;
writeControl      runtime;
writeInterval     0.1;
purgeWrite        0;
writeFormat       ascii;
writePrecision    8;
writeCompression off;
timeFormat        general;
timePrecision     6;
runTimeModifiable yes;
adjustTimeStep    yes;
maxCo             2.0;
maxDeltaT         0.001;
```



- The option **adjustTimeStep** will automatically adjust the time step to achieve the maximum desired courant number (**maxCo**) or time-step size (**maxDeltaT**).
- When any of these conditions is reached, the solver will stop scaling the time-step size.
- To use these features, you need to turn-on the option **adjustTimeStep**.
- Remember, the first time-step of the simulation is done using the value defined with the keyword **deltaT** and then it is automatically scaled (up or down), to achieve the desired maximum values (**maxCo** and **maxDeltaT**).
- It is recommended to start the simulation with a low time-step in order to let the solver scale-up the time-step size.
- If you want to change the values on-the-fly, you need to turn-on the option **runTimeModifiable**.
- The feature **adjustTimeStep** is only present in the **PIMPLE** family solvers, but it can be added to any solver by modifying the source code.

# The FVM in OpenFOAM

## The output screen

- This is the output screen of a solver supporting the option **adjustTimeStep**.
- In this case **maxCo** is equal 2 and **maxDeltaT** is equal to 0.001.
- Notice that the solver reached the maximum allowable **maxDeltaT**.

```
Courant Number mean: 0.10863988 max: 0.73950028
deltaT = 0.001
Time = 30.000289542261612
PIMPLE: iteration 1
DILUPBiCG: Solving for Ux, Initial residual = 0.003190933, Final residual = 1.0207483e-09, No Iterations 5
DILUPBiCG: Solving for Uy, Initial residual = 0.0049140114, Final residual = 8.5790109e-10, No Iterations 5
DILUPBiCG: Solving for Uz, Initial residual = 0.010705877, Final residual = 3.5464756e-09, No Iterations 4
GAMG: Solving for p, Initial residual = 0.024334674, Final residual = 0.0005180308, No Iterations 3
GAMG: Solving for p, Initial residual = 0.00051825089, Final residual = 1.6415538e-05, No Iterations 5
time step continuity errors : sum local = 8.768064e-10, global = 9.8389717e-11, cumulative = -2.6474162e-07
GAMG: Solving for p, Initial residual = 0.00087813032, Final residual = 1.6222017e-05, No Iterations 3
GAMG: Solving for p, Initial residual = 1.6217958e-05, Final residual = 6.4475277e-06, No Iterations 1
time step continuity errors : sum local = 3.4456296e-10, global = 2.6009599e-12, cumulative = -2.6473902e-07
ExecutionTime = 33091.06 s  ClockTime = 33214 s
fieldMinMax domainminandmax output:
  min(p) = -0.59404715 at location (-0.019 0.02082288 0.072) on processor 1
  max(p) = 0.18373302 at location (-0.02083962 -0.003 -0.136) on processor 1
  min(U) = (0.29583255 -0.4833922 -0.0048229716) at location (-0.02259661 -0.02082288 -0.072) on processor 0
  max(U) = (0.59710937 0.32913292 0.020043679) at location (0.11338793 -0.03267608 0.12) on processor 3
  min(nut) = 1.6594481e-10 at location (0.009 -0.02 0.024) on processor 0
  max(nut) = 0.00014588174 at location (-0.02083962 0.019 0.072) on processor 1
yPlus yplus output:
  patch square y+ : min = 0.44603573, max = 6.3894913, average = 2.6323389
  writing field yPlus
```

← Courant number (mean and maximum values)

← Current time-step

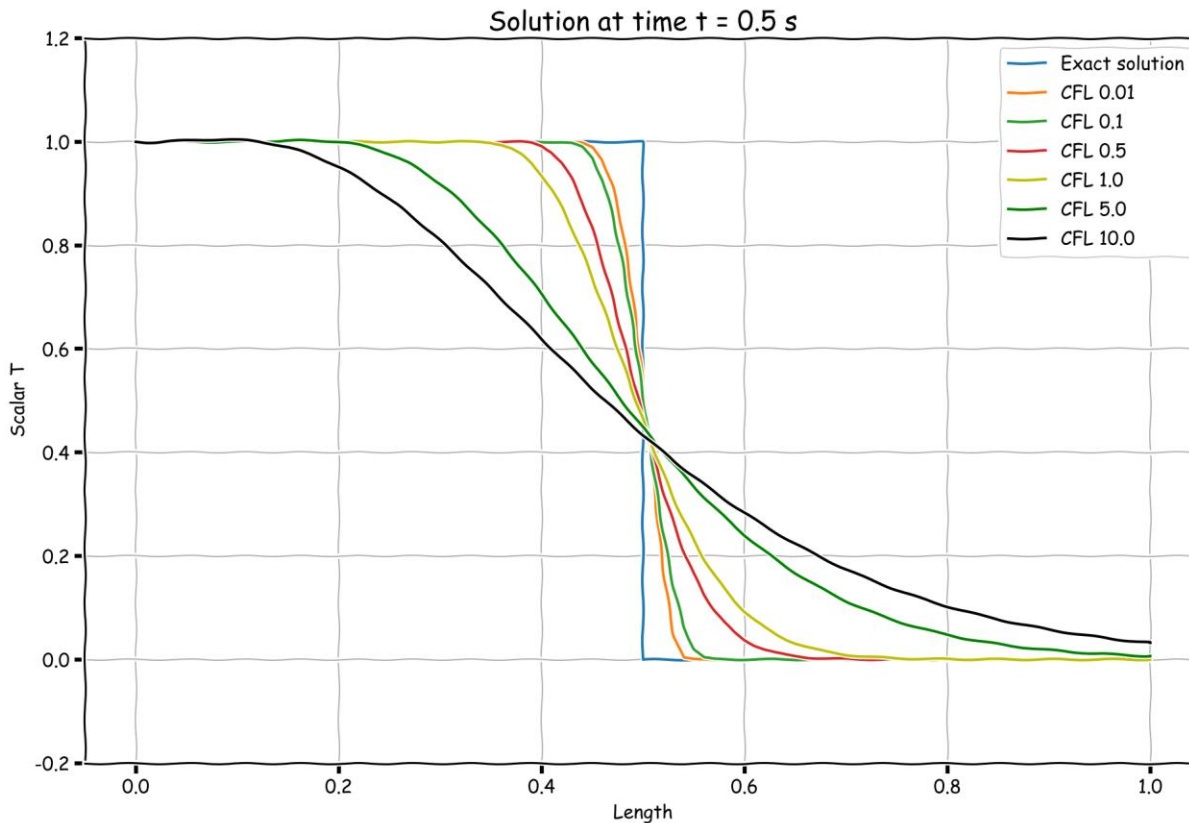
← Simulation time

← One PIMPLE iteration (outer loop), this is equivalent to PISO

← CPU time and wall clock

# The FVM in OpenFOAM

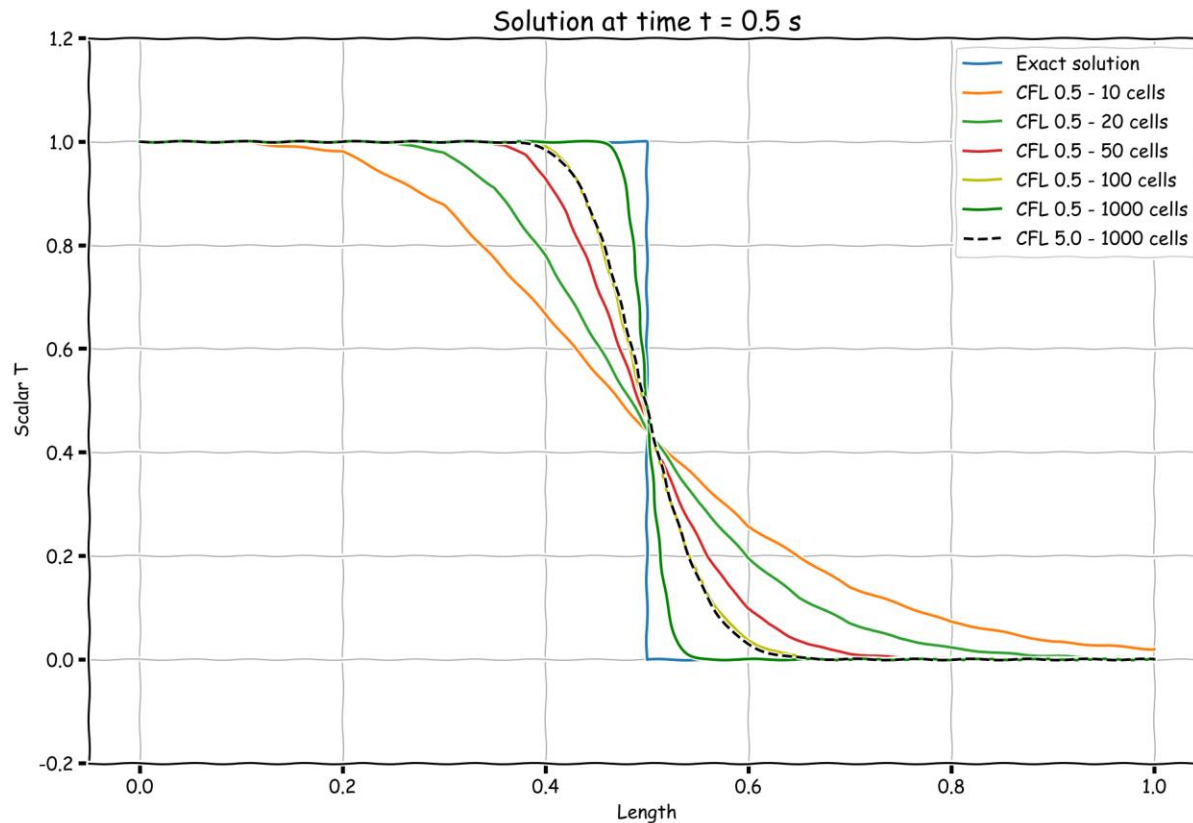
Comparison of different time-step size (CFL number) for a model problem.  
Linear upwind in space – cellMDLimited Gauss linear – Euler in time – 100 cells



$$CFL = \frac{u \Delta t}{\Delta x}$$

# The FVM in OpenFOAM

Comparison of different mesh sizes for a model problem.  
Linear upwind in space – cellMDLimited Gauss linear – Euler in time



$$CFL = \frac{u \Delta t}{\Delta x}$$



# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- ~~2. The Finite Volume Method: An overview~~
- ~~3. Navier-Stokes equations and pressure-velocity coupling~~
- ~~4. On the CFL number~~
- ~~5. Unsteady and steady simulations~~
- ~~6. Understanding the residuals~~
- ~~7. Boundary conditions and initial conditions~~
- ~~8. The FVM in OpenFOAM: some implementation details and computational pointers~~
- 9. Best standard practices – General guidelines**
10. Final remarks

# Best standard practices – General guidelines

- The absolute best practice in CFD is to get a good quality mesh.
- Remember, the mesh is everything in CFD.
- The default quality metrics hardwired in OpenFOAM are fine.

**Max. non-orthogonality = 70**

**Maximum skewness = 4**

- But if you think that they are too conservative, you can rely in the following metrics,

**Max. non-orthogonality = 80**

**Maximum skewness = 8**

- Remember, you will need to adjust the numerical method to take into account the mesh quality.
- If you can not keep the mesh quality metrics to your predefined values, it is better to get a better mesh instead of trying to get a solution by adjusting the numerics.

# Best standard practices – General guidelines

- Regarding the **SIMPLE** loop, you can use the following parameters.
- For good quality meshes it is recommended to do at least one non-orthogonal correction.
- For meshes with a non-orthogonality up to the predefined maximum value you can do 3 non-orthogonal corrections.
- Also, it is strongly recommended to use the consistent formulation (**SIMPLEC**).

```
SIMPLE
```

```
{
```

```
    consistent    yes;
```

```
    nNonOrthogonalCorrectors    3;
```

```
}
```

Enabled/disabled  
consistent formulation of  
the SIMPLE loop

# Best standard practices – General guidelines

- It is recommended to use the following under-relaxation factors (URF) with the **SIMPLE** loop.

## SIMPLE

```
relaxationFactors
{
    fields
    {
        p          0.3;
    }
    equations
    {
        U          0.7;
        k          0.6;
        omega      0.6;
    }
}
```

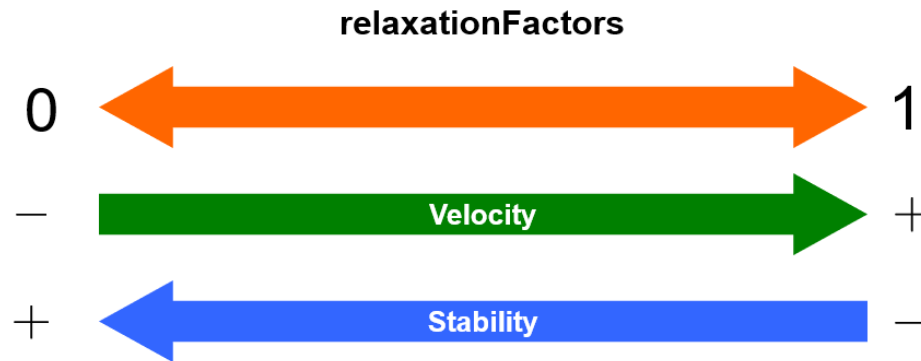
## SIMPLEC

```
relaxationFactors
{
    fields
    {
        p          0.7;
    }
    equations
    {
        U          0.7;
        k          0.7;
        omega      0.7;
    }
}
```

- Usually there is no need to under-relax pressure in the **SIMPLEC** loop; however, it is advisable.
- You can also use URF with the **PIMPLE** family of solvers.
- Notice that our recommended values are a more conservative than the ones you will find in the literature
- If you working with compressible solvers, do not forget to under-relax the thermodynamic variables (**h**, **e**, **rho**, **T**, and so on)

# Best standard practices – General guidelines

- If you do not define the URF factors in the **SIMPLE** sub-dictionary, no URF will be used (disabled).
- If you define the URF equal to 1, under-relaxation will be enabled.
- It is important to define all the URF used by the solver and models involved, if you do not define them, they will be disabled (and it is likely that the solution will diverge).



- To know the URF used by the solver, just look for the word **relax** in the source code (solvers and model libraries).
- If you are using compressible solvers, remember to under-relax density (**rho**).
- It is a good practice to start the solution using low URF and increase then slowly, one term at a time.

# Best standard practices – General guidelines

- It is not compulsory to use URF with transient solvers.
- Nonetheless, using URF with transient solvers will improve stability as they increase diagonal dominance.
- Have in mind that if you use too low URF you will lose time accuracy and add numerical diffusion to the solution.
- Therefore, it is strongly recommended to conduct a time convergence study by using different URF and time-steps values (using prototype cases).
- With transient solvers, we recommend setting all URF equal to 1 or 0.9 (to improve stability).
- You can also use the industry standard values (for extra stability).
- The following URF setups for transient solvers, will improve stability without tampering time accuracy:

URF to ensure diagonal dominance

```
relaxationFactors
{
    fields
    {
        ".*" 1.0;
    }
    equations
    {
        ".*" 1.0;
    }
}
```

URF for **PIMPLE-SIMPLE** formulation

```
relaxationFactors
{
    fields
    {
        "p.*" 0.3;
    }
    equations
    {
        "U.*" 0.7;
        "k.*" 0.7;
        "omega.*" 0.7;
    }
}
```

URF for **PIMPLE-SIMPLEC** formulation

```
relaxationFactors
{
    fields
    {
        "p.*" 0.7;
    }
    equations
    {
        "p.*" 0.7;
        "U.*" 0.7;
        "k.*" 0.7;
        "omega.*" 0.7;
    }
}
```

# Best standard practices – General guidelines

- Regarding the **PISO** loop, you can use the following parameters.
- For good quality meshes it is recommended to do at least one non-orthogonal correction.
- For meshes with a non-orthogonality up to the predefined maximum value you can do 2-3 non-orthogonal corrections.
- For good accuracy and stability, it is recommended to use at least 2 **nCorrectors**.
- However, we recommend to do at least 3 corrections, specially if you are using unstructured meshes (with some degree skewness and non-orthogonality) and you are dealing with complex physics.
- When dealing with complex physics, you can use the optional keyword **momentumPredictor** to enable or disable the momentum predictor step.
- The momentum predictor helps in stabilizing the solution as it computes better approximations for the velocity.
- However, the benefits of this correction are not quite clear.

```
PISO
{
    momentumPredictor    yes;
    nCorrectors           3;
    nNonOrthogonalCorrectors 1;
}
```

# Best standard practices – General guidelines

- For unsteady problems with very complex physics, you can use the **PIMPLE** loop.
- The **PIMPLE** method works very similar to the **PISO** method. In fact, setting the keyword **nOuterCorrectors** to 1 is equivalent to running using the **PISO** method.
- To gain more stability and accuracy, you will need to increase the number outer corrections in the loop (**nOuterCorrectors**). Have in mind that this will highly increase the computational cost.
- The **PIMPLE** method is very stable when using large time-steps or doing pseudo-transient simulation.
- It is also recommended to add under-relaxation to gain more stability.
- The following parameters will give a very robust and accurate loop that you can use with complex physics, while being time accurate.

## PIMPLE

```
{  
    momentumPredictor    yes;  
    nOuterCorrectors 2;  
    nCorrectors 3;  
    nNonOrthogonalCorrectors 1;  
}
```

## relaxationFactors

```
{  
    fields  
    {  
        “.” 0.7; //SIMPLEC URF  
    }  
}
```



# Best standard practices – General guidelines

- In OpenFOAM, the **PISO** and **PIMPLE** methods are formulated for unsteady simulations.
- Whereas the **SIMPLE** and **SIMPLEC** methods are formulated for steady simulations.
- If conserving time is not a priority, you can use the **PIMPLE** method in pseudo transient mode.
- The pseudo transient **PIMPLE** method is more stable than the **SIMPLE** method, but it has a higher computational cost.
- Depending on the method and solver you are using, you will need to define a specific sub-dictionary in the dictionary file *fvSolution*.
- For instance, if you are using the **PISO** method, you will need to specify the **PISO** sub-dictionary.
- And depending on the method, each sub-dictionary will have different entries.
- If you are conducting steady simulations, we recommend to use the **PISO** or **PIMPLE** method with local-time-stepping (**LTS**).
  - This method is more stable than the **SIMPLE** loop for steady solvers.
- However, this accuracy comes with the price of a higher computational cost and the burden of choosing the optimal pseudo time-step parameters (damping and smoothing).

# Best standard practices – General guidelines

- In practice, for gradient computation the **leastSquares** method is more accurate than the **Gauss** method.
- However, we have found that the **leastSquares** method tends to be more oscillatory on tetrahedral meshes.
- To avoid over and under shoots on the gradient computations, it is recommended to always use limiters.
- However, do not add limiters to all variables as doing so might add to much diffusion or might smear the solution.
- For example, it is not recommended to add very aggressive limiters to pressure (**p**), volume-of-fraction (**alpha**), and interface curvature (**nHat**).
- We recommend to use the following limiters for velocity (**U**), and the transported turbulent quantities (**k**, **omega**, **epsilon**, **nut**, and so on),

**cellLimited leastSquares 1.0;**

or

**cellLimited Gauss linear 1.0;**

# Best standard practices – General guidelines

- These are the convective discretization schemes that you will use most of the times:
  - **upwind**: first order accurate.
  - **linearUpwind**: second order accurate, bounded.
  - **linear**: second order accurate, unbounded.
  - A good TVD scheme (**vanLeer** or **Minmod**): TVD, second order accurate, bounded.
  - **limitedLinear**: second order accurate, unbounded, but more stable than pure linear. Recommended for LES simulations (kind of similar to the Fromm method).
- So, you can start using a first order method and then switch to second order accuracy.
- If at any point of the simulation you see oscillations, you can switch to first order and stabilize the solution.
- You can also stabilize the solution by increasing the viscosity.
- **Start with robustness and end with accuracy.**

# Best standard practices – General guidelines

- Remember, the discretization schemes of the diffusive terms are highly related to the mesh quality.
- So, for good quality meshes you can use the following method,

**Gauss linear limited 1;**

- For industrial meshes or meshes where you can not maintain the predefined best quality, you can use the following method,

**Gauss linear limited 0.5;**

- Remember, to enable non-orthogonal corrections you will need to do at least one correction (**nNonOrthogonalCorrectors**).
- However, we recommend to do at least 3 non-orthogonal corrections when the mesh non-orthogonal quality is more than 60.

# Best standard practices – General guidelines

- The fact that we are using an implicit solver (unconditionally stable), does not mean that we can choose a time step of any size.
- The time-step must be chosen in such a way that it resolves the time-dependent features and it maintain the solver stability.
- In our personal experience, we have been able to go up to a  $CFL = 5.0$  while maintaining the accuracy. But to achieve this we had to increase the number of corrector steps and tighten the convergence criteria; and this translates into a higher computational cost.
- We have managed to run with CFL numbers up to 200, but this highly reduces the accuracy of the solution. Also, it is quite difficult to maintain the stability of the solver.
- If you are interested in the unsteadiness of the solution, it is recommended to use a CFL number not larger than 2.0, and if accuracy is the goal (e.g., predicting forces), definitively use a CFL less than 1.0.
- Remember, a smaller time-step may be needed during the first iterations to maintain solver stability.
- Have in mind that the initial time-steps may take longer to converge, do not be alarmed of this behavior, it is perfectly normal.

# Best standard practices – General guidelines

- If you use the first order **Euler** scheme, try to use a CFL number less than 2.0 and preferably in the order of 1.0-0.8, this is in order to keep time diffusion to a minimum.
- In order to speed up the computation and if you are not interested in the initial transient, you can start using a large time-step (high CFL).
- When you are ready to sample the quantity of interest, reduce the time-step to get a CFL less than one and let the simulation run for a while before doing any sampling.
- Our recommendation is to always monitor your solution, and preferably use the **PIMPLE** family of solvers with high under-relaxation values (more than 0.7).
- When using the **PIMPLE** or **PISO** solver (with maximum CFL feature enable), the solver will use as a time-step for the first iteration the value set by the keyword **deltaT** in the *controlDict* dictionary.
  - So, if you set a high value, is likely that the solver will explode.
- Our advice is to set a low **deltaT** and then let the solver gradually adjust the time-step until the desired maximum CFL is reached.
- In multiscale problems (e.g., multiphase flows), it is not recommended to use adjustable time-step. It is strongly recommended to fix the tie-step.

# Best standard practices – General guidelines

- Some fancy high-resolution numerical schemes (such as the vanLeer, vanAlbada, superbee, MUSCL, OSPRE – cool names no – ), are hard to start.
  - So, it is better to begin with a small time-step and increase it gradually.
- And by small time-step we mean a time-step that will give you a CFL about 0.1 - 0.3.
- If you do not know what time discretization scheme to use, go for the **Crank-Nicolson**.

```
ddtSchemes
{
    default    CrankNicolson  $\psi$ ;
}
```

- Setting  $\psi$  equal to 0 is equivalent to running a pure **Euler** scheme (robust but first order accurate).
- By setting the blending factor equal to 1 you use a pure **Crank-Nicolson** (accurate but oscillatory, formally second order accurate).
- Most of the time, a value of 0.7 is a good compromise.

# Best standard practices – General guidelines

- The linear solvers are defined in the *fvSolution* dictionary.
- You will need to define a solver for each variable you are solving.
- Remember, the solvers can be modified on the fly.
- The **GAMG** solver (generalized geometric-algebraic multigrid solver), can often be the optimal choice for solving the pressure equation. However, we have found that when using more than 1000 processors is better to use Newton-Krylov type solvers.
- The selection of the linear solvers is hardware and problem dependent.
- Sometimes the **GAMG** is a good choice, and sometimes the **PCG** or **PBiCGStab** is a better choice.
- Remember, if you use Newton-Krylov type solver, you also need to choose a good preconditioner, which again, is problem and hardware dependent.
- It is often a good idea to set the minimum number of iterations (**minIter**), we like to do at least 3 iterations (for symmetric and asymmetric matrices).
- The utility `renumberMesh` can dramatically increase the speed of the linear solvers, specially during the initial iterations.



# Best standard practices – General guidelines

- Remember, residuals are not a direct indication that you are converging to the right solution. It is always a good idea to also monitor a quantity of interest.
- The first time-steps the solution might not converge, this is acceptable.
- Also, you might need to use a smaller time-step during the first iterations to maintain solver stability.
- If the solution is not converging, try to reduce the time-step size.
- So how do we set the tolerances?
- The pressure equation is particularly important, so we should resolve it accurately. Solving the pressure equation is the expensive part of the whole iterative process.
- For the pressure equation (symmetric matrix), you can start the simulation with a **tolerance** equal to **1e-6** and **relTol** equal to **0.01**. After a while you change these values to **1e-6** and **0.0**, respectively.
- If the linear solver is taking too much time, you can change the convergence criterion to **1e-4** and **relTol** equal to **0.05**. You usually will do this during the first iterations.
- For the velocity field (**U**) and the transported quantities (asymmetric matrices), you can use an absolute tolerance equal to **1e-8** and relative tolerance equal to **0**. Solving for these variables is relatively inexpensive, so you can start right away with a tight tolerance

# Best standard practices – General guidelines

- We usually start by using a not so tight convergence criterion and as the simulation runs, we can tight the convergence criterion.
- By proceeding in this way, you can put all the computational effort only in last corrector step (**.\*Final**).
- For all the intermediate corrector steps, you can use a more relaxed convergence criterion.
- For example, you can use the following solver and tolerance criterion for all the intermediate corrector steps (**p**), then in the final corrector step (**pFinal**) you tight the solver tolerance.

## Loose tolerance for p

```
p
{
    solver          PCG;
    preconditioner  DIC;
    tolerance       1e-4;
    relTol          0.05;
}
```

## Tight tolerance for pFinal

```
pFinal
{
    solver          PCG;
    preconditioner  DIC;
    tolerance       1e-6;
    relTol          0.0;
}
```

# Best standard practices – General guidelines

- Some additional comments about the *fvSolution* dictionary.

momentumPredictor	yes;	Set to yes for high Reynolds flows, where convection dominates (default value is yes)
nOuterCorrectors	1;	Recommended value is 1 (equivalent to PISO). Increase to improve the stability of second order time discretization schemes (LES, complex physics, combustion). Increase for strongly coupled problems.
nCorrector	3;	Recommended to use at least 3 correctors. It improves accuracy and stability. Use 4 or more for highly transient flows or strongly coupled problems.
nNonOrthogonalCorrectors	1;	Recommend to use at least 1 corrector (even in good quality meshes). Increase the value for bad quality meshes.
turbOnFinalIterOnly	false;	Flag to indicate whether to solve the turbulence on the final pimple iteration only. For SRS simulations the recommended value is false (the default value is true)

# Best standard practices – General guidelines

- Some additional comments about the *fvSolution* dictionary.
- These options are related to multiphase solvers.

"alpha.*"			←	Field name of the volume of fraction or phase
{				
MULESCorr	yes;	←		Turn on/off semi-implicit MULES
nAlphaSubCycles	1;	←		For semi-implicit MULES use 1 or 2 if you are using large time-steps. Use 3 or more for explicit MULES.
nAlphaCorr	2;	←		Number of VOF corrections. Use 2-3 for slowly varying flows. Use 3 or more for highly transient, high Reynolds, high CFL number flows.
nLimiterIter	10;	←		Number of iterations to calculate the MULES limiter. Use 3-5 if CFL number is less than 3. Use 5-10 if CFL number is more than 3.
alphaApplyPrevCorr	yes;	←		Use previous time corrector as initial estimate. Set to yes for slowly varying flows. Set to no for highly transient flows.
...				
}				

- The semi-implicit MULES offers significant speed-up and stability over the explicit MULES

# Best standard practices – General guidelines

- If you are doing LES or DES, remember that these models are intrinsically 3D and unsteady.
- In LES you should choose your time-step in such a way to get a CFL of less than 1 and preferably of about 0.5 for LES.
- DES simulations can use larger CFL values (up to 5 for reasonable accuracy).
- If you are doing RANS, it is perfectly fine to use upwind to discretize the turbulence closure equations. After all, turbulence is a dissipative process. However, some authors may disagree with this, make your own conclusions.
- On the other hand, if you are doing LES you should keep numerical diffusion to the minimum, so you should use second order methods.
- LES methods can be sensitive to mesh element type; it is highly recommended to use hexahedral meshes.
- Avoid the use of adaptive time-stepping and adaptive save intervals, as they may introduce oscillations in your solution.
- Many times, RANS simulation can mysteriously explode, if this is your case, try reducing the URF of the turbulent quantities to something around 0.5 or even less.

# Best standard practices – General guidelines

- Determining when the solution is converged can be difficult, especially if you are new to CFD.
- Solutions can be considered converged when the flow field and scalar fields are no longer changing, but usually this is not the case for unsteady flows.
- Most flows in nature and industrial applications are highly unsteady.
- The fact that the initial residuals are not falling in a monotonic way, is not an indication of a problem. This is just telling you that the solution is unsteady.
- The final residuals will reach the tolerance criterion in each iteration, but the flow field may be continuously changing from instant to instant.
- In order to properly assess convergence, it is also recommended to monitor a physical quantity.
- If this physical quantity does not change in time, you may say that the solution is converge.
- But be careful, it can be the case that the monitored quantity exhibit a random oscillatory behavior or a periodic oscillatory behavior.
  - In the former case you are in the presence of a highly unsteady and turbulent flow with no periodic behavior.
  - In latter case, you may say that you have reached a converged periodic unsteady solution.

# Best standard practices – General guidelines

- Remember, for unsteady flows you will need to analyze/sample the solution in a given time window. Do not take the last saved solution as the final answer.
- If your goal is predicting forces (e.g., drag prediction), you should monitor the forces and use them as your convergence criterion.
- The convergence depends on the mesh quality, so a **good quality mesh** means faster convergence and accurate solution.
- In general, overall mass balance should be satisfied.
- Remember, the method is conservative. What goes in, goes out (unless you have source terms).
- However, the method is not bounded, so to avoid spurious oscillations remember to use limiters.
- Residuals are not your solution. Low residuals do not automatically mean a correct solution, and high residuals do not automatically mean a wrong solution.
- Initial residuals are often higher with higher order discretization schemes than with first order discretization. That does not mean that the first order solution is better.
- Always ensure proper convergence before using a solution. A not converged solution can be misleading when interpreting the results.

# Best standard practices – General guidelines

- We highly recommend to always monitor the minimum and maximum values of the field variables.
- For us this is the best indication of stability and accuracy.
- If at one point of the simulation velocity is higher than the speed of light (believe us, that can happen), you know that something is wrong.
- And believe us, with upwind you can get solutions at velocities higher than the speed of light.
- You can also see an oscillatory behavior of the minimum and maximum values of the monitored quantity. This is also an indication that something is going bananas.
- For some variables (e.g., volume fraction), the method must be bounded.
- This means that the values should not exceed some predefined minimum or maximum value (usually 0 and 1).
- So, if your solution is oscillating, you can switch to upwind (first order accuracy), stabilize the solution, and then go back to second order accuracy.
- However, it is not guaranteed that this trick will always work.



# Best standard practices – General guidelines

```
ddtSchemes
{
    default          CrankNicolson 0;
}
gradSchemes
{
    default          cellLimited Gauss linear 0.5;
    grad(U)          cellLimited Gauss linear 1;
}
divSchemes
{
    default          none;
    div(phi,U)       Gauss linearUpwind grad(U);
    div(phi,omega)    Gauss linearUpwind default;
    div(phi,k)        Gauss linearUpwind default;
    div((nuEff*dev(T(grad(U)))) Gauss linear;
}
laplacianSchemes
{
    default          Gauss linear limited 1;
}
interpolationSchemes
{
    default          linear;
}
snGradSchemes
{
    default          limited 1;
}
```

- For most cases, we recommend the following discretization method.
- It is very similar to the default method you will find in commercial solvers.
- In overall, this setup is second order accurate and fully bounded.
- According to the quality of your mesh, you will need to change the blending factor of the **laplacianSchemes** and **snGradSchemes** keywords.
- To time diffusion to a minimum, use a CFL number less than 2.
- If during the simulation the turbulence quantities become unbounded, you can safely change the discretization scheme to upwind.
- For gradient discretization the **leastSquares** method is more accurate. But we have found that it is a little bit oscillatory in tetrahedral meshes.

# Best standard practices – General guidelines

```
ddtSchemes
{
    default Euler;
}
gradSchemes
{
    default cellLimited Gauss linear 0.5;
    grad(U) cellLimited Gauss linear 1;
}
divSchemes
{
    default none;
    div(phi,U) Gauss upwind;
    div(phi,omega) Gauss upwind;
    div(phi,k) Gauss upwind;
    div((nuEff*dev(T(grad(U)))) Gauss linear;
}
laplacianSchemes
{
    default Gauss linear limited 0.5;
}
interpolationSchemes
{
    default linear;
}
snGradSchemes
{
    default limited 0.5;
}
```

- If your simulation is mysteriously crashing, you can use the following discretization method.
- This setup is very stable but too diffusive.
- It is first order in space and time.
- Use this discretization method together with the **PISO** method, set **nCorrectors 5**, and **nNonOrthogonalCorrectors 3**.
- Also, be sure to use a high value of viscosity.
- Use Newton-Krylov type linear solvers (do not use multigrid), and set the minimum number of iterations to 5 for all variables.
- Use a CFL number in the order of 0.5.
- If your simulation continues to crash using this method, you better check your boundary conditions, physical properties, or the physical models involved.
- Maybe there is something incompatible in your setup.

# Roadmap

- ~~1. CFD and Multiphysics simulations~~
- ~~2. The Finite Volume Method: An overview~~
- ~~3. Navier-Stokes equations and pressure-velocity coupling~~
- ~~4. On the CFL number~~
- ~~5. Unsteady and steady simulations~~
- ~~6. Understanding the residuals~~
- ~~7. Boundary conditions and initial conditions~~
- ~~8. The FVM in OpenFOAM: some implementation details and computational pointers~~
- ~~9. Best standard practices – General guidelines~~
- 10. Final remarks**

# Final remarks



- **Some kind of conclusion,**
  - **Good mesh – good results.**
  - **Start robustly and end with accuracy.**
  - **Stability, accuracy and boundedness, play by these terms and you will succeed.**
  - **Do not sacrifice accuracy and stability over computing speed.**

## Final remarks

**That was only the tip of the iceberg**



**Now the rest is on you**

# Final remarks

## Some FVM/CFD references

- There is vast amount of literature in the field of FVM/CFD and numerical analysis. We will give you some of our favorite references, which are closed related to what you will find in OpenFOAM.
  - **The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction With OpenFOAM and Matlab**  
F. Moukalled, L. Mangani, M. Darwish. 2015, Springer-Verlag
  - **Finite Volume Methods for Hyperbolic Problems**  
R. Leveque. 2002, Cambridge University Press
  - **Computational Gasdynamics**  
C. Laney. 1998, Cambridge University Press
  - **Computational Techniques for Multiphase Flows**  
G. H. Yeoh, J. Tu. 2009, Butterworth-Heinemann
  - **An Introduction to Computational Fluid Dynamics**  
H. K. Versteeg, W. Malalasekera. 2007, Prentice Hall
  - **Computational Fluid Dynamics: Principles and Applications**  
J. Blazek. 2015, Butterworth-Heinemann.
  - **Computational Methods for Fluid Dynamics**  
J. H. Ferziger, M. Peric. 2001, Springer
  - **Numerical Heat Transfer and Fluid Flow**  
S. Patankar. 1980, Taylor & Francis
  - **Numerical Methods for Partial Differential Equations: Finite Difference and Finite Volume Methods**  
S. Mazumder. 2015, Academic Press.
  - **Iterative Methods for Sparse Linear Systems**  
Y. Saad. 2003, SIAM.

# Final remarks

## Some FVM/CFD references

- There is vast amount of literature in the field of FVM/CFD and numerical analysis. We will give you some of our favorite references, which are closed related to what you will find in OpenFOAM.
  - **Matrix analysis and applied linear algebra**  
C. D. Meyer. 2010, SIAM.
  - **A Finite Volume Method for the Prediction of Three-Dimensional Fluid Flow in Complex Ducts**  
M. Peric. PhD Thesis. 1985. Imperial College, London
  - **Error analysis and estimation in the Finite Volume method with applications to fluid flows**  
H. Jasak. PhD Thesis. 1996. Imperial College, London
  - **Computational fluid dynamics of dispersed two-phase flows at high phase fractions**  
H. Rusche. PhD Thesis. 2002. Imperial College, London
  - **High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws**  
P. K. Sweby. SIAM Journal on Numerical Analysis, Vol. 21, No. 5, pp. 995-1011, 1984.
  - **A Pressure-Based Method for Unstructured Meshes**  
S. R. Mathur, J. Y. Murthy. Numer. Heat Transfer, Vol. 31, pp. 195-216, 1997.
  - **A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows**  
S. V. Patankar, D. B. Spalding. Int. J. Heat Mass Transfer, 15, pp. 1787-1806, 1972.
  - **Solution of the implicitly discretized fluid flow equations by operator-splitting**  
R. I. Issa. J. Comput. Phys., 62, pp. 40-65, 1985.
  - **Further discussion of numerical errors in CFD**  
J. H. Ferziger, M. Peric. Int. J. Numer. Methods in Fluids, Vol. 23, pp. 1263-1274, 1996.
  - **Limiters for Unstructured Higher-Order Accurate Solutions of the Euler Equations**  
K. Michalak, C. Ollivier-Gooch. 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, USA, 2008.

# Final remarks

## CFD best practices guidelines

- ERCOFTAC best practice guidelines (aerospace CFD, automotive CFD, turbomachinery CFD, electronic cooling CFD, heat transfer).
- NAFEMS best practice guidelines.
- MARNET CFD best practice guidelines for marine applications of CFD.
- NPARC alliance CFD verification and validation archive.
- NASA Turbulence Modeling Resource.
- ERCOFTAC classic collection database for validation and verification.
- NASA CFL3D documentation and validation cases.
- Documentation of commercial CFD solver (e.g., Ansys Fluent, Ansys CFX, Star-CCM+, NUMECA, and so on).
  
- **Verification and validation in computational science and engineering**  
P. J. Roache, Hermosa Publishers
- **Verification and Validation in Scientific Computing**  
W. L. Oberkampf , C. J. Roy, Cambridge University Press.



# Thank you for your attention

- We hope you have found this training useful and we hope to see you in one of our advanced training sessions:
  - OpenFOAM® – Multiphase flows
  - OpenFOAM® – Naval applications
  - OpenFOAM® – Turbulence Modeling
  - OpenFOAM® – Compressible flows, heat transfer, and conjugate heat transfer
  - OpenFOAM® – Advanced meshing
  - DAKOTA – Optimization methods and code coupling
  - Python – Programming, data visualization, and exploratory data analysis
  - Python and R – Data science and big data
  - ParaView – Advanced scientific visualization and python scripting
  - And many more available on request
- Besides consulting services, we also offer '**Mentoring Days**' which are days of one-on-one coaching and mentoring on your specific problem.
- For more information, ask your trainer, or visit our website  
<http://www.wolfdynamics.com/>

**Be collaborative, be innovative, be cloud**



**wolf**dynamics  
multiphysics simulations,  
optimization & data analytics

**Let's connect**



[guerrero@wolfdynamics.com](mailto:guerrero@wolfdynamics.com)



[www.wolfdynamics.com](http://www.wolfdynamics.com)