OpenFOAM® Beginner training session

Presented at the 15th OpenFOAM workshop.
June 22-26, 2020. Arlington, VA, USA.
http://www.cpe.vt.edu/ofw15/

A Crash Introduction to the Finite Volume Method and Discretization Schemes in OpenFOAM®

wolf dynamics

Copyright and disclaimer

Revision 1-2020
Joel Guerrero

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 indicate 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.

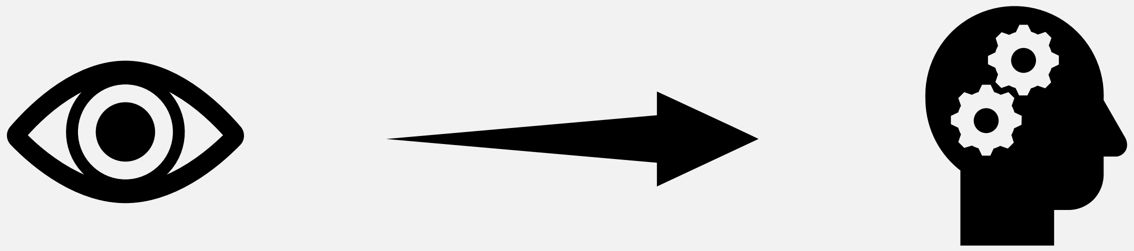
Before we begin

On the training material

- This training is based on OpenFOAM® version 7.
- You can extract the training material wherever you want. From now on, this directory will become:
  - **`$TM`**
- To uncompress the training material go to the directory where you copied it and then type in the terminal,
  - `$> tar -zxvf file_name.tar.gz`
- In every single tutorial, you will find the file `README.FIRST`. In this file you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example,
  - `$> sh run_solver.`
- If you are a beginner, we highly recommend to open the `README.FIRST` file 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®, use the automatic scripts to run the cases.

Before we begin

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



Roadmap

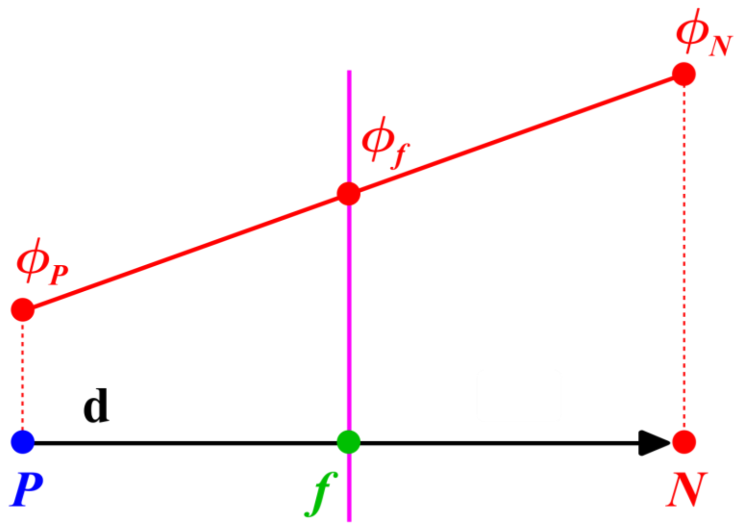
1. Important concepts to remember
2. The Finite Volume Method: An overview
3. The FVM in OpenFOAM®: some implementation details and computational pointers
4. Some kind of conclusion
5. What else we did not cover?
6. Goodbye

Roadmap

1. **Important concepts to remember**
2. The Finite Volume Method: An overview
3. The FVM in OpenFOAM®: some implementation details and computational pointers
4. Some kind of conclusion
5. What else we did not cover?
6. Goodbye

Important concepts to remember

- Let us recall linear interpolation.
- In reference to the figure below, to find the value of the quantity ϕ in *f*, using the known values of ϕ in *P* and *N*, we can proceed as follows,



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

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

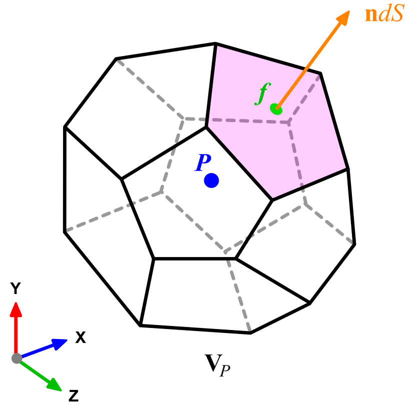
Important concepts to remember

- Let us recall the Gauss theorem (also know as Divergence theorem or Ostrogradsky theorem),

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

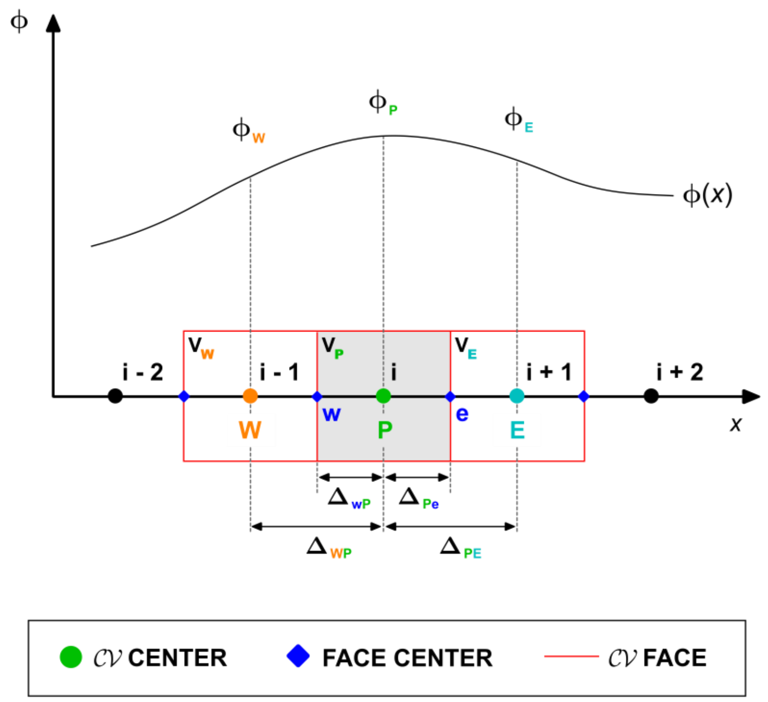
where ∂V_P is a closed surface bounding the control volume V_P and dS represents an infinitesimal surface element with associated normal $\mathbf{n}$ pointing outwards of the surface ∂V_P, and $\mathbf{n}dS = d\mathbf{S}$.

- 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

- 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,

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

- 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 momentum equations,

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

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. ~~Important concepts to remember~~
2. The Finite Volume Method: An overview
3. The FVM in OpenFOAM®: some implementation details and computational pointers
4. Some kind of conclusion
5. What else did not cover?
6. Goodbye

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 ϕ 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.
- Remember, 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,

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

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

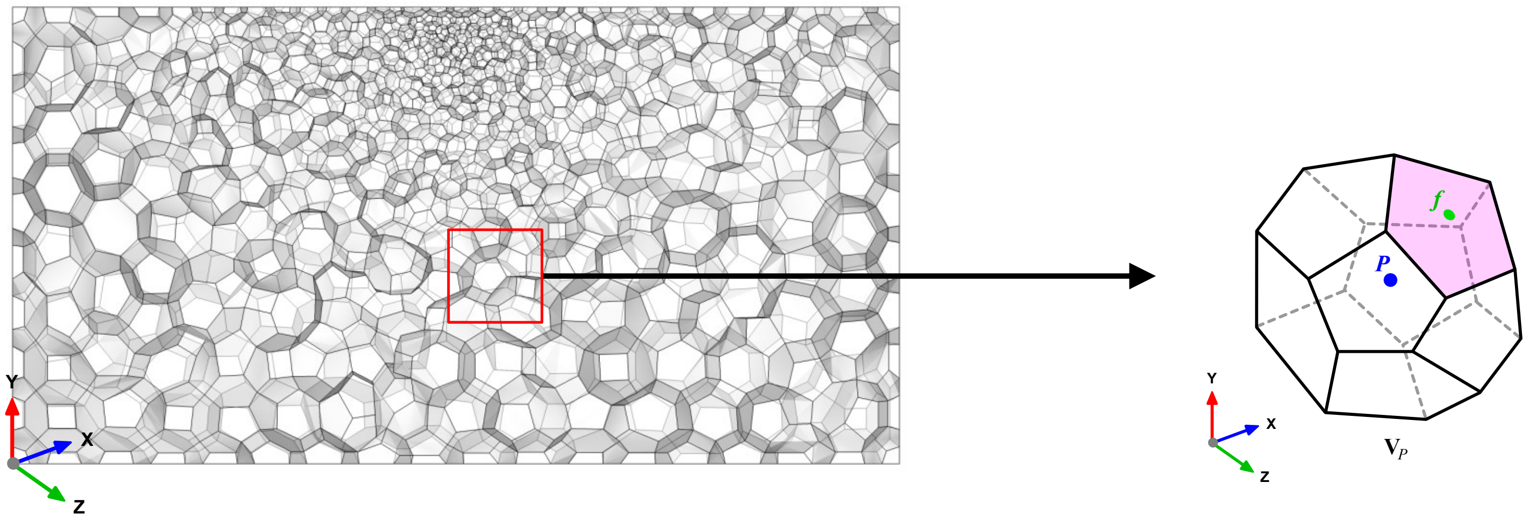
Profile assumptions using Taylor expansions around point *P* (in space) and point *t* (in time)

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.
- The control volumes can be of any shape (*e.g.*, tetrahedrons, hexes, prisms, pyramids, dodecahedrons, and so on).
- The only requirement is that the faces that made up the control volume need to be planar.
- Inside each control volume the solution is sought.
- We also know which control volumes are internal and which control volumes lie on the boundaries.
- 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.



The Finite Volume Method: An overview

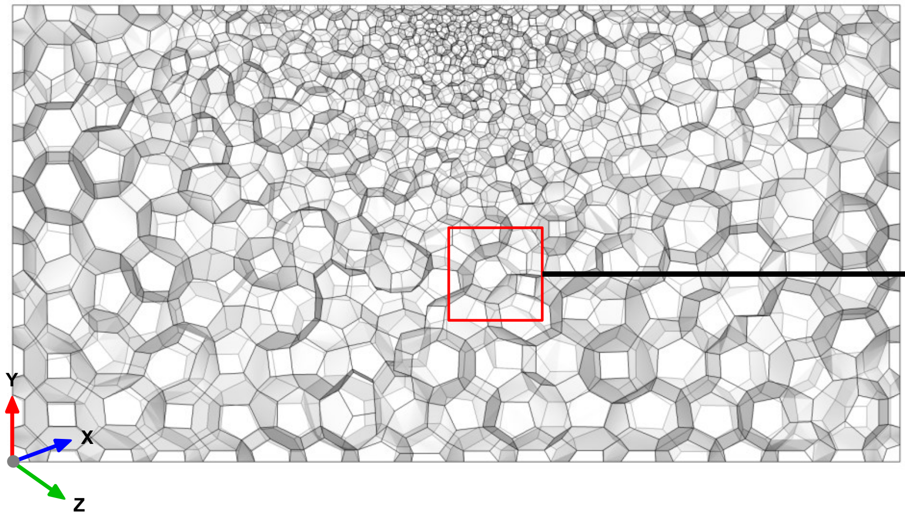
- In the control volume illustrated, the centroid ***P*** and face center ***f*** are known, and computed as,

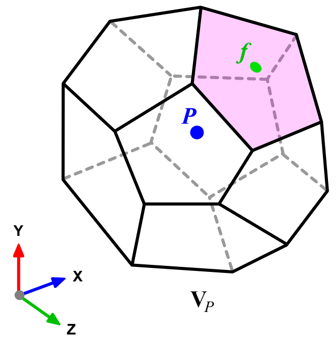
$$\int_{V_P} (\mathbf{x} - \mathbf{x}_P)\, dV = 0 \qquad \int_{S_f} (\mathbf{x} - \mathbf{x}_P) dS = 0$$

- We also assume that the values of all variables are computed and stored in the centroid of the control volume ***V***V_p, this is known as the collocated arrangement.
- We also assume that all variables are represented by a piecewise constant profile (the mean value),

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

- <u>All approximations and assumptions taken so far are at least second order accurate.</u>



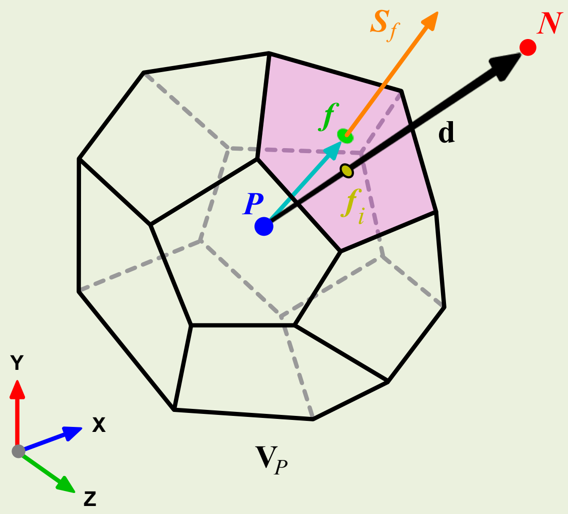


The Finite Volume Method: An overview

- 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 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

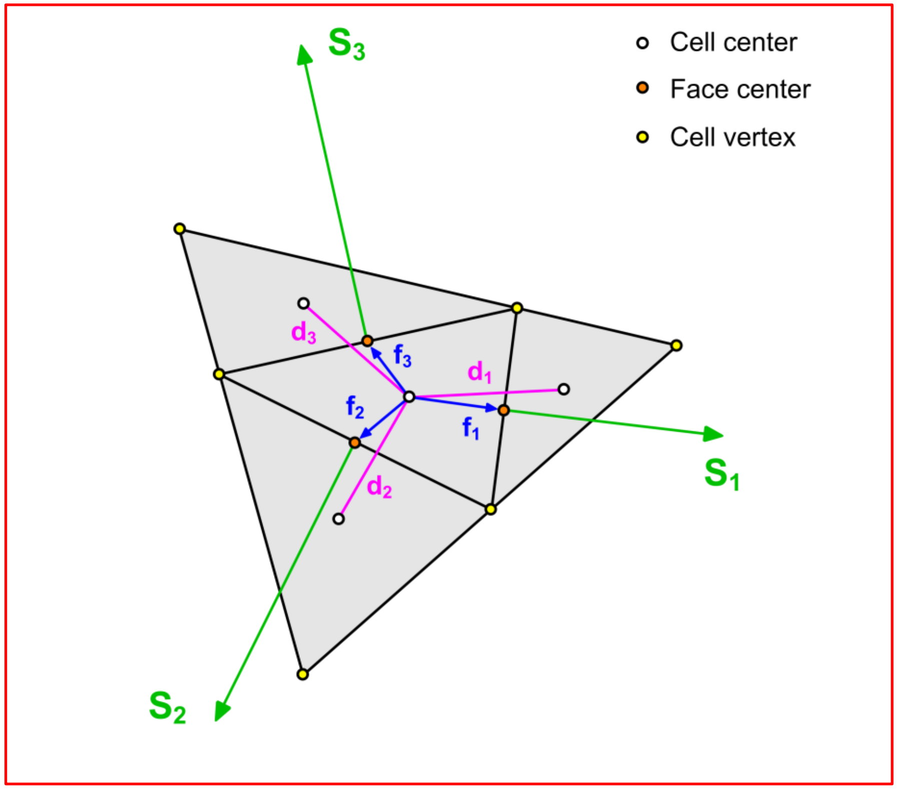
- Have in mind that there are different FVM formulations based on the variable arrangement (*e.g.*, cell centered, node/vertex based).
- Hereafter we will address the cell centered collocated arrangement, which is the one implemented in OpenFOAM® and many commercial CFD software (e.g., Ansys Fluent and 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 explore how to interpolate 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 **S** (vector normal to face passing by the face center) and **d** (vector connecting two cell centers).
- This dependence on the dot product $\mathrm{S} \cdot \mathrm{d}$ is due to the fact that the coefficients contain the following term,

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

- For perfect cells (orthogonal meshes), the dot product is equal to one (there is no deviation between the vectors **S** and **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).



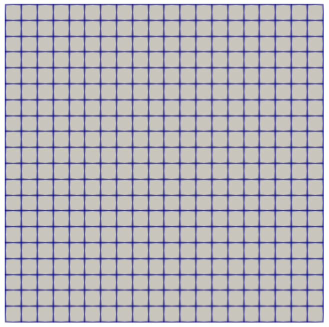
In the figure:

- **S** is the vector normal to face and anchored at the face center
- **d** is the vector connecting two cell centers.
- **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.

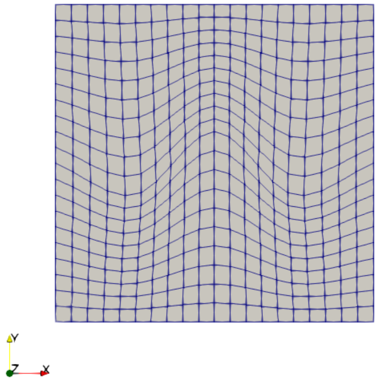
The Finite Volume Method: An overview

- 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.

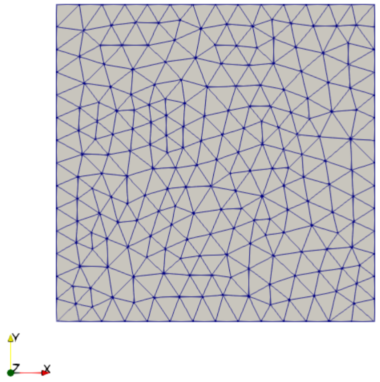
Orthogonal mesh (perfect mesh)

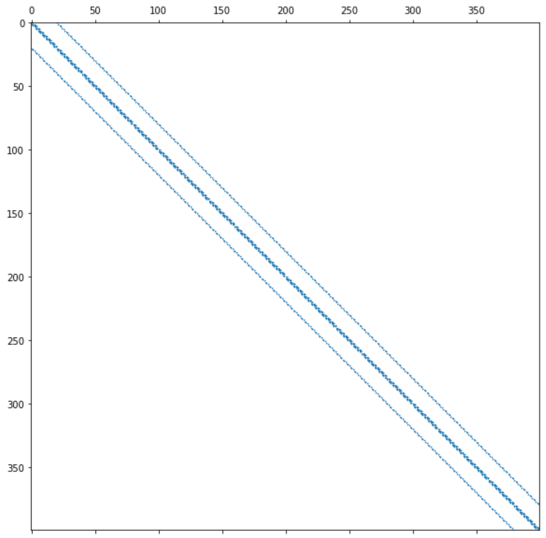


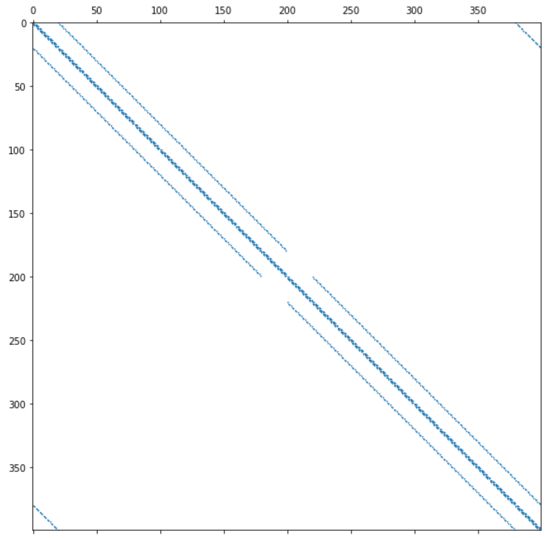
Non-orthogonal mesh

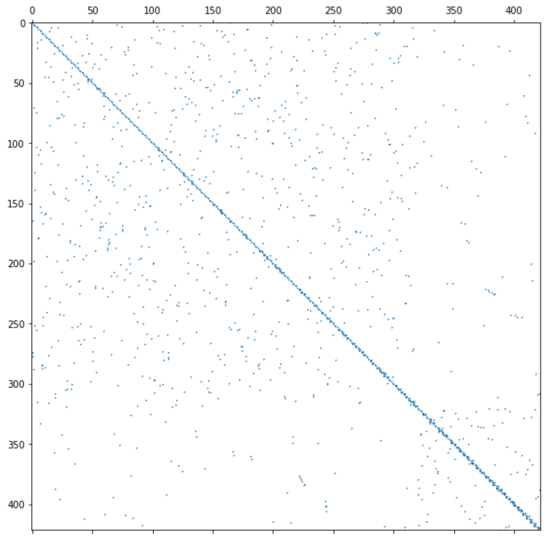


Unstructured triangular mesh









The Finite Volume Method: An overview

Gauss theorem and face fluxes computation

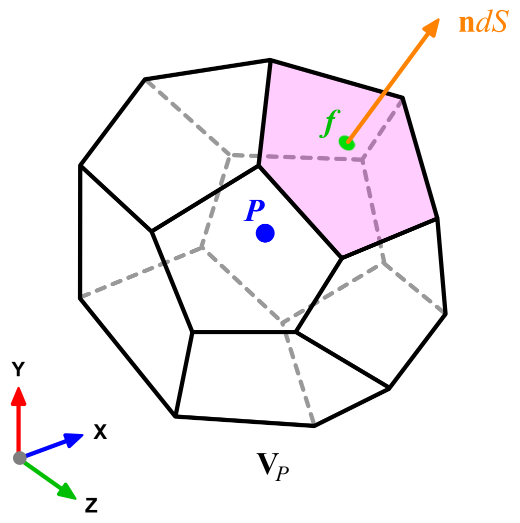
The Finite Volume Method: An overview

- 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 ∂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 ∂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

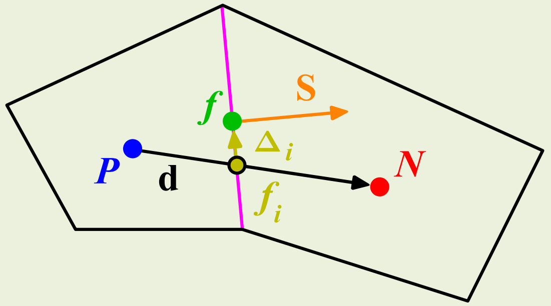
- Let us use the Gauss theorem to convert the volume integrals into surface integrals,

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

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

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

- 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 (Δ_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:

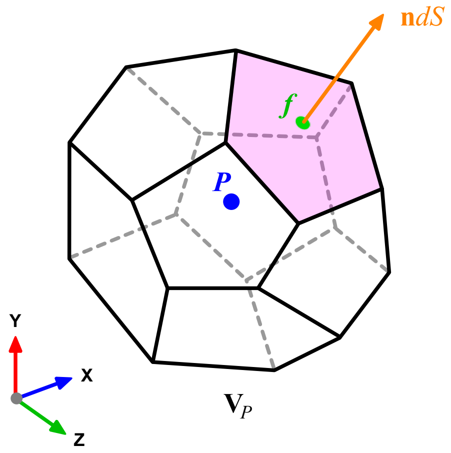
$$\int_{V_P} \underbrace{\nabla \cdot (\rho \mathbf{u} \phi)\, dV}_{\text{convective term}} = \oint_{\partial V_P} \underbrace{d\mathbf{S} \cdot (\rho \mathbf{u} \phi)}_{\text{convective flux}} = \sum_f \int_f d\mathbf{S} \cdot (\rho \mathbf{u} \phi)_f \approx \underbrace{\sum_f \mathbf{S}_f \cdot \left(\overline{\rho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

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:

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



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:

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

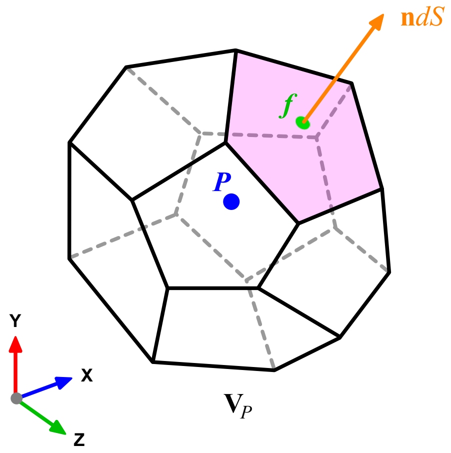
$\downarrow$

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

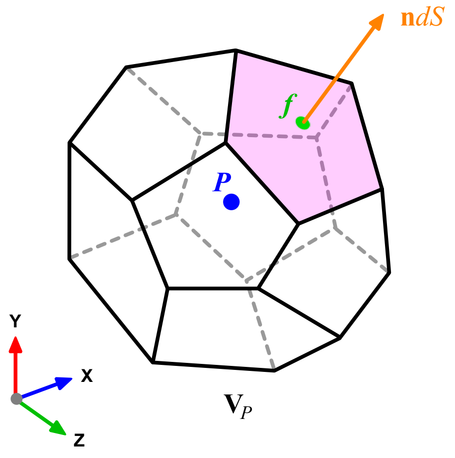
Gradient term:

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

where we have approximated the centroid gradients by using the Gauss theorem.
This method is second order accurate

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.

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

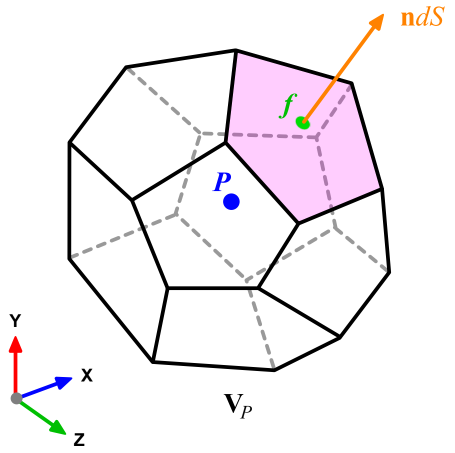
Source term:

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

This approximation is exact if S_ϕ is either constant or varies linearly within the control volume; otherwise is second order accurate.
Sc is the constant part of the source term and ***Sp*** 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:

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

Diffusive term:

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

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

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

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.

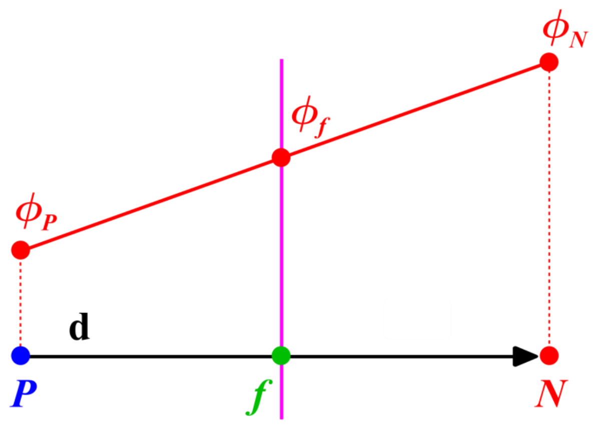
The Finite Volume Method: An overview

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,



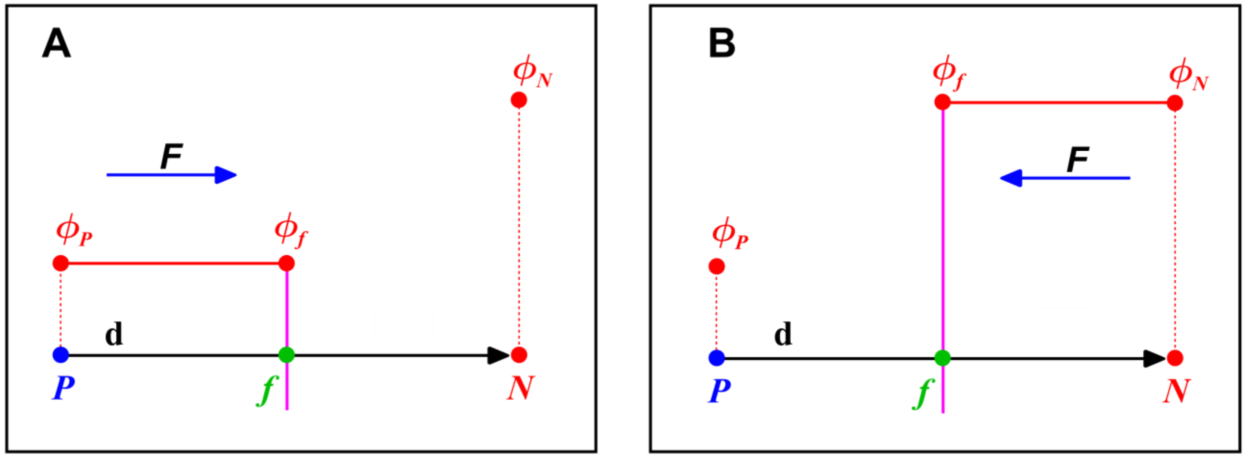
$$\phi_f = f_x \phi_P + (1 - f_x)\,\phi_N \qquad\qquad f_x = \frac{fN}{PN} = \frac{|\,\mathbf{x}_f - \mathbf{x}_N\,|}{|\,\mathbf{d}\,|}$$

- 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,



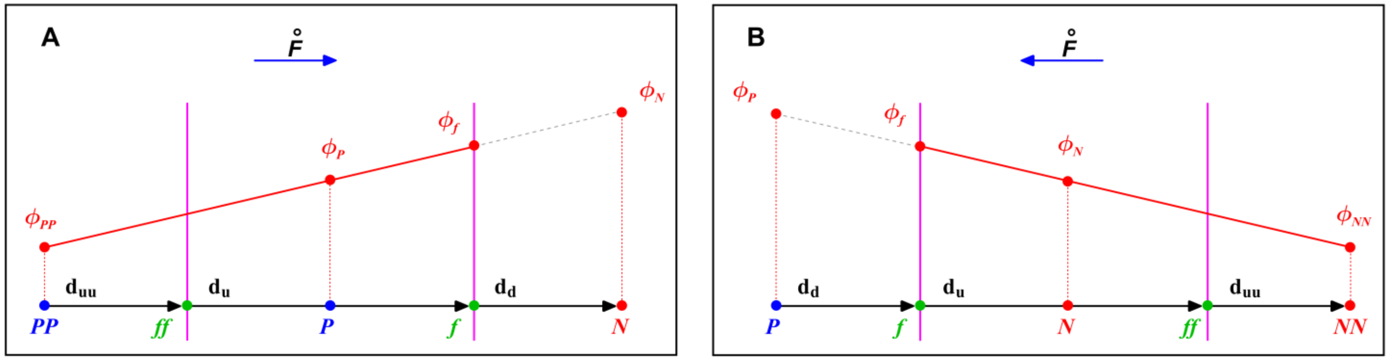
$$\phi_f = \begin{cases} \phi_f = \phi_P & \text{for} \quad \overset{\circ}{F} \geq 0, \\ \phi_f = \phi_N & \text{for} \quad \overset{\circ}{F} < 0. \end{cases}$$

- 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,



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

- 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.

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

- 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

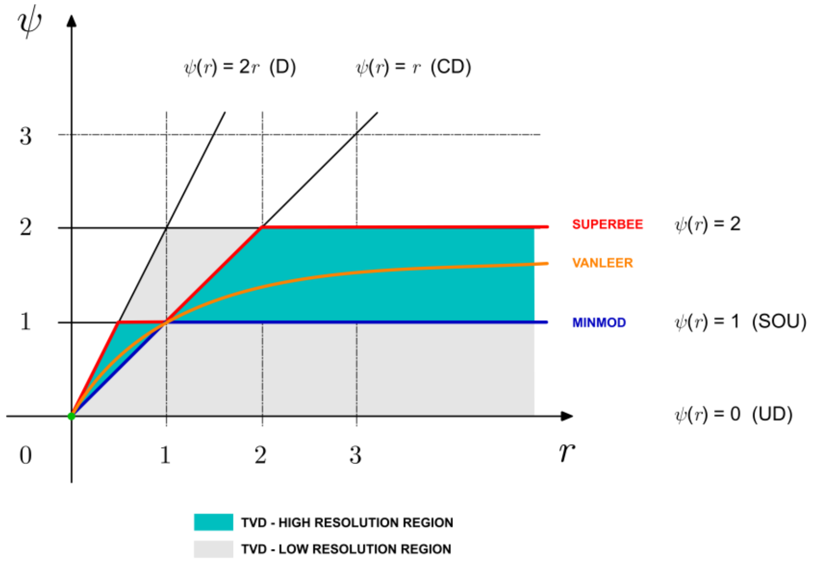
TVD Schemes

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 schemes.
- The Sweby diagram [1], gives the necessary and sufficient conditions for a scheme to be TVD.
- In the figure, the shaded area represents the admissible TVD region. However, not all limiter functions are second order.
- The choice of the limiter function $\psi(r)$ dictates the order of the scheme and its boundedness.
- High-resolution schemes falls in the blue area and low-resolution schemes falls in the grey area.
- The drawback of the limiters is that they reduce the accuracy of the scheme **locally** to first order (low resolution scheme), when *r* < 0 (sharp gradient, opposite slopes or zero gradient). However, this is justified when it serves to suppress oscillations.
- 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.

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



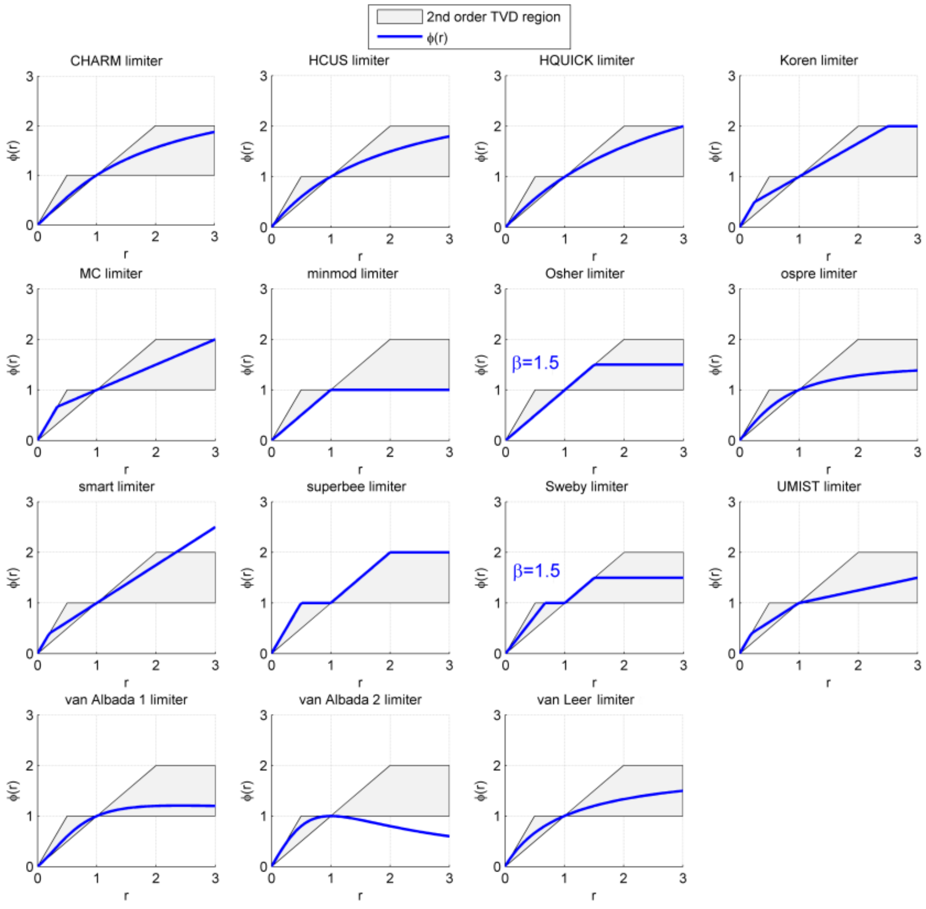
UD = upwind
SOU = second order upwind
CD = central differencing
D = downwind

[1] P. K. Sweby. High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws. SIAM Journal on Numerical Analysis, Vol. 21, No. 5, pp. 995-1011, 1984.

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

minmod (by Roe, 1986)

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

superbee (by Roe, 1986)

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

Non differentiable limiters

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

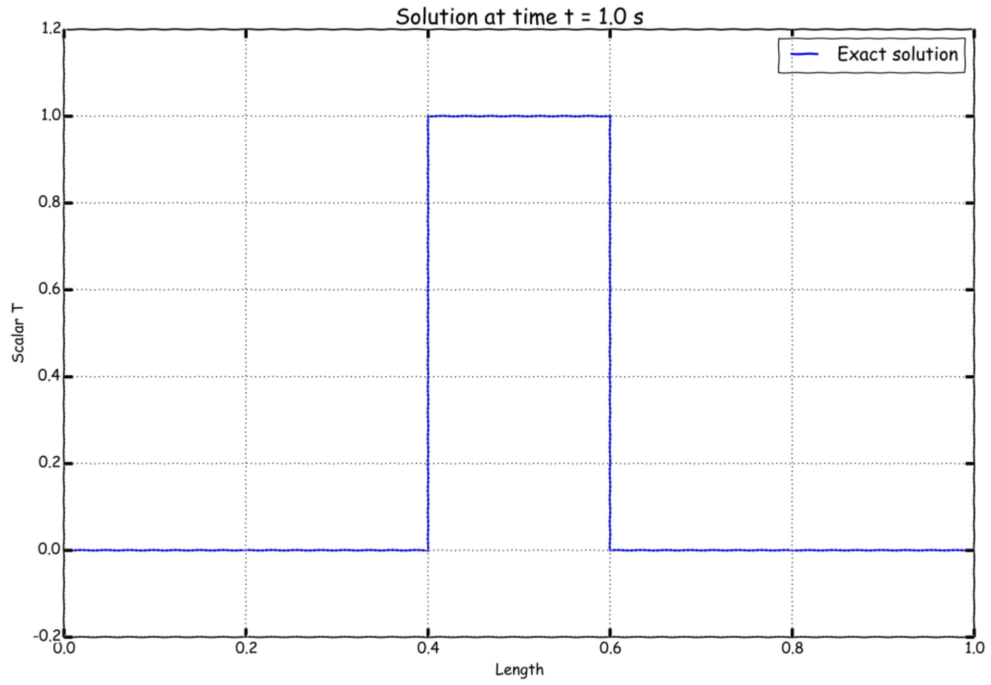
The Finite Volume Method: An overview

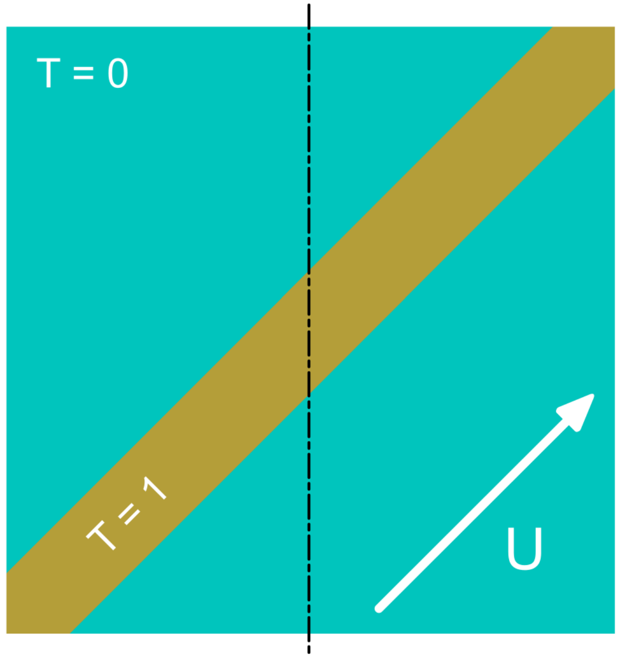
TVD schemes in action – A numerical schemes killer test case

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.



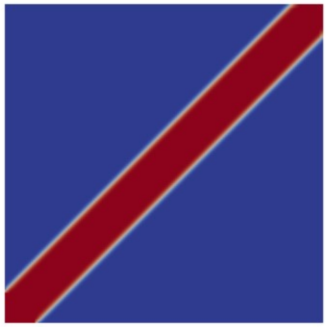


The Finite Volume Method: An overview

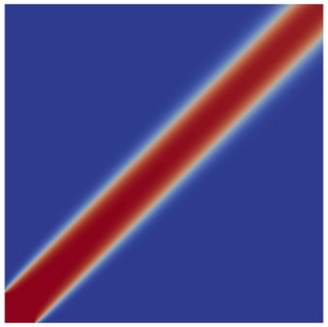
Interpolation of the convective fluxes – Linear and non-linear limiter functions

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

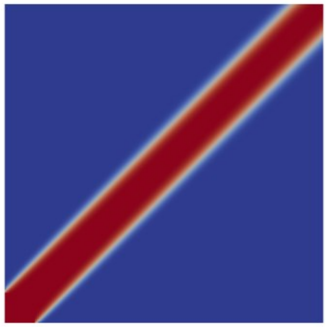
SuperBee - Compressive

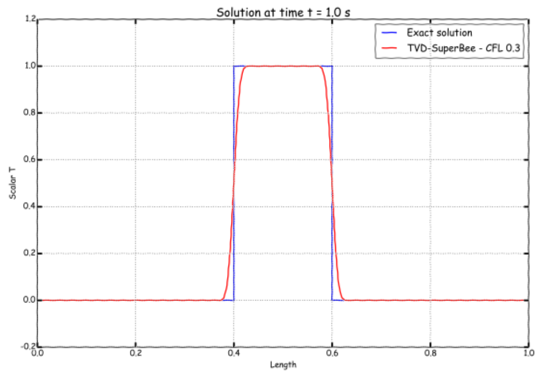


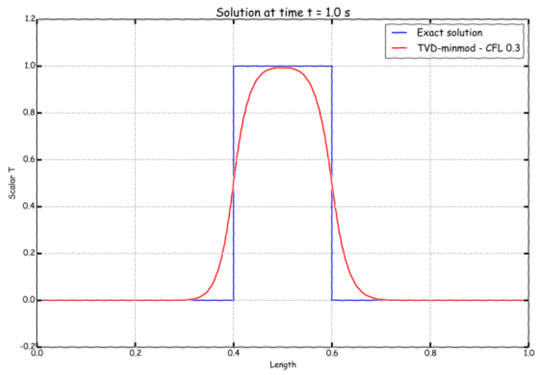
Minmod - Diffusive

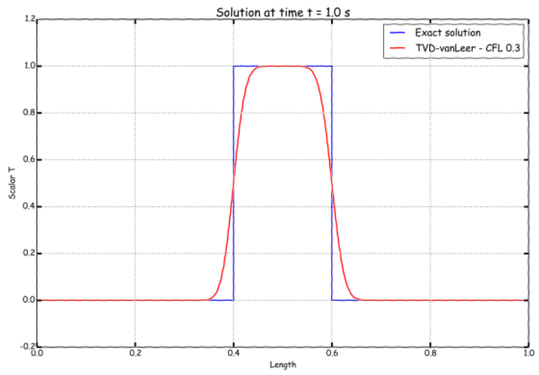


vanLeer - Smooth







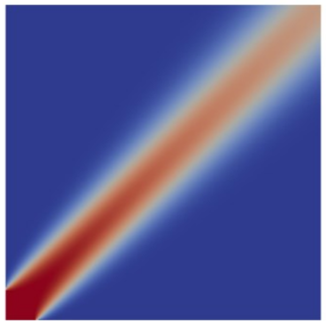


The Finite Volume Method: An overview

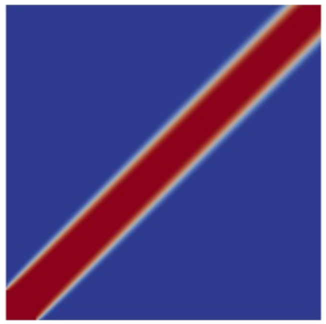
Interpolation of the convective fluxes – Linear and non-linear limiter functions

- Comparison of linear limiters (upwind and linear upwind) and non-linear limiters (SuperBee).
- Recall that the linear upwind method is 2nd order and the upwind method is 1st 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.

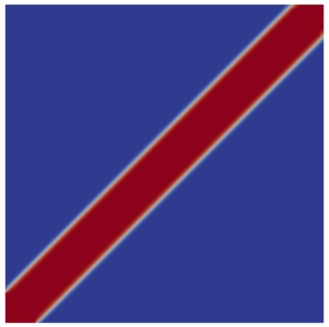
Upwind – 1st order



Linear Upwind – 2nd order



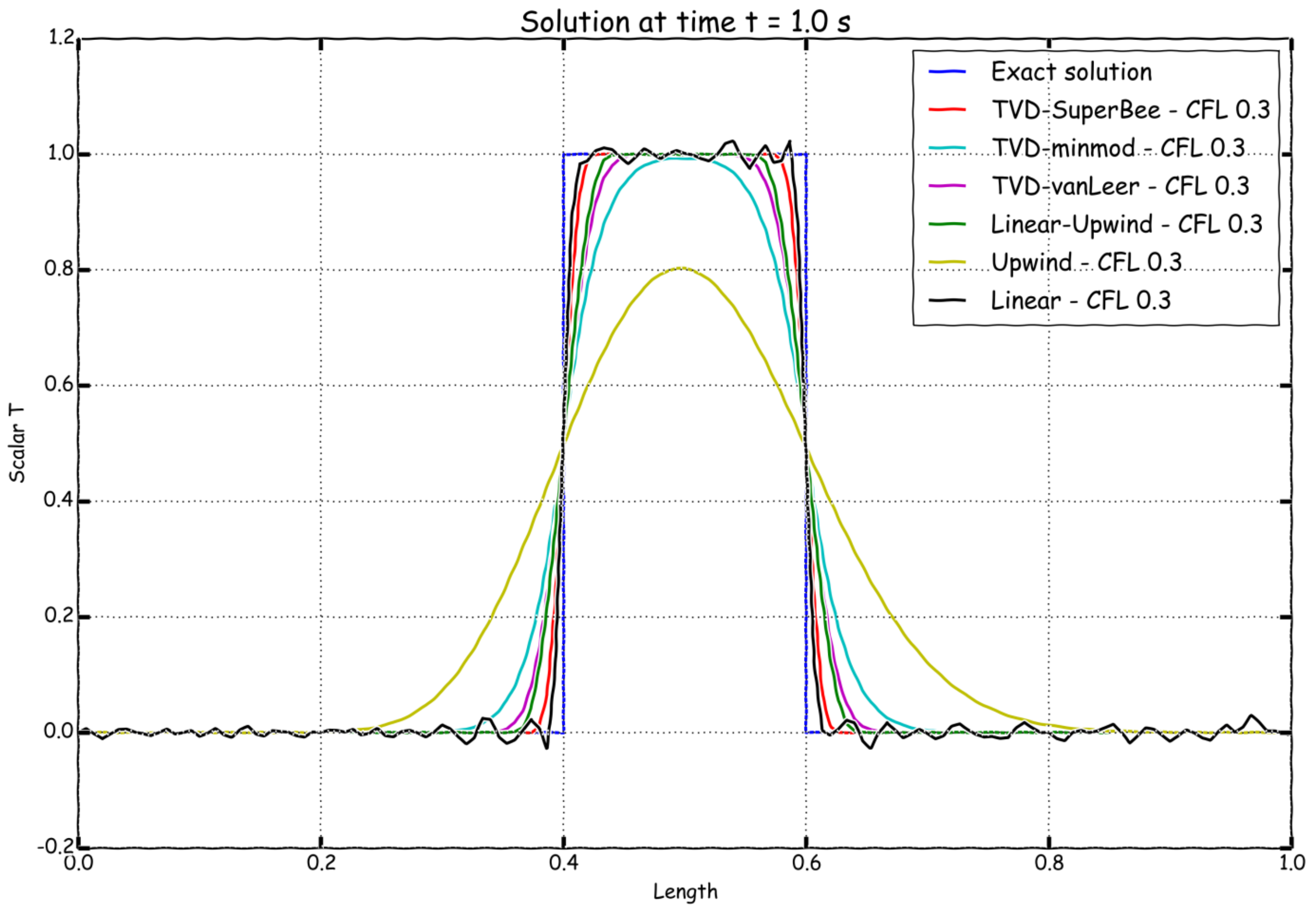
SuperBee – TVD



The Finite Volume Method: An overview

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

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



The Finite Volume Method: An overview

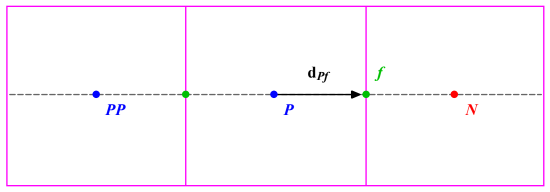
Interpolation of the convective fluxes – Unstructured meshes

The Finite Volume Method: An overview

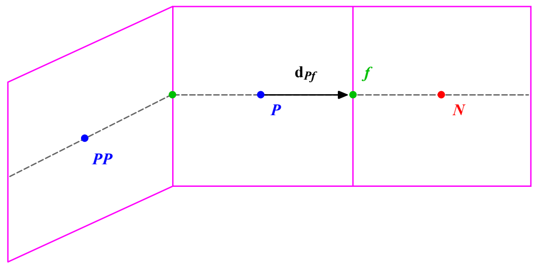
Interpolation of the convective fluxes – Unstructured meshes

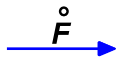
- All the high-order (HO) and high-resolution (HR) schemes we have seen so far, assume line structure (figure A). That is, the cell centers PP, P, and N are all aligned.
- In other words, they are formulated for structured meshes (orthogonal meshes).
- In orthogonal meshes, the cell centers PP, P, and N are all aligned (colinear). Therefore, constructing wide stencils is relative straightforward.
- In unstructured meshes or when the cell centers are not colinear, it is not straightforward to use the previous schemes 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



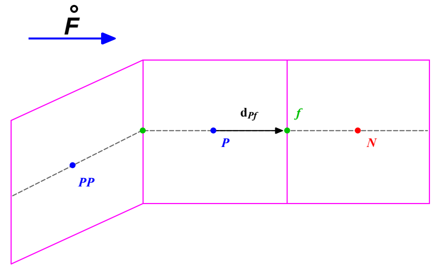
B





The Finite Volume Method: An overview

Interpolation of the convective fluxes – Unstructured meshes



- A simple way around this problem is to redefine HO and HR schemes in terms of gradients at the control volume center P and face center f.
- For example, using the gradient at P and f, we can compute the face values as follows (upwind bias formulations),

Upwind $\rightarrow \quad \phi_f = \phi_P$

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

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

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

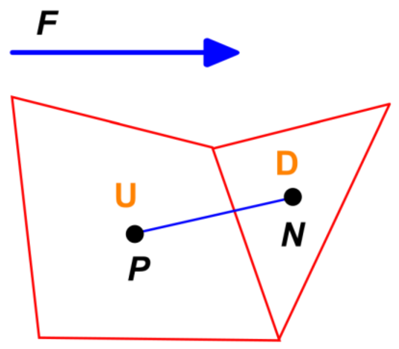
- Notice that in this new formulation the cell PP does not appear anymore.
- The problem now turns in the accurate evaluation of the gradients at the cell and face centers. So if the computation of the gradients is second order accurate, it does not matter the way they are computed.
- For example, the 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.

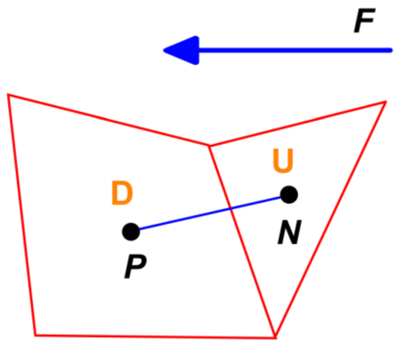
The Finite Volume Method: An overview

Interpolation of the convective fluxes – Unstructured meshes

- In unstructured meshes, as often the value of the node PP (of NN) is not available or straightforward to compute, the ratio of successive gradients *r* can be computed as follows [1],

$$r = \left[\frac{(2\nabla\phi_P \cdot \mathbf{d}_{PN})}{\phi_D - \phi_U} - 1 \right]$$





U → Upwind

D → Downwind

- As you can see, the value of *r* depends on the flow direction.
- There are many ways to compute *r*. This is an area of active research

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

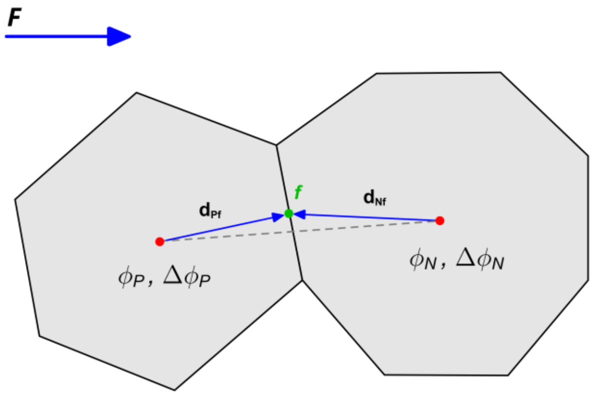
The Finite Volume Method: An overview

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,

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

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



- In the previous relations, ψ_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.

[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 ψ is the flux limiter and ϕ 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.

[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"

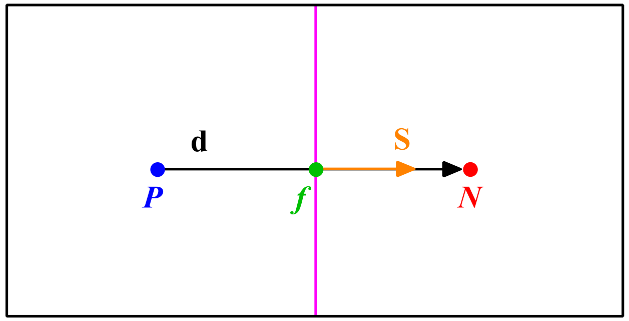
The Finite Volume Method: An overview

Interpolation of diffusive fluxes

The Finite Volume Method: An overview

Interpolation of diffusive fluxes in orthogonal and non-orthogonal meshes

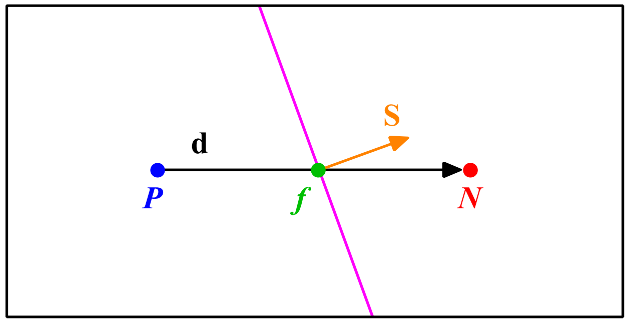
- By looking the figure below, the face values appearing in the diffusive flux in an orthogonal mesh can be computed as follows,



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

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

- By looking the figure below, the face values appearing in the diffusive flux in a non-orthogonal mesh (20°) can be computed as follows,



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

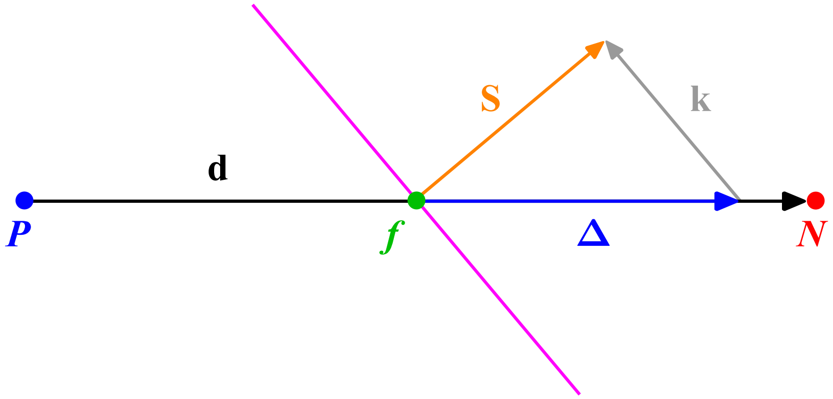
- This type of approximation is second order accurate but involves a larger truncation error. It also uses a larger numerical stencil, which make it less stable.

- Remember, the non-orthogonal angle is the angle between the vector **S** and the vector **d**

The Finite Volume Method: An overview

Correction of diffusive fluxes in non-orthogonal meshes

- By looking the figures below, the face values appearing in the diffusive flux in a non-orthogonal mesh (40°) can be computed as follows.
- Using the over-relaxed approach, the diffusive fluxes can be corrected as follow,



Over-relaxed approach

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

$$\mathbf{S} = \Delta_{\perp} + \mathbf{k}.$$

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

The Finite Volume Method: An overview

Correction of diffusive fluxes in non-orthogonal meshes

- Notice that non-orthogonality gives rise to secondary face gradients that somehow needs to be reconstructed from the cell center.

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

- We presented the over relaxed approach, have in mind that there are many more methods to deal with mesh non-orthogonality, this is an area of active research.
- 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.
- When solving the NSE, non-orthogonality mainly affects the pressure equation, and in the case of compressible flows, it also affects the energy equation.
- From this discussion, it is clear why we want to avoid large non-orthogonal angles.

The Finite Volume Method: An overview

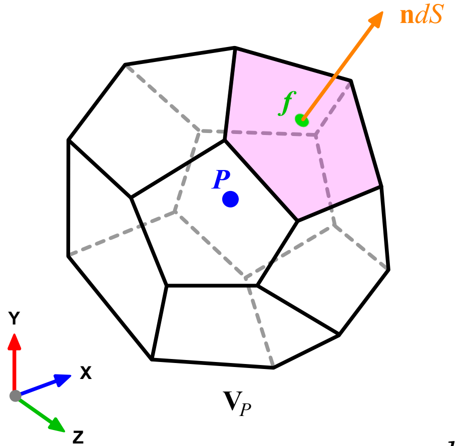
Gradient computation at cell centers and gradient reconstruction at face centers

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, node-based reconstruction, and so on.
- Using the Gauss method, the cell centered gradients can be computed as follows,

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



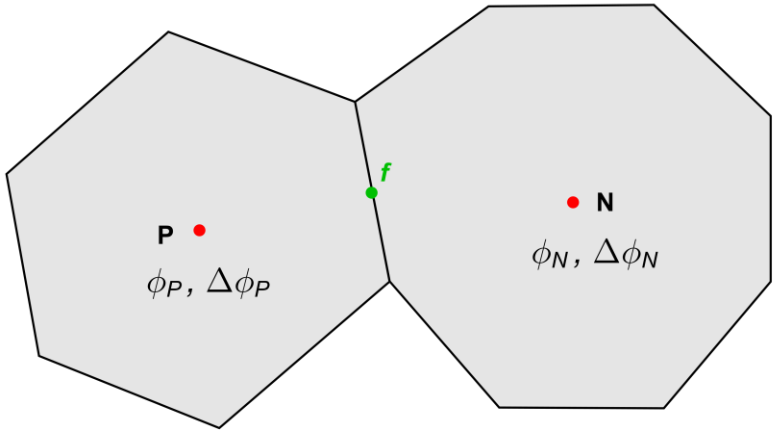
$$\mathbf{n}dS = d\mathbf{S}$$

- 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.

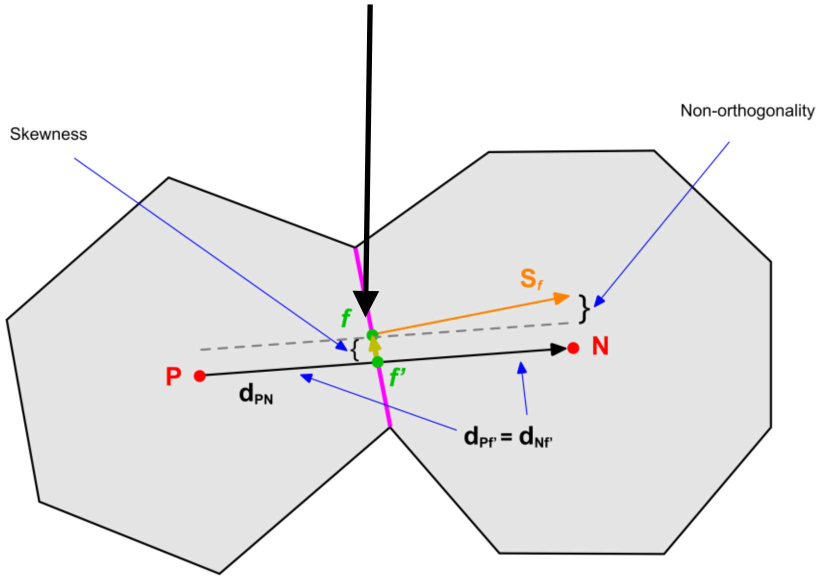


The Finite Volume Method: An overview

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).

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

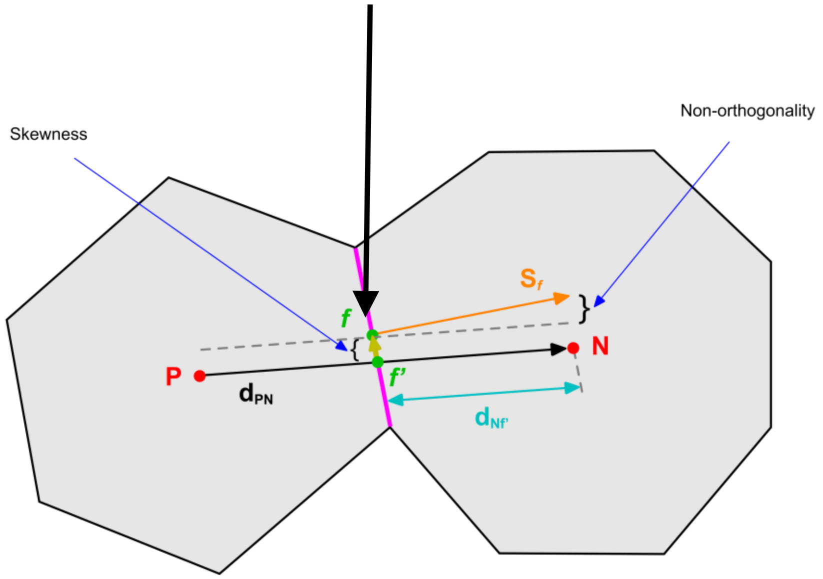


The Finite Volume Method: An overview

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.

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



The Finite Volume Method: An overview

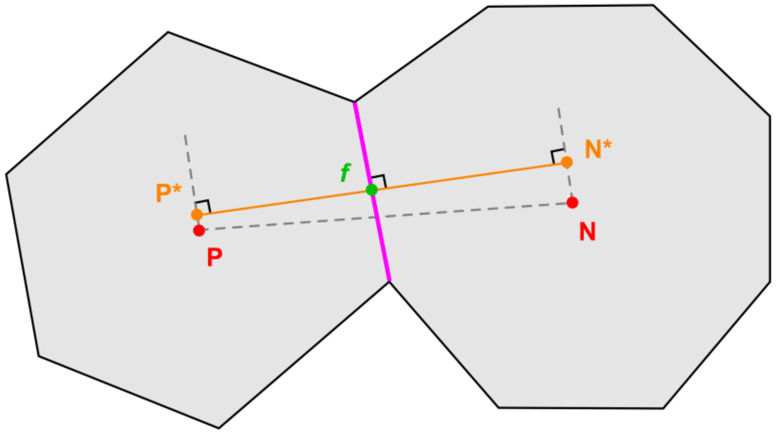
Gradients reconstruction at face centers

- 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),

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

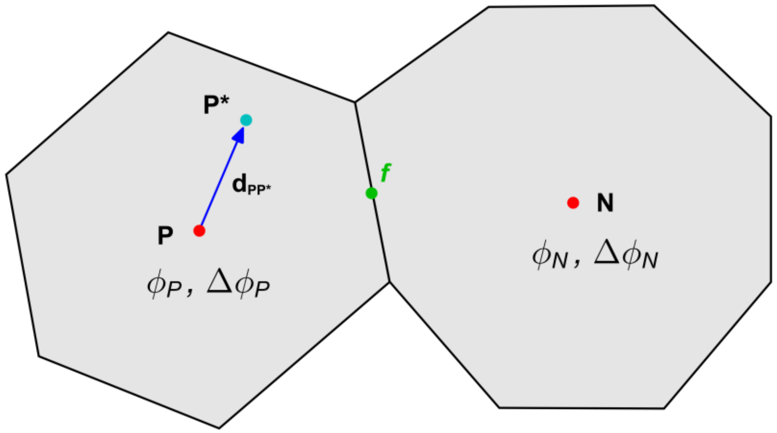


The Finite Volume Method: An overview

Gradients reconstruction at face centers

- In the previous formulation, recall that any cell centered quantity ϕ_P can be reconstructed in a new location **P*** (within the cell volume), as follows,

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

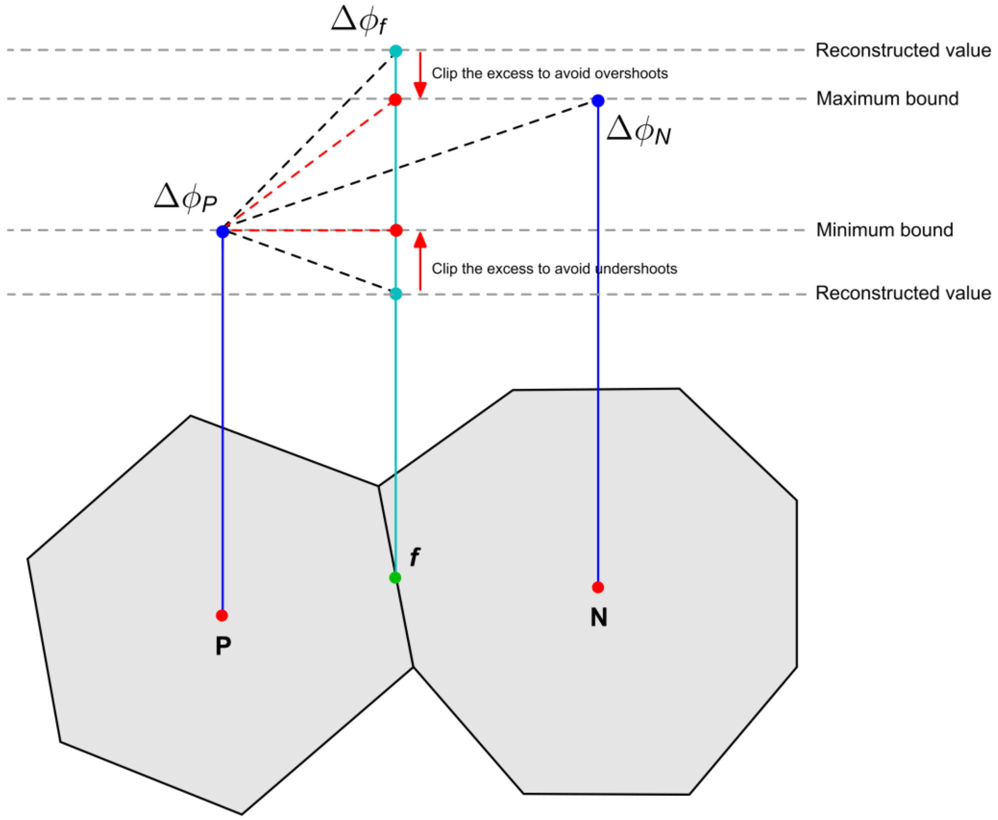


- All the previous approximations are second order accurate in good quality meshes.
- Also, the use of non-orthogonal corrections suggests the adoption of an iterative method to compute better face gradient approximations.

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). This increases 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.



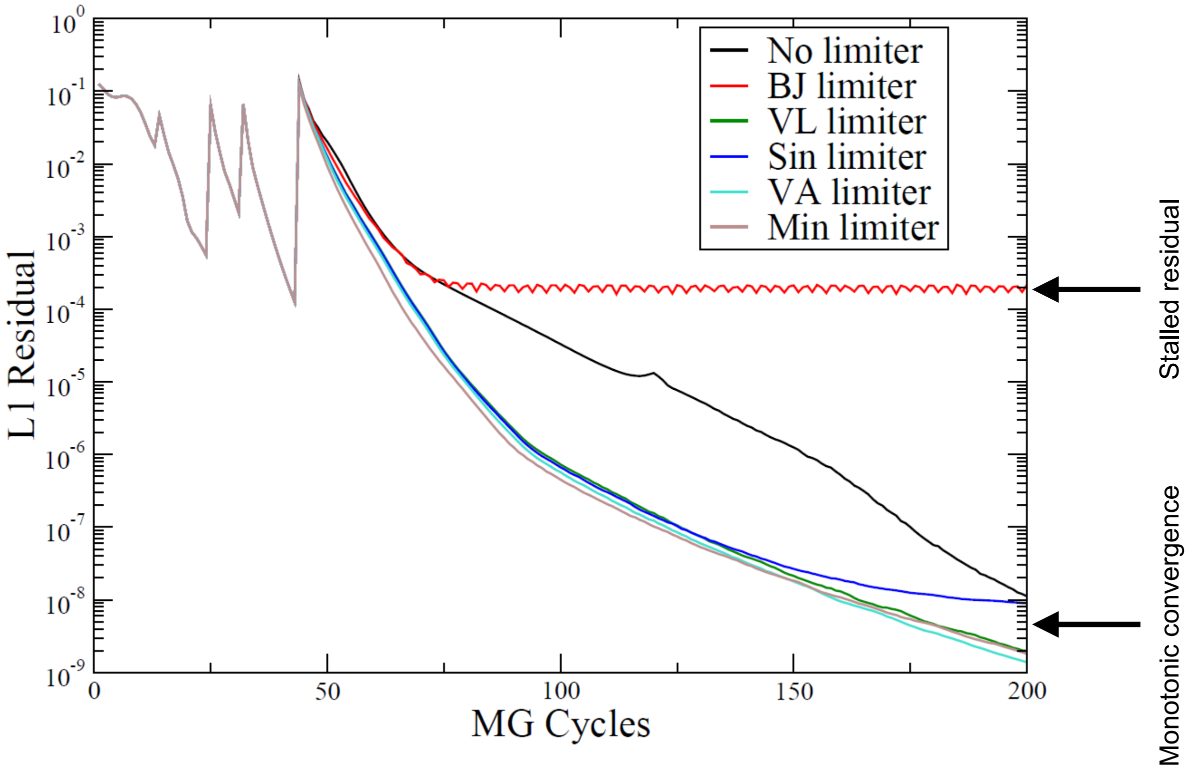
The Finite Volume Method: An overview

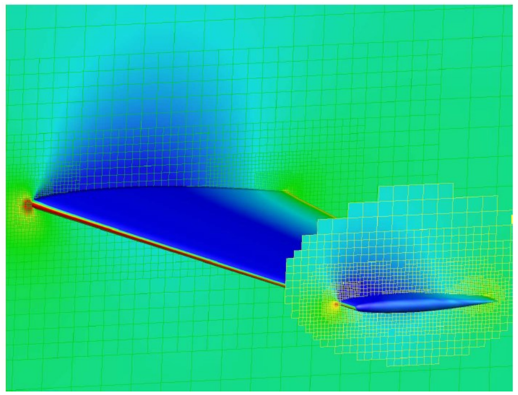
Effect of gradient limiters on solution accuracy and convergence to steady state

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

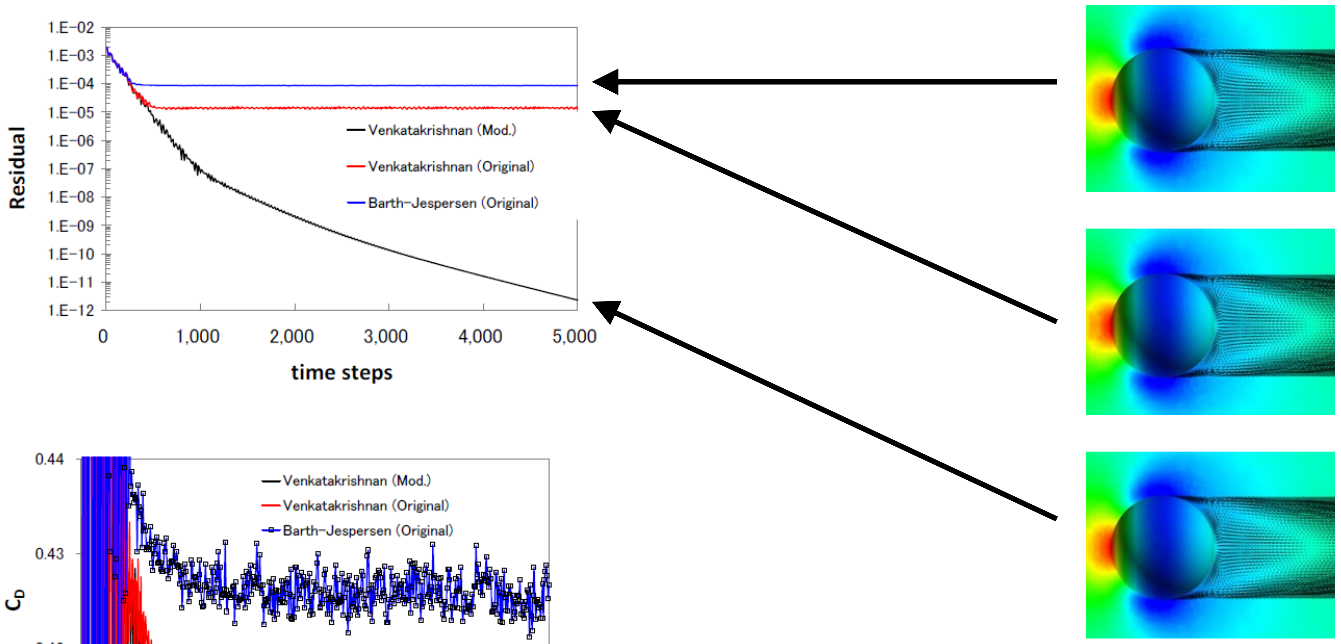
Computed drag for different limiter formulations, in order of increasing dissipation associated with the limiter.

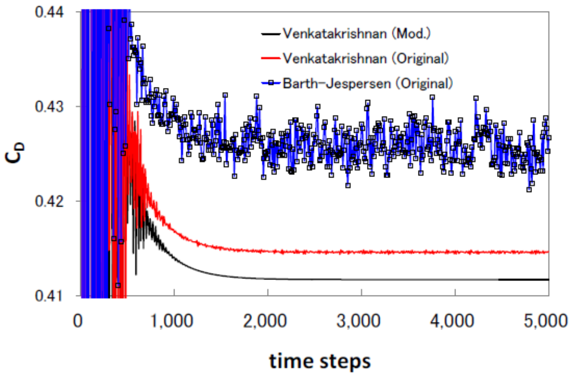
Reference:
M. Berger, M. Aftosmis, S. Murman, "Analysis of Slope Limiters on Irregular Grids"

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”

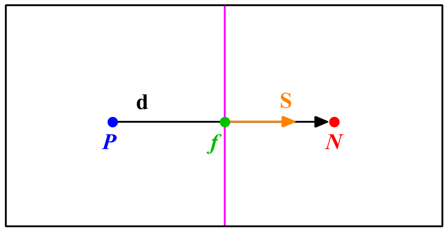
The Finite Volume Method: An overview

Mesh induced errors

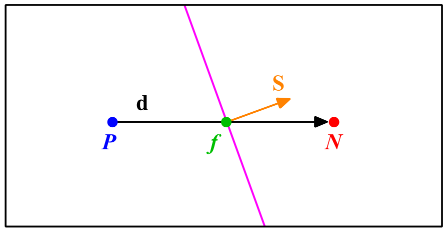
The Finite Volume Method: An overview

Mesh induced errors

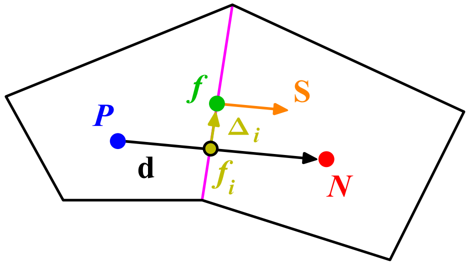
- In order to maintain second order accuracy and to avoid unboundedness, we need to correct non-orthogonality and skewness errors.
- Non-orthogonality and skewness errors can be avoided by having a good quality mesh.
- Or can be corrected (or minimized) numerically, but at the risk of adding numerical diffusion.
- The ideal case is to have an orthogonal and non skew mesh, but this is the exception rather than the rule.



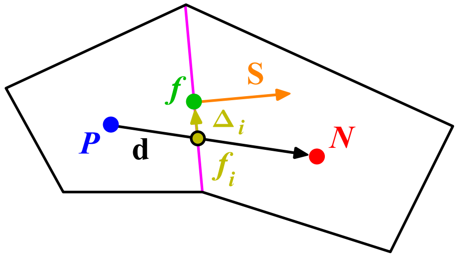
Orthogonal and non skew mesh



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,

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

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 temporal 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 temporal 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 temporal 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

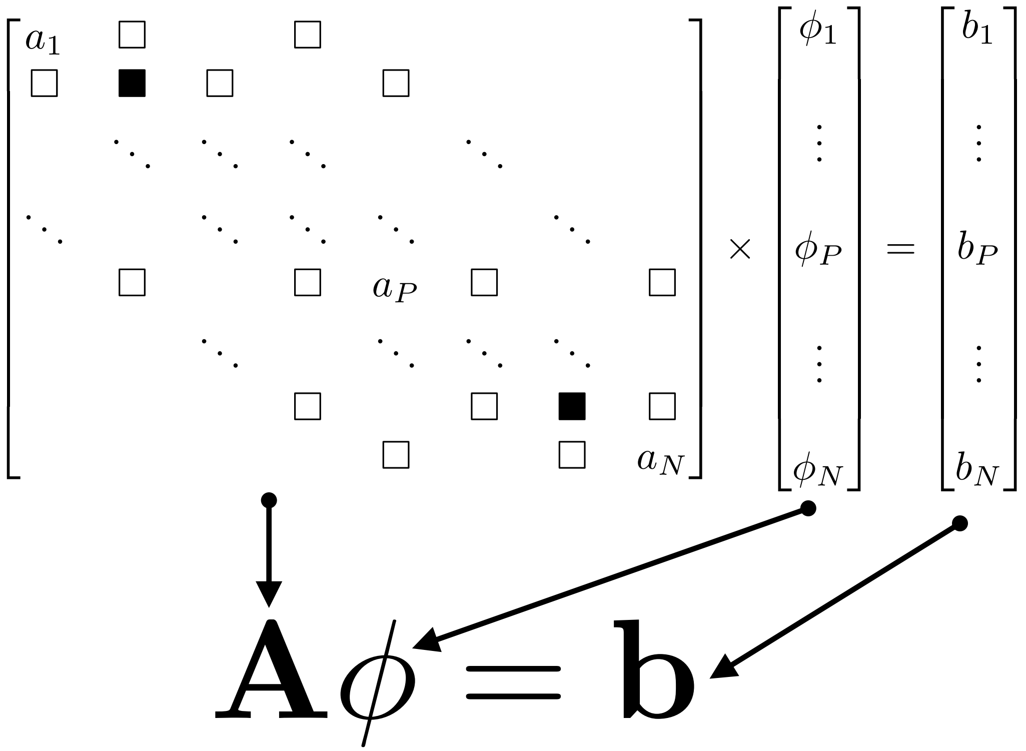
- After spatial and temporal discretization and by using the following equation,

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

in every control volume V_P of the domain, a system of differential algebraic equations (DAE) for the transported quantity ϕ is assembled,

■ $=$ Diagonal contribution

□ $=$ Off-diagonal contribution



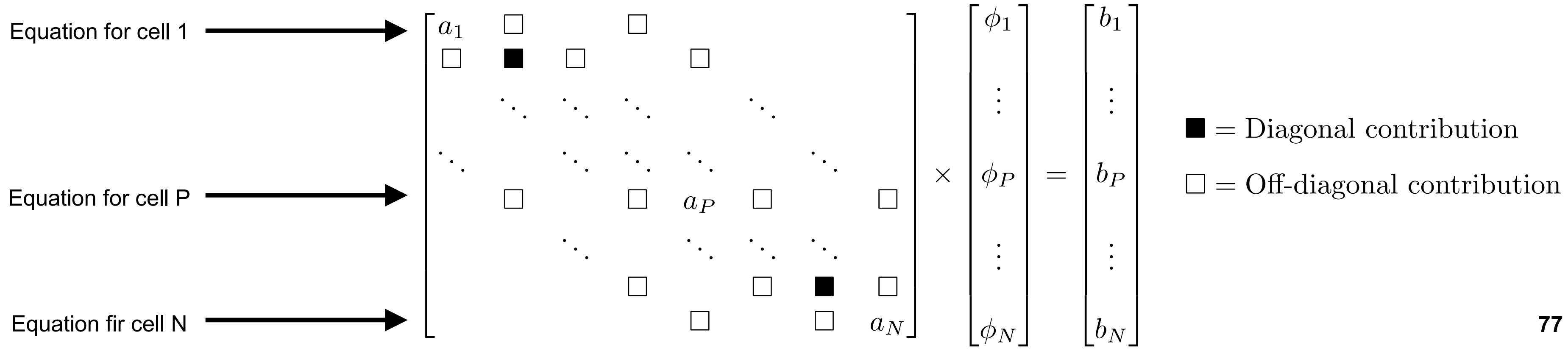
The Finite Volume Method: An overview

Linear system solution

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

$$\begin{tikzcd}
\boxed{\text{Matrix of coefficients}} & \mathbf{A}\phi = \mathbf{b} \arrow[l] \arrow[d] \arrow[r] & \boxed{\text{Boundary conditions and source terms vector}} \\
 & \boxed{\text{Solution vector}} &
\end{tikzcd}$$

- 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 **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).



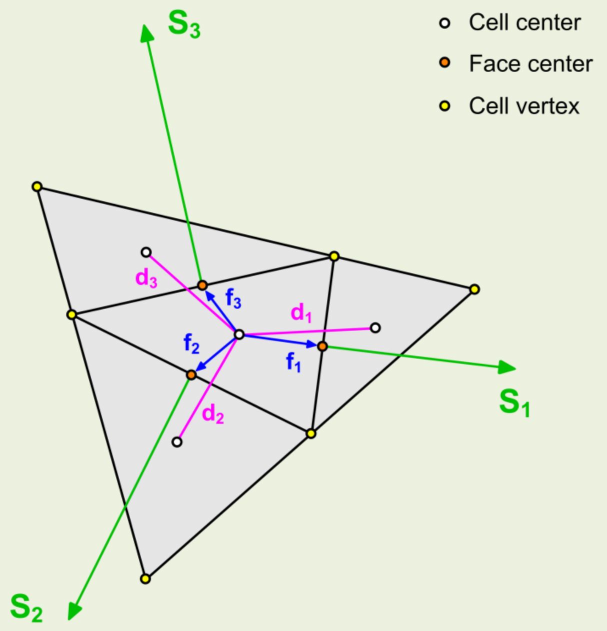
The Finite Volume Method: An overview

Linear system solution

- The matrix of coefficients **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 **S** (vector normal to face passing by the face center) and **d** (vector connecting two cell centers), that is, $\mathrm{S} \cdot \mathrm{d}$
- This dependence on the dot product $\mathrm{S} \cdot \mathrm{d}$ is because the coefficients contain the following term,

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

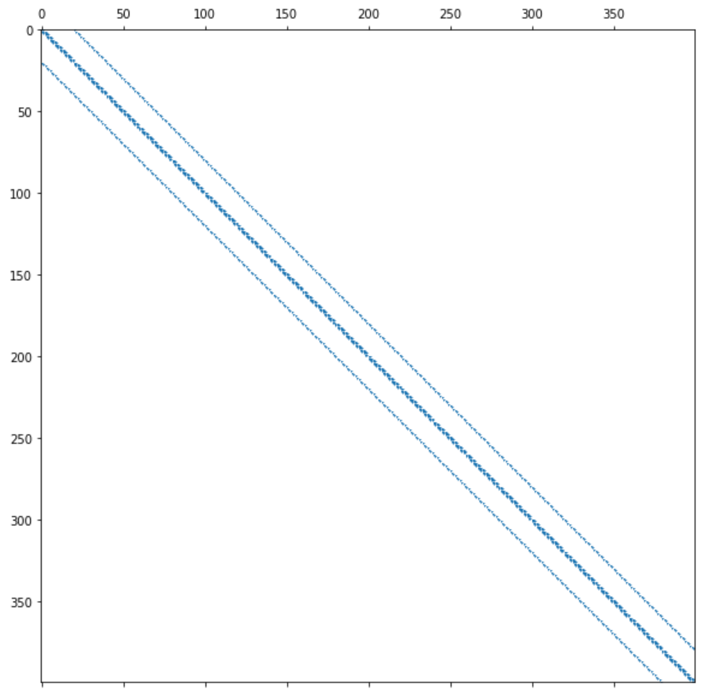
- For orthogonal meshes (perfect ones), the dot product is equal to one (there is no deviation between the vectors **S** and **d**).
- The more a cell deviates from its perfect shape, the smaller the dot product becomes, and this results in large values of the matrix coefficients which increases the 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.



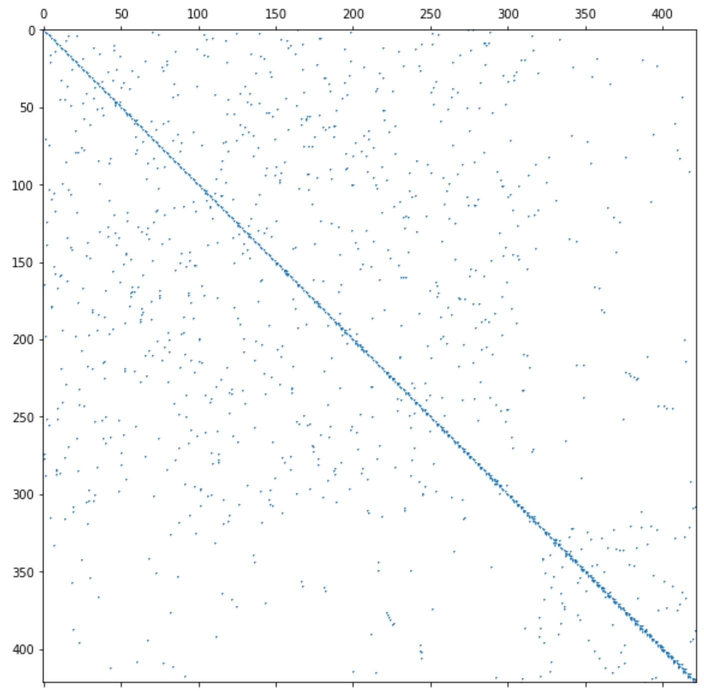
The Finite Volume Method: An overview

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 unknow quantity ϕ_P is distributed along the diagonal.
- The off-diagonal entries, represent the contribution of the neighboring cells ϕ_{nb}



Sparse matrix – Banded type
Typical of orthogonal meshes

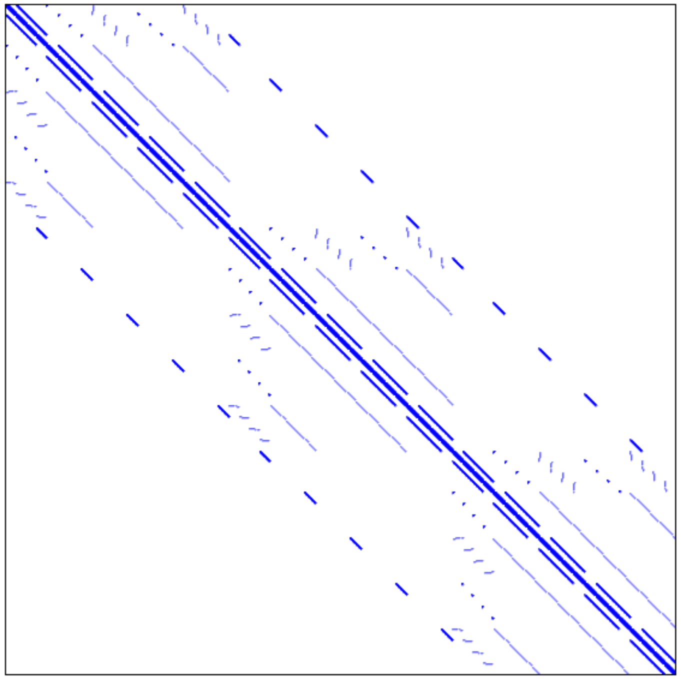


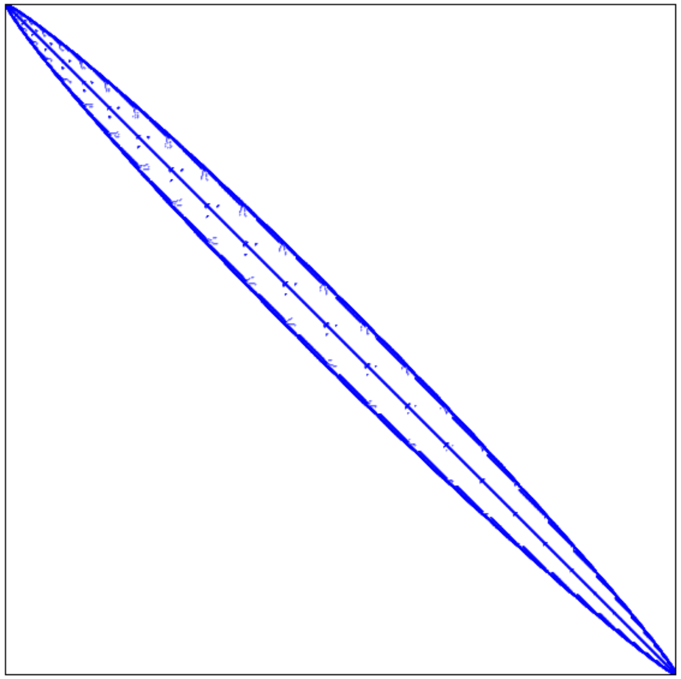
Sparse matrix – Non-banded structure
Typical of general unstructured meshes

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,

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

- And at least one *i*,

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

- 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].

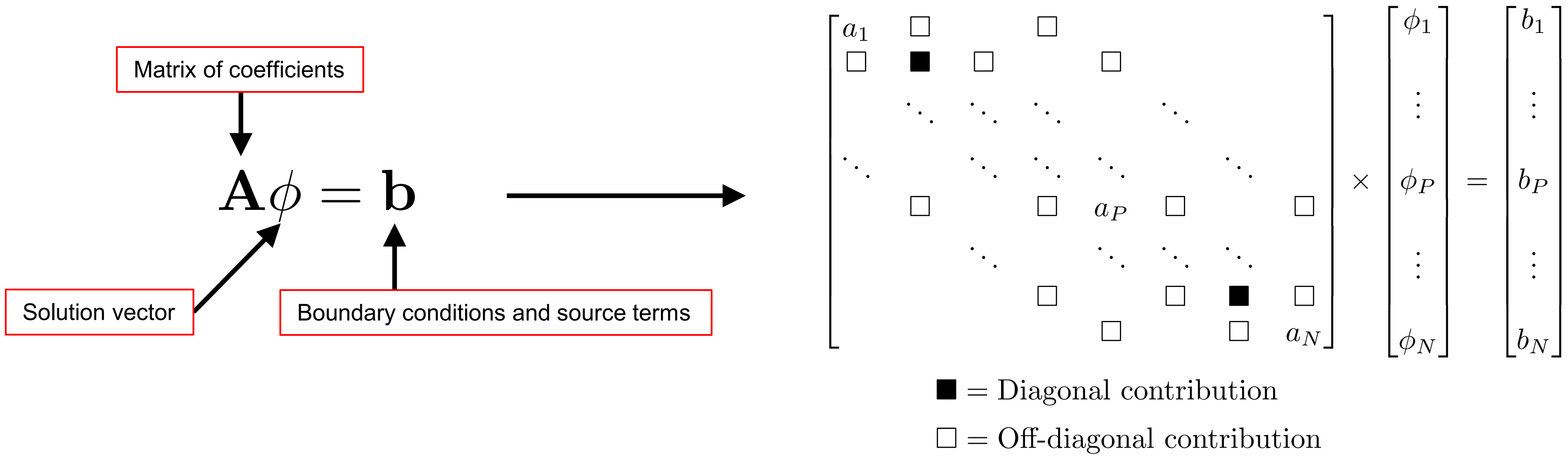
$$\frac{\sum |a_{nb}|}{|a_p|} \begin{cases} \leq 1, \text{for all equations} \\ < 1, \text{for at least one equation} \end{cases}$$

- 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).

[1] James Blaine Scarborough (1958). Numerical Mathematical Analysis. Johns Hopkins Press.

The Finite Volume Method: An overview

Linear system solution


Matrix of coefficients
$\mathbf{A}\phi = \mathbf{b}$
Solution vector
Boundary conditions and source terms
a_1
a_P
a_N
ϕ_1
ϕ_P
ϕ_N
b_1
b_P
b_N
■ = Diagonal contribution
□ = Off-diagonal contribution

- 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.

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

- Basically, iterative solvers incrementally reduce the error, until reaching a given residual **r** (absolute or relative tolerance),

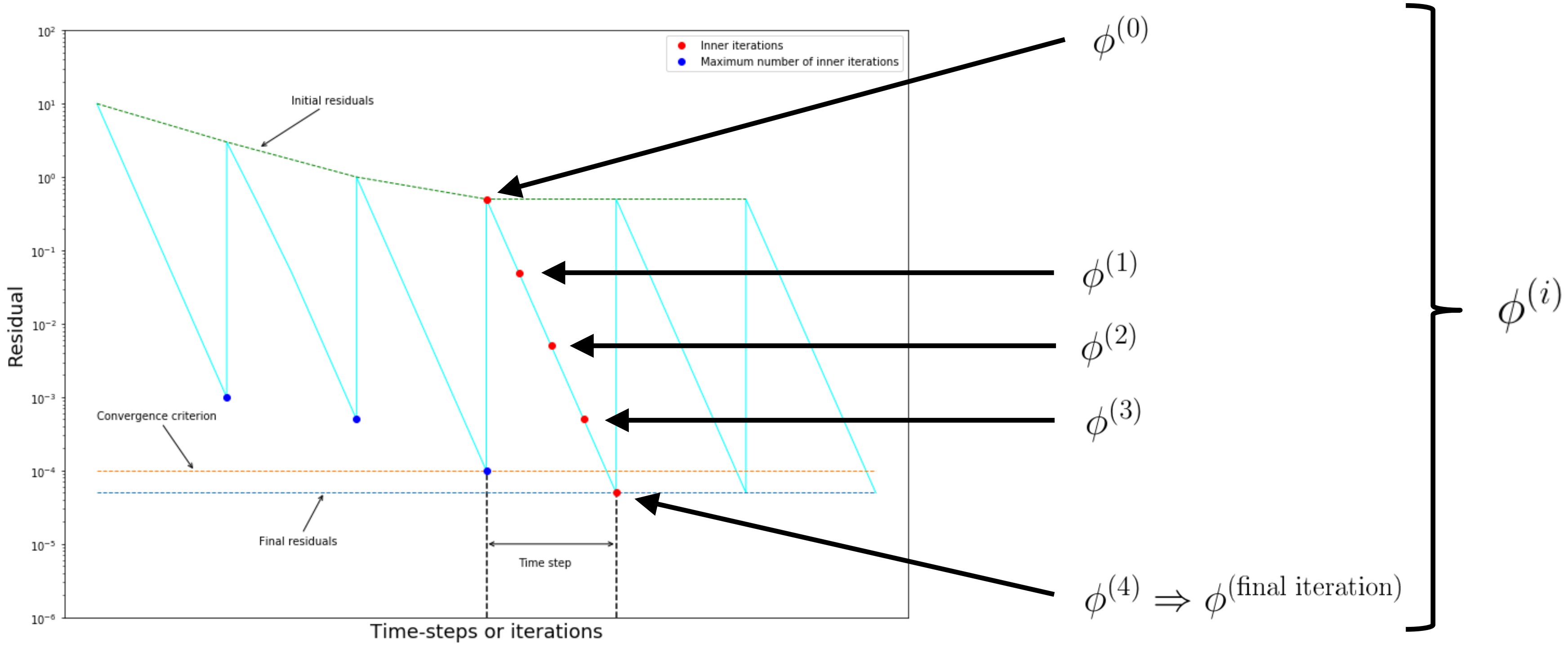
$$\left|\mathbf{A}\phi^i - \mathbf{b}\right| \leq \left|\mathbf{r}^i\right|$$

- The convergence rate of iterative solvers greatly depends on the matrix of coefficients **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.

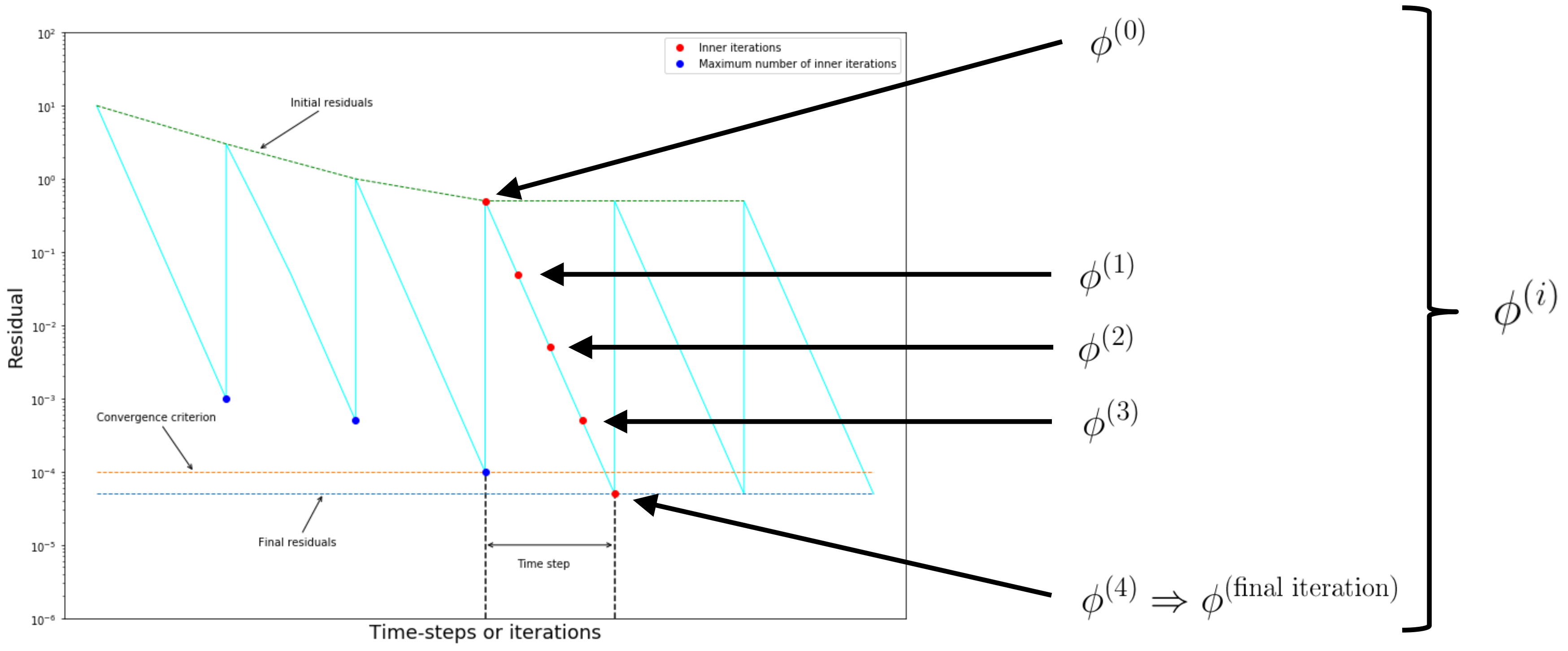


- $\phi^{(0)}$ is the initial guess used to start the iterative method.
- Iteration 0 defines the initial residual, and greatly influence the convergence rate.
- You can use any value at iteration 0, but usually is a good choice to take the previous solution vector.
- Remember, the closest you are to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

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.

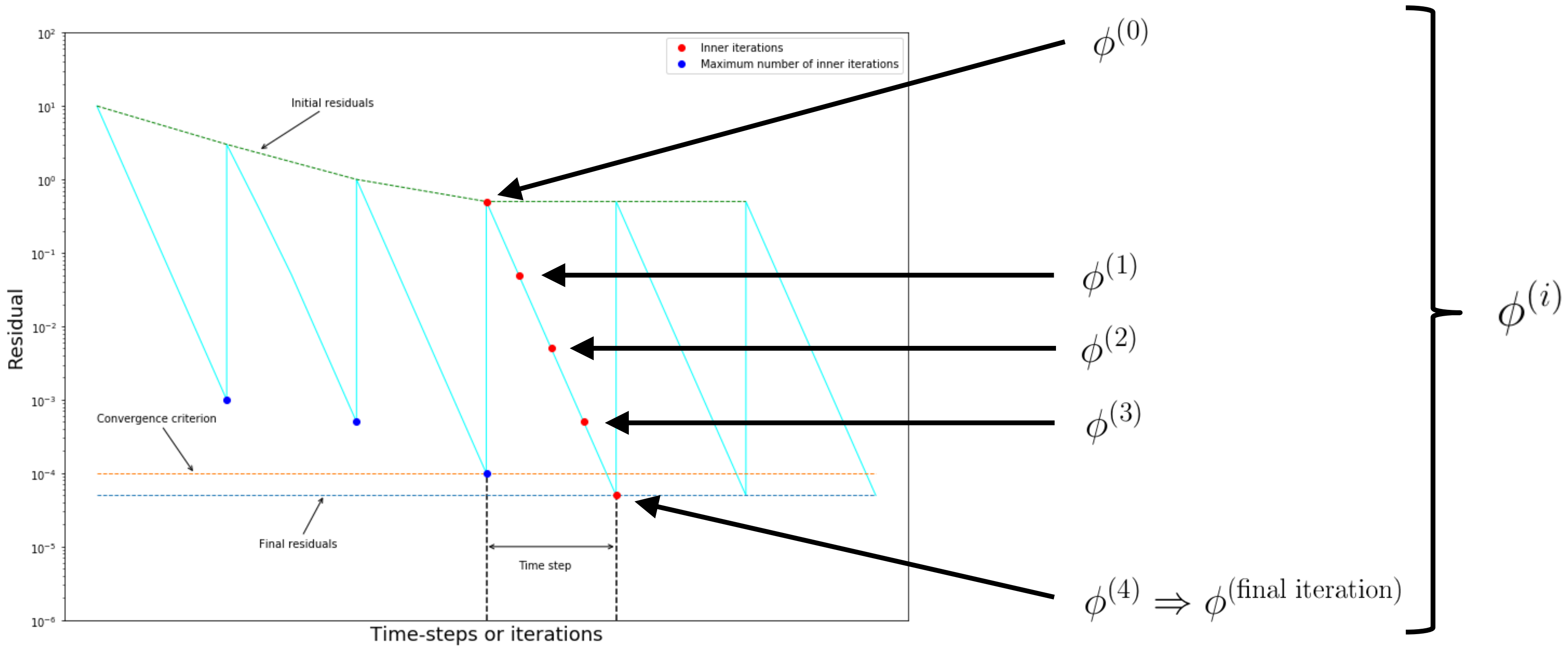
$$\left|\mathbf{A}\phi^{i} - \mathbf{b}\right| \leq |\mathbf{r}|$$

- This condition defines the final residual, where **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.

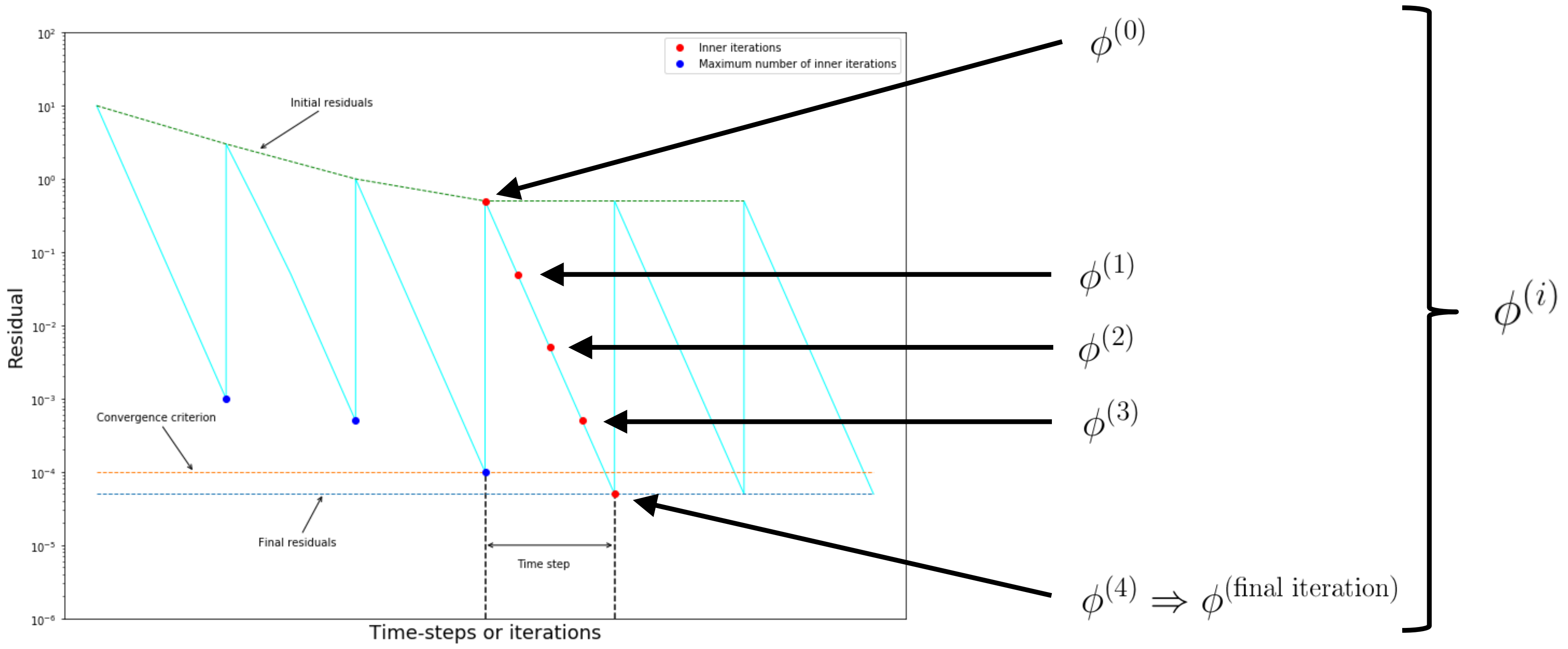


- By working in an iterative way, every single iteration $\phi^{(i)}$ is a better approximation of the previous iteration $\phi^{(i-1)}$
- Sometimes the linear solver might stop iterating because it has reached the maximum number of iterations, you should be careful of this because we are talking of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

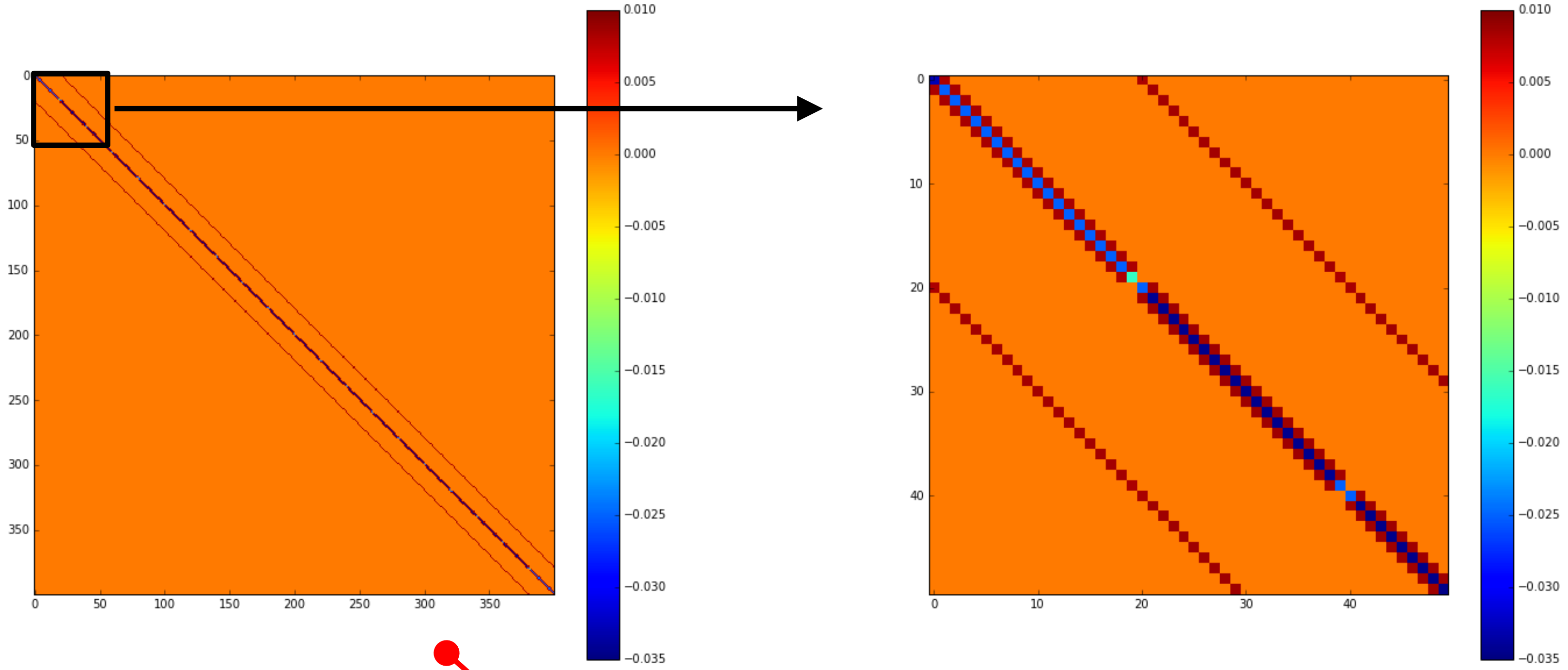


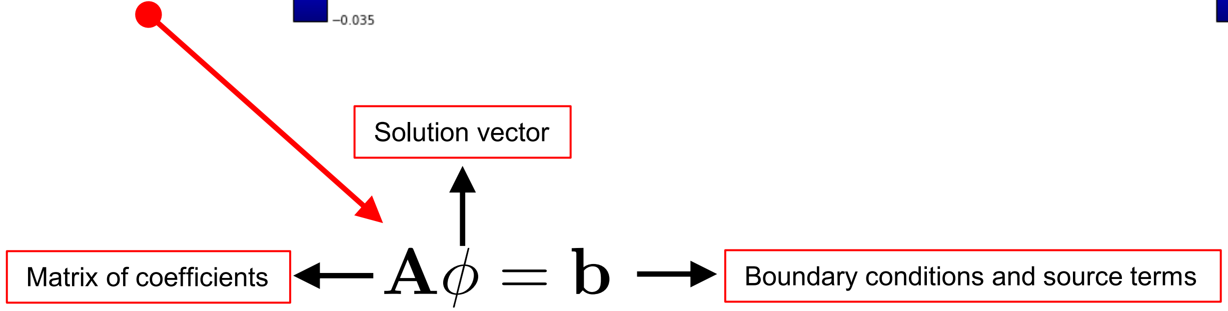
- It is clear that if the initial residual $|\mathbf{A}\phi - \mathbf{b}|^0$ is the same as the final residual $|\mathbf{A}\phi - \mathbf{b}|^{\text{Final iteration}}$ (we are converging in one iteration), we can say that we have reached a steady solution (this does not happen very often).
- Every iterative 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.





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

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 FVM solver do?

The Finite Volume Method: An overview

So, what does an 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 differential algebraic equations (DAE), and then it solves this system of DAE to find the solution of the transported quantities.
- Therefore, 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® and SU2).

Roadmap

1. ~~Important concepts to remember~~
2. ~~The Finite Volume Method: An overview~~
3. **The FVM in OpenFOAM®: some implementation details and computational pointers**
4. Some kind of conclusion
5. What else we did not cover?
6. Goodbye

The FVM in OpenFOAM®

So, what does OpenFOAM® do?

- It simply discretize 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 discrete algebraic equations (DAE), and then it solves this system of DAE 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`**
  - Physical models, 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.
  - How to discretize in time the obtained semi-discrete governing equations. This information is set in the *`system/fvSchemes`* dictionary.
  - How to solve the linear system of discrete 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 **functionObjects** for post-processing. This information is contained in the specific dictionaries contained 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 commercial solvers 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; therefore, it is important to understand the theory.

The FVM in OpenFOAM®

Are there default options in OpenFOAM®?

- Hereafter we are going to give you what we think are the best options.
- The recipes that we are going to give you, are based on 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.

The FVM in OpenFOAM®

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

```
ddtSchemes
{
        default         backward;
}
```

$$\frac{\partial \phi}{\partial t}$$

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

$$\nabla \phi_P$$

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

$$\nabla \cdot (\mathbf{U}\phi)$$

```
laplacianSchemes
{
        default         Gauss linear orthogonal;
}
```

$$\nabla \cdot \Gamma \nabla \phi$$

```
interpolationSchemes
{
        default         linear;
}
```

$$\left\{\begin{array}{l} \phi_f = f_x \phi_P + (1 - f_x)\,\phi_N \\ f_x = \dfrac{fN}{PN} = \dfrac{|\,\mathbf{x}_f - \mathbf{x}_N\,|}{|\,\mathbf{d}\,|} \end{array}\right.$$

```
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 term 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  ψ;
}
```

- Setting ψ 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.
  - **limitedLinear**: second order accurate, unbounded, but more stable than pure linear ((better boundedness). Recommended for LES simulations (kind of similar to the Fromm method).
  - A good TVD scheme (**vanLeer** or **Minmod**): at least second order accurate, bounded.
  - **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)**,

```
gradSchemes
{
        grad(U)         cellMDLimited Gauss linear 1.0;
}

divSchemes
{
        div(phi,U)      Gauss linearUpwind          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.*,

$$\underbrace{\text{grad(U)}}_{\substack{\text{Compute the gradient of this field variable,}\\ \textit{e.g.}\text{, U, p, T, alpha, k, omega, and so on}}} \quad \overbrace{\text{Gauss}}^{\text{Gradient computation method}} \quad \underbrace{\text{linear;}}_{\text{Cell-center to face-center interpolation method}}$$

- 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 for the **cellLimited** option.
- Recall that the **cubic** or **Venkatakrishnan** are differentiable limiters, whereas the **minmod** is non-differentiable.
- You will find the source code in the following directory:
  - `$WM_PROJECT_DIR/src/finiteVolume/finiteVolume/gradSchemes/limitedGradSchemes/cellLimitedGrad/gradientLimiters`
- 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>**

The FVM in OpenFOAM®

Gradient terms discretization schemes

- These are the gradient discretization schemes that you will use most of the times:
    - **Gauss** (use with interpolation method, *i.e.*, Gauss linear)
    - **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:

cellMDLimited

cellLimited

faceMDLimited

faceLimited

Less diffusive

↑

More diffusive

Note: for smooth field variation, cell limiting may provide less numerical dissipation on meshes with skewed cells.

- 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®

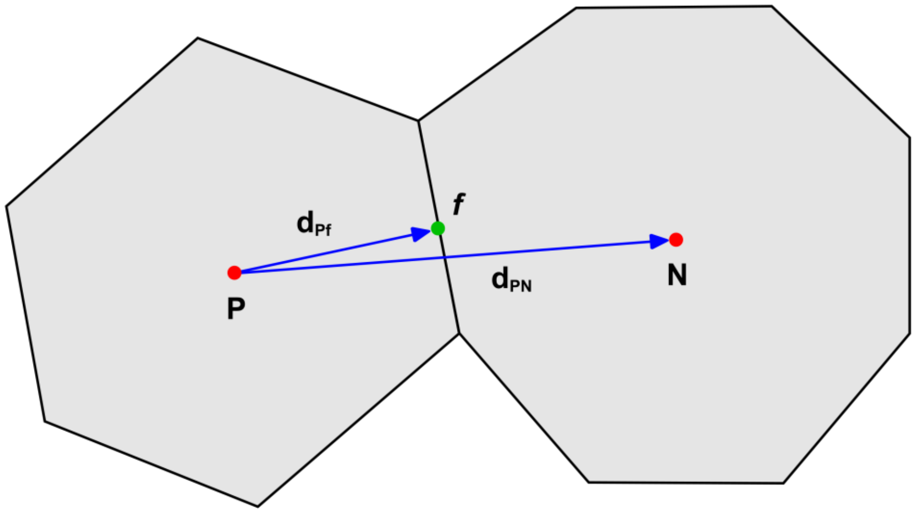
Gradient terms discretization schemes

- Limiting direction:
  - Cell-to-cell direction limiting,

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

  - Cell-to-face direction limiting,

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

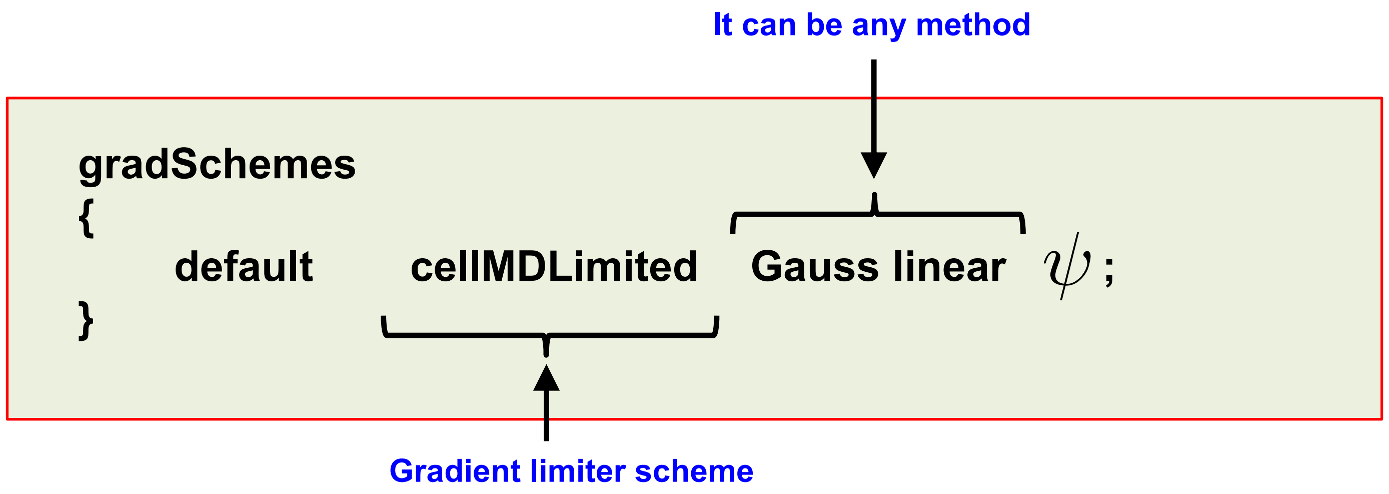


- Cell based limiters will limit cell-to-cell values. That is, in the direction $\mathbf{d}_{\mathbf{PN}}$.
- Face based limiters will limit cell-to-face values. That is, in the direction $\mathbf{d}_{\mathbf{Pf}}$.
- The more skewed the mesh is, the bigger the different between these methods.
- In good quality meshes both limiters will give 2nd order accuracy. However, in highly skewed meshes the face limiters might give 1st 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 ψ.



```
gradSchemes
{
        default       cellMDLimited   Gauss linear   ψ ;
}
```

- Setting ψ 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.

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

  - **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.

Can be computed using the over-relaxed approach

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

  - **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.

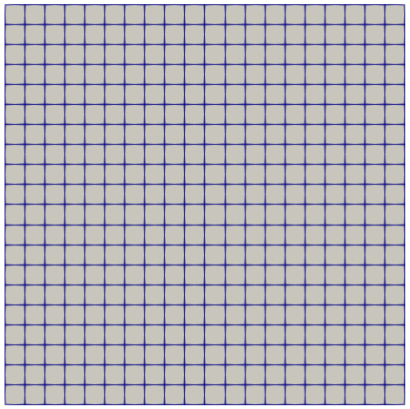
$$\longrightarrow \quad \mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \cancel{\underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}}.$$

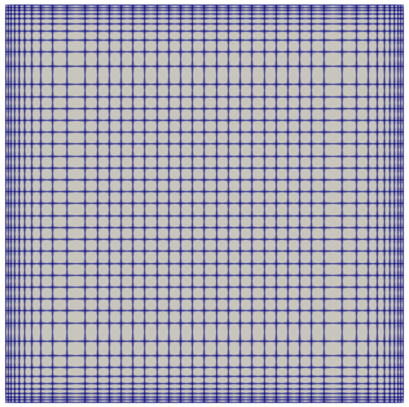
Can be computed using the over-relaxed approach

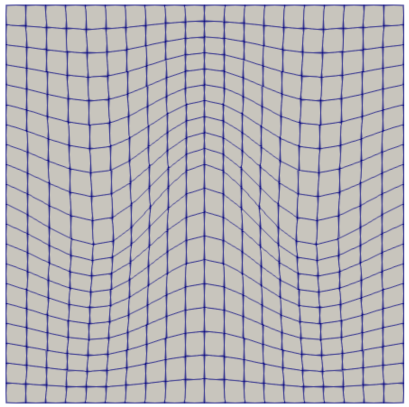
The FVM in OpenFOAM®

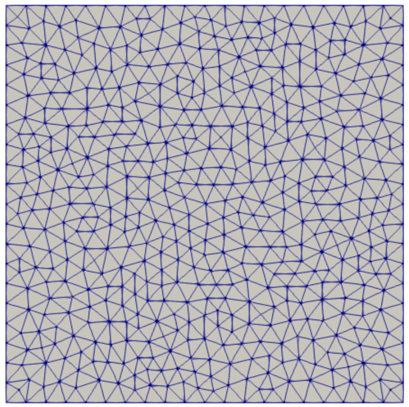
Laplacian terms discretization schemes

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


Perfect orthogonal mesh with no strectching
laplacianSchemes → orthogonal


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

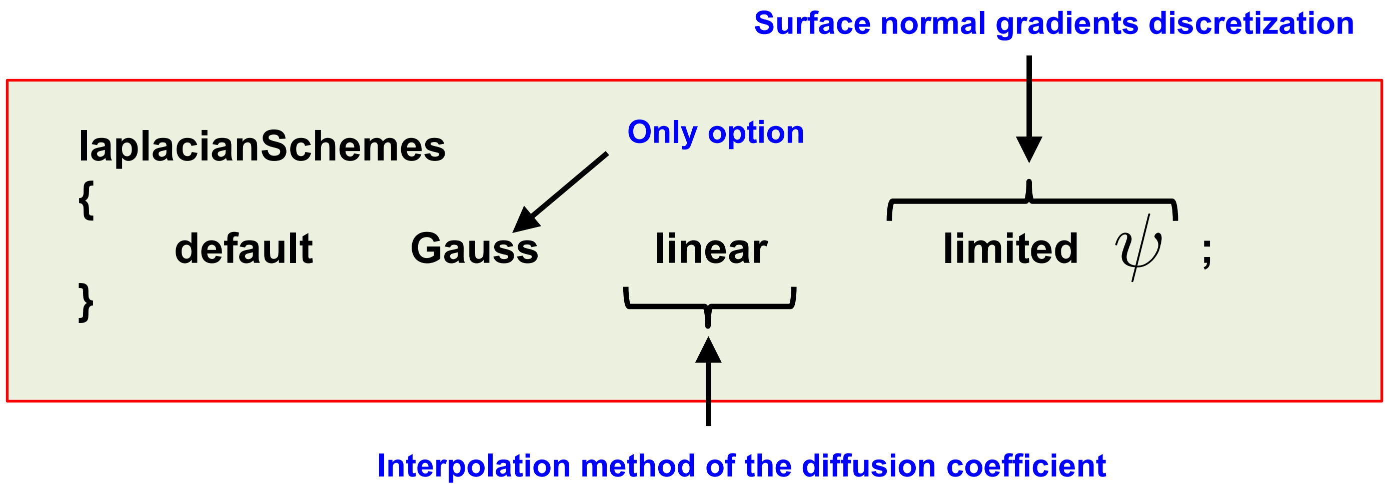

Mesh with some degree of non-orthogonality (low to medium)
laplacianSchemes → limited 1 to **limited 0.5**


General unstructured meshes
laplacianSchemes → limited 0.5

The FVM in OpenFOAM®

Laplacian terms discretization schemes

- The limited method uses a blending factor ψ.



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

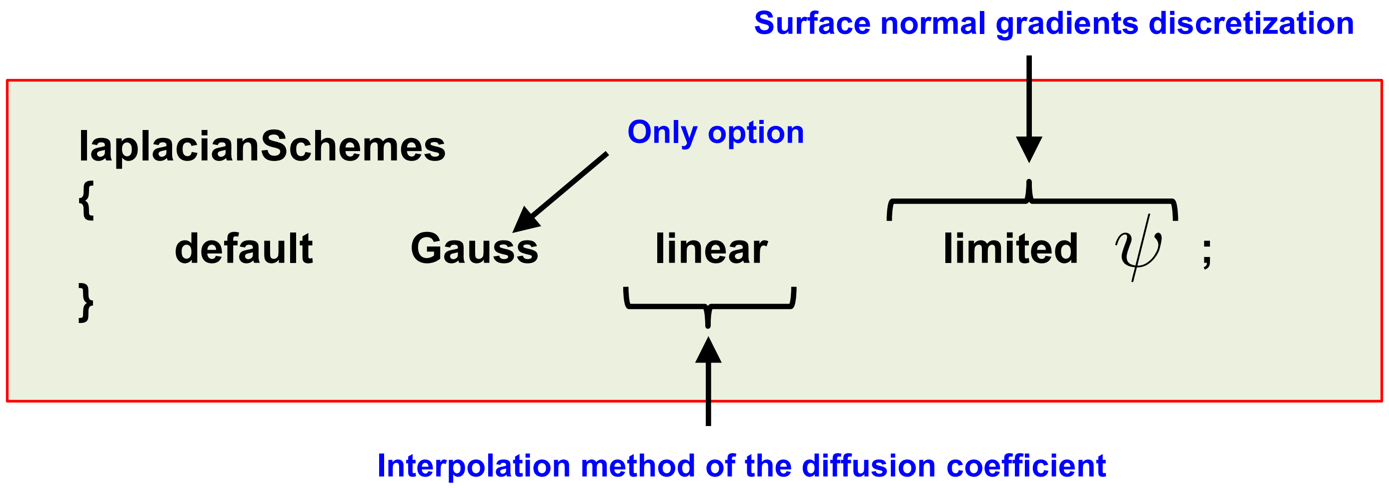
Surface normal gradients discretization
Only option
Interpolation method of the diffusion coefficient

- Setting ψ to 1 is equivalent to using the **corrected** method. You gain accuracy, but the solution might become unbounded.
- By setting the blending factor equal to 0 is equivalent to using the **uncorrected** method. You give up accuracy but gain stability.
- If you set the blending factor to 0.5, you get the best of both worlds. In this case, the non-orthogonal contribution does not exceed the orthogonal part. You give up accuracy but gain stability.
- By setting the blending factor to 0.333 you get more stability at the cost of losing accuracy.

The FVM in OpenFOAM®

Laplacian terms discretization schemes

- The limited method uses a blending factor ψ.



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

Surface normal gradients discretization
Only option
Interpolation method of the diffusion coefficient

- 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. Also, you will need to increase the number of non-orthogonal corrections.
- For meshes with non-orthogonality more than 85, it is better to get a better mesh. But if you definitely want to use that mesh, you can set the blending factor to 0.333, and 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).
- These are conservative indications based on our experience, and the values may change according to the application.

The FVM in OpenFOAM®

Laplacian terms discretization schemes

- Just to make it clear, the blending factor ψ is used to avoid the non-orthogonal contribution exceeding the orthogonal part, that is, non-orthogonal contribution ≤ 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 part

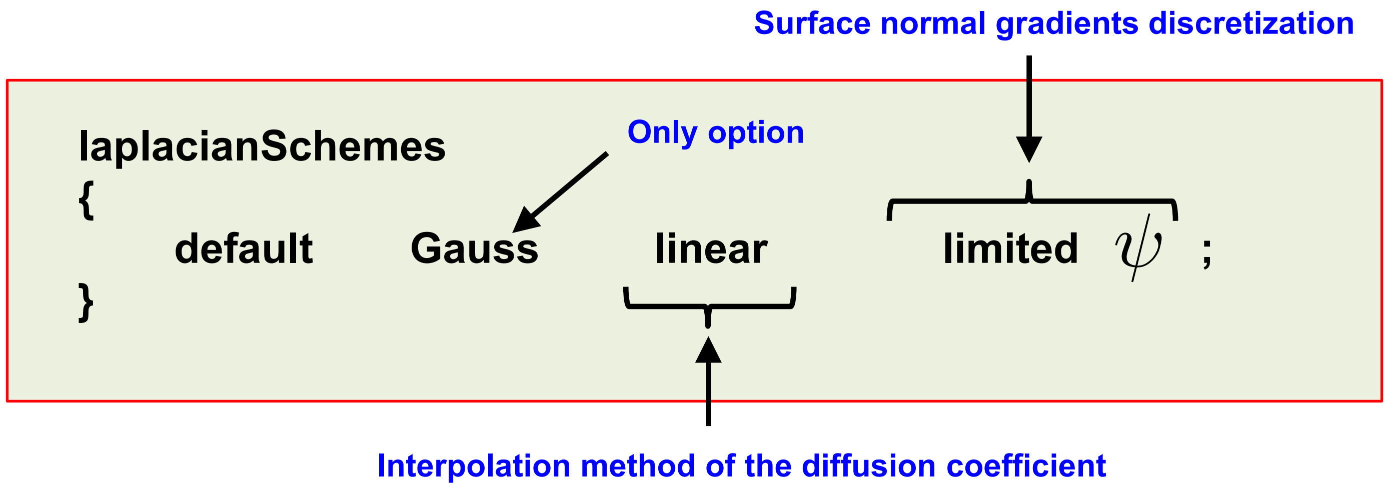
Explicit part

- In meshes with large non-orthogonality, the explicit term can lead to unboundedness and eventually divergence.
- 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.
- It becomes harder to guarantee diagonal dominance of the matrix of coefficient; therefore, the Scarborough criterion might not be satisfied.

The FVM in OpenFOAM®

Laplacian terms discretization schemes

- The limited method uses a blending factor ψ.



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

Surface normal gradients discretization
Only option
Interpolation method of the diffusion coefficient

- 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;
}
```

- **<u>This setup is recommended for most of the cases.</u>**
- 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.
- 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®

A very accurate but oscillatory numerics

```
ddtSchemes
{
        //default          backward;
        default            CrankNicolson 0.7;
}
gradSchemes
{
        default            Gauss leastSquares;
}
divSchemes
{
        default                      none;
        div(phi,U)                   Gauss linear;    //limitedLinear
        div(phi,omega)               Gauss linear;    //limitedLinear
        div(phi,k)                   Gauss linear;    //limitedLinear

        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 and meshes with overall good quality.
- 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®

A very stable but too diffusive numerics

```
ddtSchemes
{
      default            Euler;
}
gradSchemes
{
      default            cellLimited Gauss linear 1;
      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.
- **Start robustly, end with accuracy.**
- You can use this method for troubleshooting. If the solution diverges, you better check boundary conditions, physical properties, and so on.

Roadmap

1. ~~Important concepts to remember~~
2. ~~The Finite Volume Method: An overview~~
3. ~~The FVM in OpenFOAM®: some implementation details and computational pointers~~
4. **Some kind of conclusion**
5. What else we did not cover?
6. Goodbye

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.

Roadmap

1. ~~Important concepts to remember~~
2. ~~The Finite Volume Method: An overview~~
3. ~~The FVM in OpenFOAM®: some implementation details and computational pointers~~
4. ~~Some kind of conclusion~~
5. **What else we did not cover?**
6. Goodbye

What else we did not cover?

A lot!

What else we did not cover?

- Just to give you an idea,
    - Linear solvers and acceleration techniques.
    - Pressure-velocity coupling.
    - CFL number.
    - Under-relaxation factors.
    - Unsteady and steady simulations.
    - Understanding the residuals and solution monitoring.
    - Boundary conditions and initials conditions.
    - Advanced physics. Turbulence modeling, multiphase flows, compressible flows, moving bodies, source terms, passive scalars, combustions, and so on.
    - Solution stabilization techniques.
    - Parallel issues.
- Of all these topics, let me address briefly pressure-velocity coupling.

What else we did not cover?

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
- 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.

What else we did not cover?

Pressure-velocity coupling

- In OpenFOAM®, you will find segregated pressure-based solvers.
    - In the segregated algorithm, the individual governing equations for the primitive variables are solved one after another.
    - In the pressure-based approach the velocity field is obtained from the momentum equations.
    - Then, the pressure is obtained by solving the pressure-Poisson equation.
    - There is some mathematical manipulation involved.

- Coupled solvers are under development in OpenFOAM®.
    - The coupled approach solves the continuity, momentum, and energy equation simultaneously, that is, coupled together.
    - They are very efficient when used with steady solvers.
    - However, the memory requirements are much higher than segregated solvers.

- Pressure-based methods (segregated or coupled) are intrinsically implicit.

What else we did not cover?

Pressure-velocity coupling

- In OpenFOAM®, the following segregated pressure-based 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 ITA-PISO.

- The ITA-PISO formulation can give you more accuracy and stability when using very large time-steps, pseudo-transient simulations, or when dealing with complex physics.
- In OpenFOAM®, the **PISO** and **PIMPLE** methods are formulated for unsteady simulations.
- Whereas, the **SIMPLE** and **SIMPLEC** methods are formulated for steady simulations.

What else we did not cover?

Pressure-velocity coupling 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).

- **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).

What else we did not cover?

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,

$$\cancelto{0}{\nabla \cdot \left(\frac{\partial \mathbf{U}}{\partial t} \right)} + \nabla \cdot (\nabla \cdot (\mathbf{U}\mathbf{U})) - \cancelto{0}{\nabla \cdot (\nu \nabla^2 \mathbf{U})} = -\nabla^2 p$$

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

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

What else we did not cover?

Equations used in the SIMPLE and PISO loops

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

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

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

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

- 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 Naiver-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
- 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).

What else we did not cover?

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***U**,

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

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

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

What else we did not cover?

Equations used in OpenFOAM® SIMPLE and PISO loops

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

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

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

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

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

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

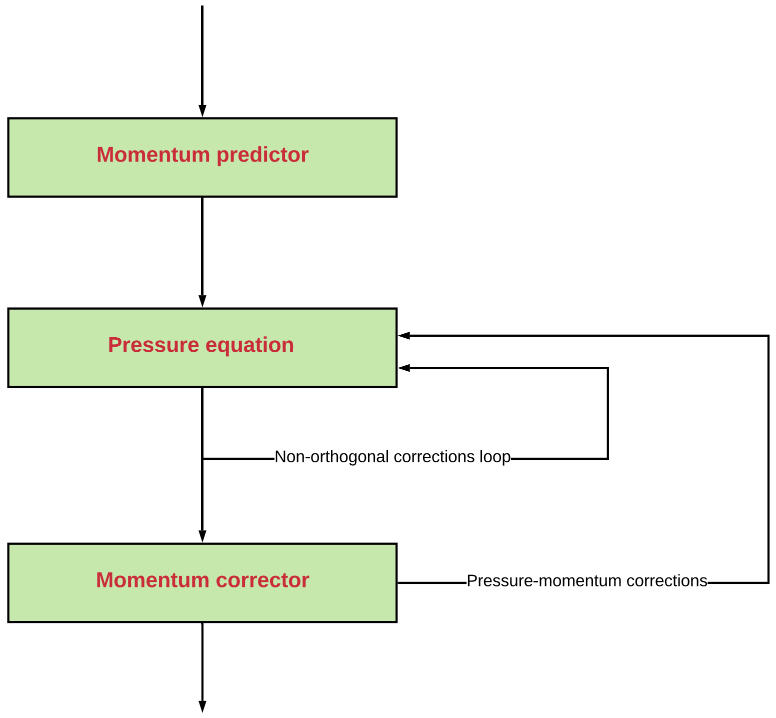
What else we did not cover?

Equations used in OpenFOAM® SIMPLE and PISO loops

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

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

- 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.
- Notice that mesh non-orthogonality and skewness introduces secondary gradients in every equation where a diffusion term arises.
- For example, energy equation ($f\left(\nabla T\right)$).
- This equation tends to be more sensitive to secondary gradients than the pressure equation.


Momentum predictor
Pressure equation
Non-orthogonal corrections loop
Momentum corrector
Pressure-momentum corrections

What else we did not cover?

The SIMPLE loop in OpenFOAM®

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

Momentum equation without the pressure gradient term

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

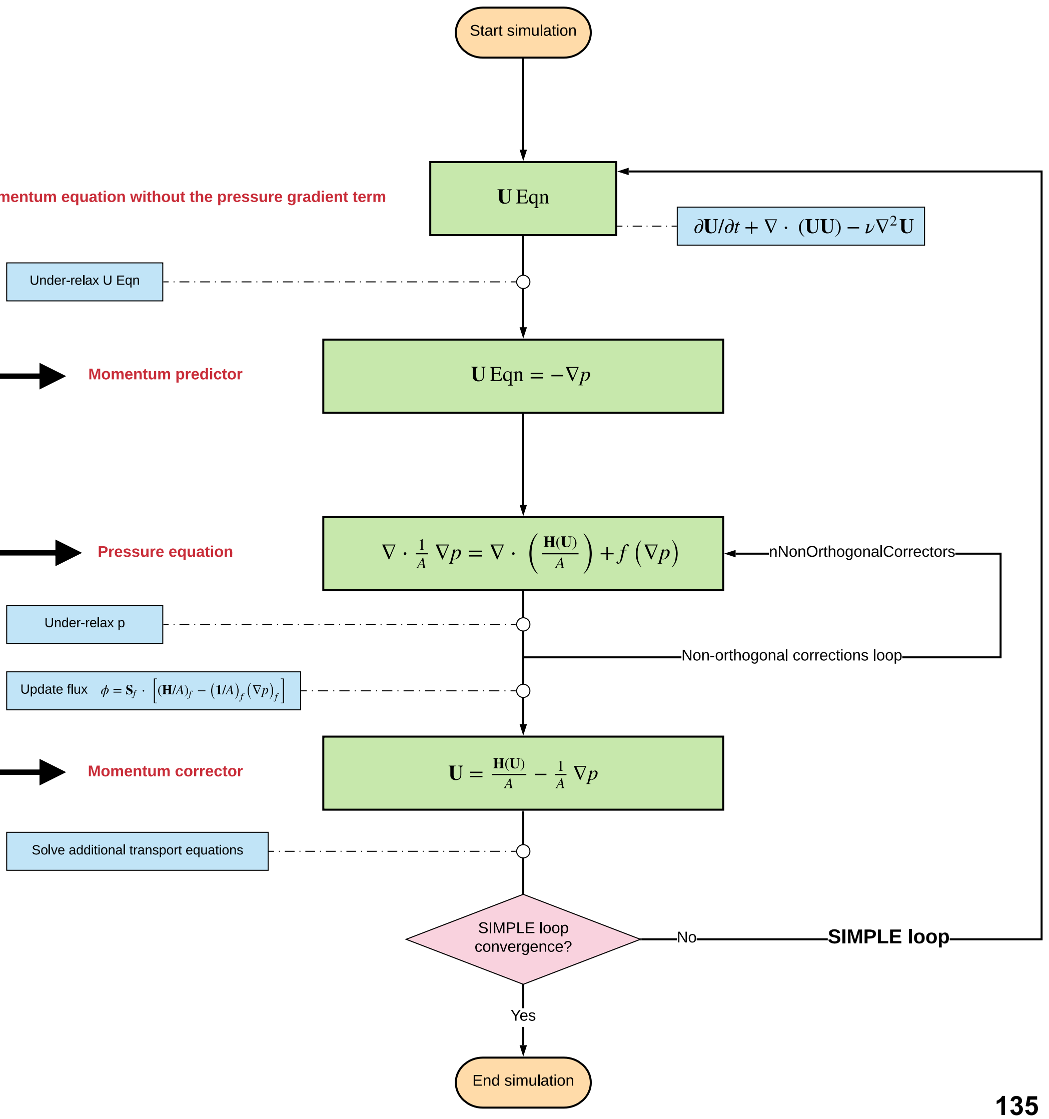
Momentum predictor

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

Pressure equation

```
U = HbyA – rAU*fvc::grad(p);
```

Momentum corrector



Start simulation
U Eqn
$\partial \mathbf{U}/\partial t + \nabla \cdot (\mathbf{UU}) - \nu \nabla^2 \mathbf{U}$
Under-relax U Eqn
$\mathbf{U}\,\mathrm{Eqn} = -\nabla p$
$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left(\frac{\mathbf{H(U)}}{A}\right) + f(\nabla p)$
nNonOrthogonalCorrectors
Under-relax p
Non-orthogonal corrections loop
Update flux $\phi = \mathbf{S}_f \cdot \left[(\mathbf{H}/A)_f - (1/A)_f (\nabla p)_f\right]$
$\mathbf{U} = \frac{\mathbf{H(U)}}{A} - \frac{1}{A}\nabla p$
Solve additional transport equations
SIMPLE loop convergence?
No
SIMPLE loop
Yes
End simulation

What else we did not cover?

The SIMPLE loop in OpenFOAM®

- The **SIMPLE** loop in OpenFOAM is controlled using the sub-dictionary *fvSolution*.
- 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;
}
```

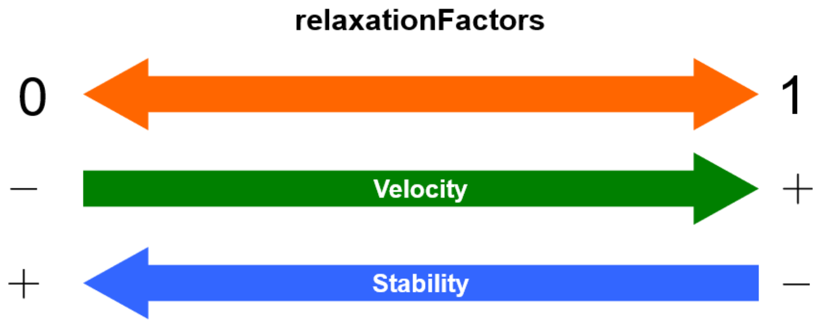
What else we did not cover?

The SIMPLE loop in OpenFOAM®

- The **SIMPLE** loop in OpenFOAM uses under-relaxation factors (URF).
- The under-relaxation factors control the change of the variable ϕ .

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

- 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 URF values, OpenFOAM® will not under-relax (this feature is off).
- If you set URF to 1, OpenFOAM ® will apply some under-relaxation to ensure diagonal dominance (this is not equivalent to off).
- 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.



What else we did not cover?

The SIMPLE loop in OpenFOAM®

- 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;
    }
}
```

Explicit under-relaxation

Implicit under-relaxation

SIMPLEC

```
relaxationFactors
{
    fields
    {
        p           1.0;
    }
    equations
    {
        p           1.0;
        U           0.9;
        k           0.9;
        omega       0.9;
    }
}
```

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

What else we did not cover?

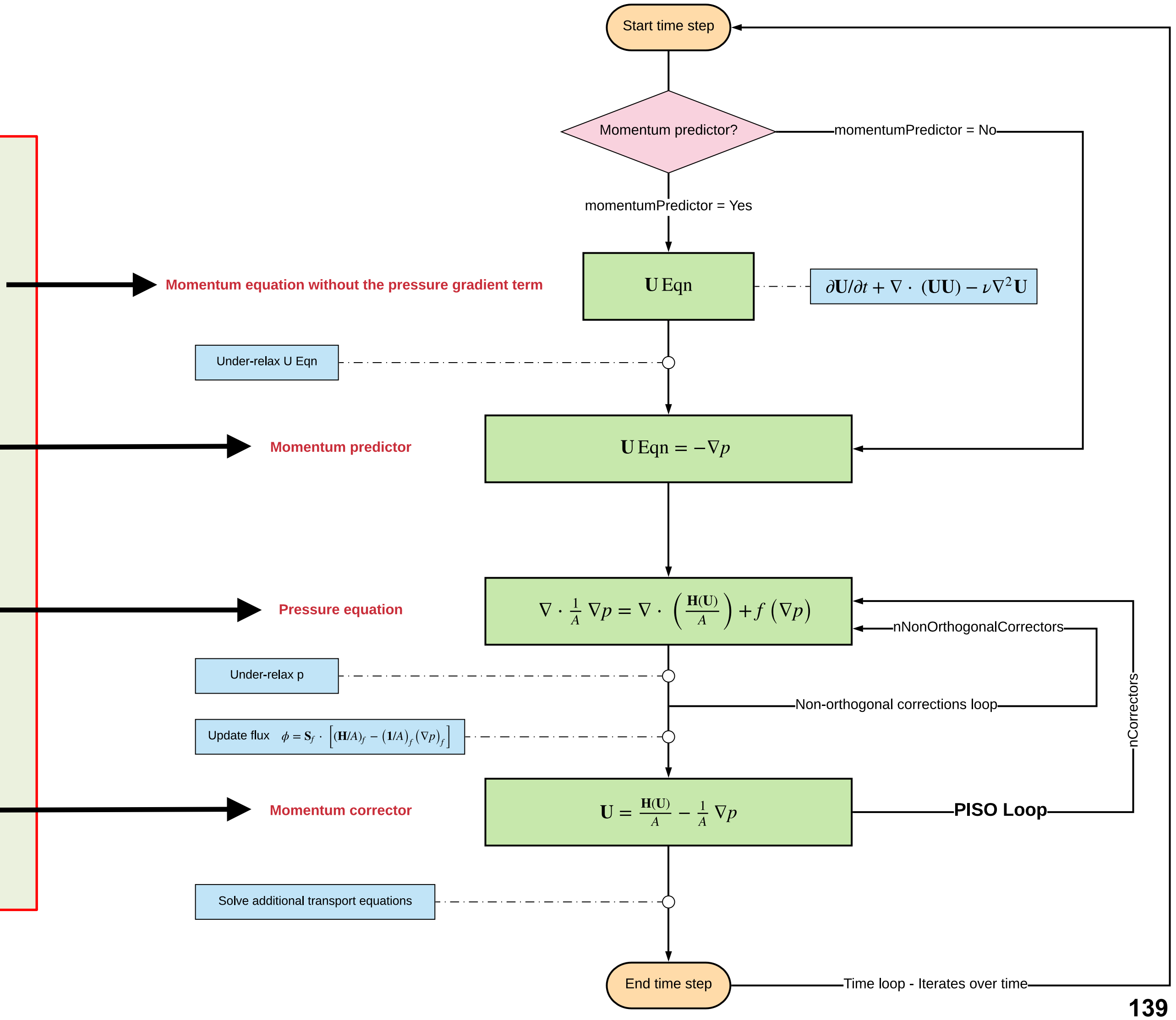
The PISO loop in OpenFOAM®

```
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);
```



Momentum equation without the pressure gradient term
Momentum predictor
Pressure equation
Momentum corrector
Start time step
Momentum predictor?
momentumPredictor = No
momentumPredictor = Yes
$\mathbf{U}$ Eqn
$\partial \mathbf{U}/\partial t + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nu \nabla^2 \mathbf{U}$
Under-relax U Eqn
$\mathbf{U}$ Eqn $= -\nabla p$
$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left(\frac{\mathbf{H(U)}}{A} \right) + f\left(\nabla p\right)$
nNonOrthogonalCorrectors
nCorrectors
Under-relax p
Non-orthogonal corrections loop
Update flux $\phi = \mathbf{S}_f \cdot \left[(\mathbf{H}/A)_f - \left(\mathbf{1}/A\right)_f \left(\nabla p\right)_f \right]$
$\mathbf{U} = \frac{\mathbf{H(U)}}{A} - \frac{1}{A}\nabla p$
PISO Loop
Solve additional transport equations
End time step
Time loop - Iterates over time

What else we did not cover?

The PISO loop in OpenFOAM®

- The **PISO** loop in OpenFOAM is controlled using the sub-dictionary *`fvSolution`*.
- The **PISO** method requires at least one correction (**nCorrectors**). To improve the accuracy and stability you can increase the number of corrections.
- For good accuracy and stability, it is recommended to use al 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;
}
```

What else we did not cover?

The PISO loop in OpenFOAM®

- You can use the optional keyword **momentumPredictor** to enable or disable the momentum predictor step.
- The momentum predictor helps in stabilizing the solution as we are computing 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 (high Reynolds number). If you are working with low Reynolds flow or creeping flows it is recommended to turn it off.
- 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).
- Also, if you want to use URF you will need to apply then to all field variables (including **.*Final**).

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

What else we did not cover?

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

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

→ **Momentum equation without the pressure gradient term**

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

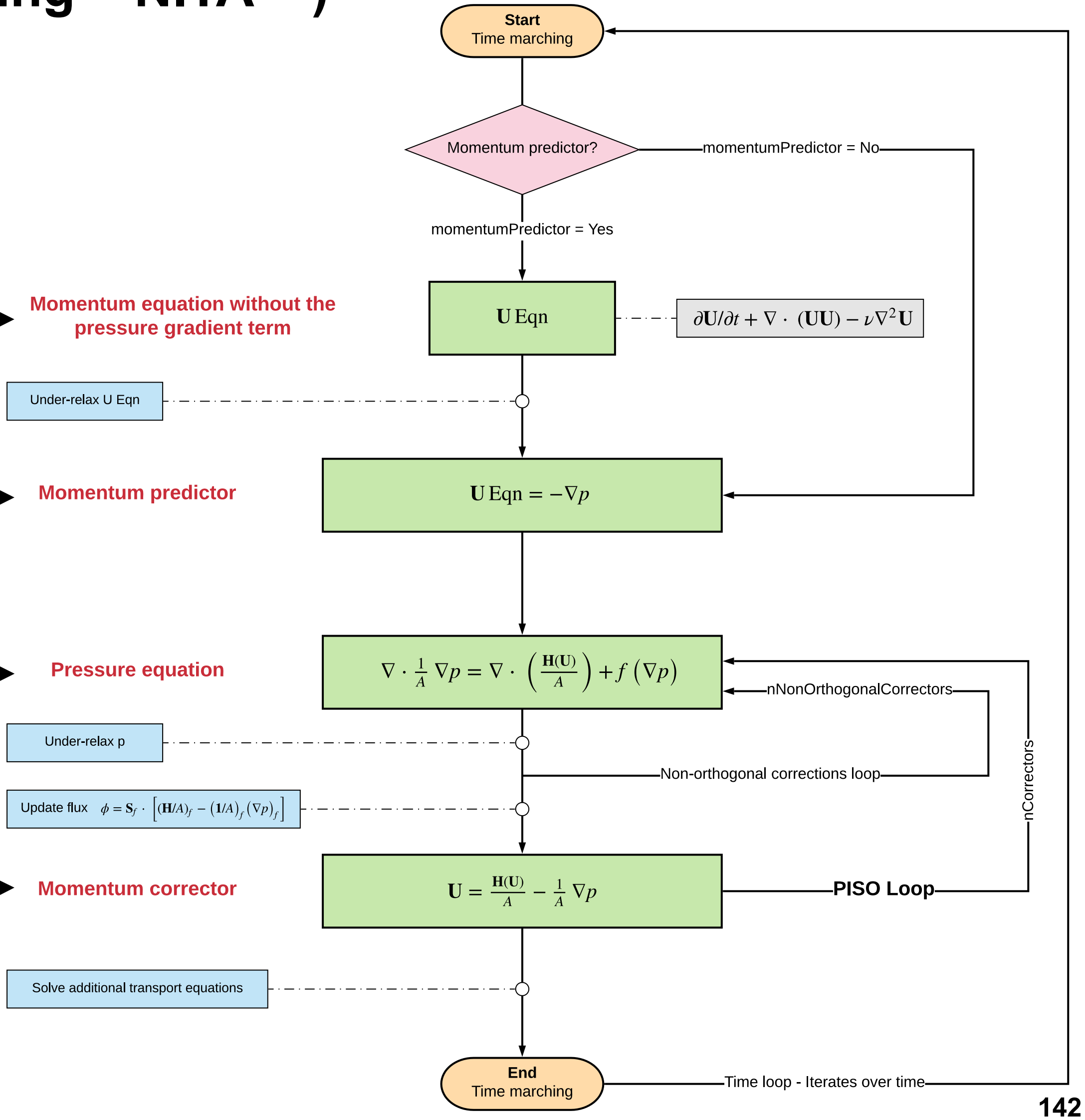
→ **Momentum predictor**

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

→ **Pressure equation**

```
U = HbyA – rAU*fvc::grad(p);
```

→ **Momentum corrector**


Start
Time marching
Momentum predictor?
momentumPredictor = No
momentumPredictor = Yes
U Eqn
$\partial \mathbf{U}/\partial t + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nu \nabla^2 \mathbf{U}$
Under-relax U Eqn
$\mathbf{U}\,\mathrm{Eqn} = -\nabla p$
$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left(\frac{\mathbf{H(U)}}{A}\right) + f\left(\nabla p\right)$
nNonOrthogonalCorrectors
nCorrectors
Under-relax p
Non-orthogonal corrections loop
Update flux $\phi = \mathbf{S}_f \cdot \left[(\mathbf{H}/A)_f - (1/A)_f (\nabla p)_f\right]$
$\mathbf{U} = \frac{\mathbf{H(U)}}{A} - \frac{1}{A}\nabla p$
PISO Loop
Solve additional transport equations
End
Time marching
Time loop - Iterates over time

What else we did not cover?

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

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

Momentum equation without the pressure gradient term

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

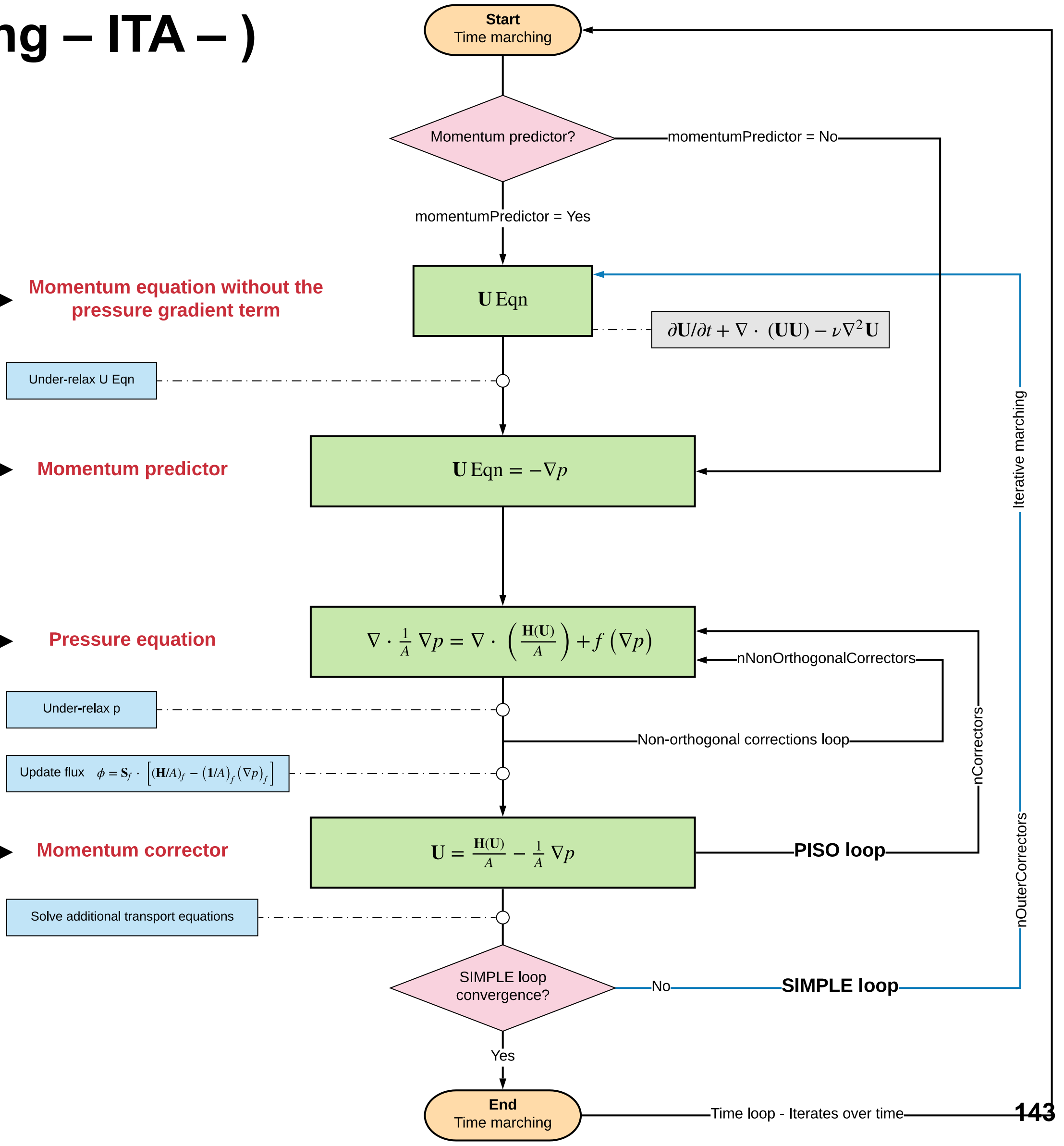
Momentum predictor

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

Pressure equation

```
U = HbyA – rAU*fvc::grad(p);
```

Momentum corrector

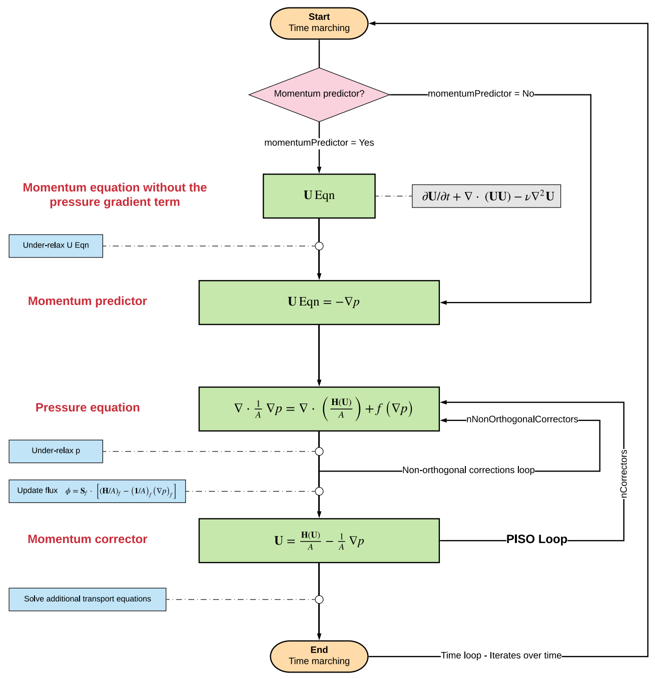


Start
Time marching
Momentum predictor?
momentumPredictor = No
momentumPredictor = Yes
U Eqn
$\partial \mathbf{U}/\partial t + \nabla \cdot (\mathbf{UU}) - \nu \nabla^2 \mathbf{U}$
Under-relax U Eqn
$\mathbf{U}\,\text{Eqn} = -\nabla p$
$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left(\frac{\mathbf{H(U)}}{A} \right) + f\,(\nabla p)$
nNonOrthogonalCorrectors
Under-relax p
Non-orthogonal corrections loop
Update flux $\phi = \mathbf{S}_f \cdot \left[(\mathbf{H}/A)_f - (\mathbf{1}/A)_f (\nabla p)_f \right]$
$\mathbf{U} = \frac{\mathbf{H(U)}}{A} - \frac{1}{A} \nabla p$
PISO loop
nCorrectors
Solve additional transport equations
SIMPLE loop convergence?
No
SIMPLE loop
nOuterCorrectors
Iterative marching
Yes
End
Time marching
Time loop - Iterates over time

What else we did not cover?

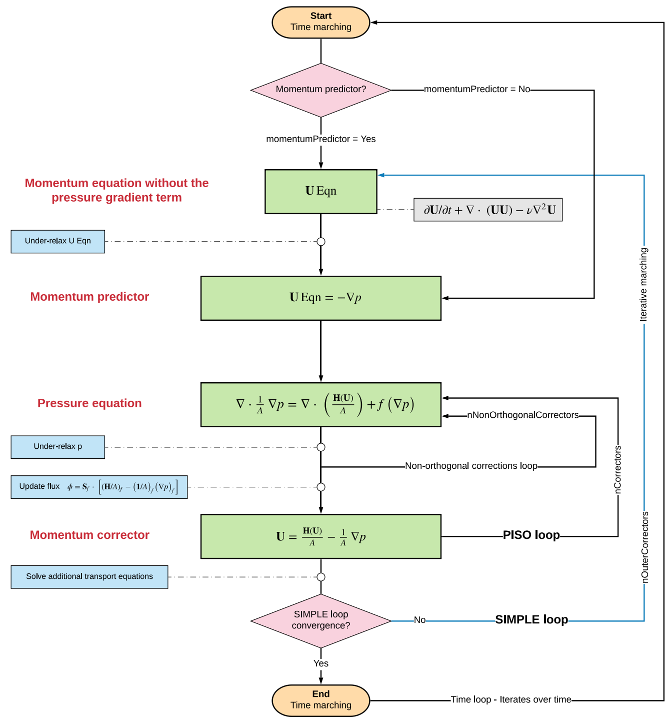
PISO-NITA and PISO-ITA comparison

- The main difference between both methods is the outer loop 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).



Start
Time marching
Momentum predictor?
momentumPredictor = No
momentumPredictor = Yes
Momentum equation without the pressure gradient term
U Eqn
$\partial \mathbf{U}/\partial t + \nabla \cdot (\mathbf{UU}) - \nu \nabla^2 \mathbf{U}$
Under-relax U Eqn
Momentum predictor
$\mathbf{U}\,\text{Eqn} = -\nabla p$
Pressure equation
$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left(\frac{\mathbf{H(U)}}{A}\right) + f(\nabla p)$
nNonOrthogonalCorrectors
Under-relax p
Non-orthogonal corrections loop
Update flux $\phi = \mathbf{S}_f \cdot \left[(\mathbf{H}/A)_f - (1/A)_f (\nabla p)_f\right]$
nCorrectors
Momentum corrector
$\mathbf{U} = \frac{\mathbf{H(U)}}{A} - \frac{1}{A}\nabla p$
PISO Loop
Solve additional transport equations
End
Time marching
Time loop - Iterates over time

PISO-NITA



Start
Time marching
Momentum predictor?
momentumPredictor = No
momentumPredictor = Yes
Momentum equation without the pressure gradient term
U Eqn
$\partial \mathbf{U}/\partial t + \nabla \cdot (\mathbf{UU}) - \nu \nabla^2 \mathbf{U}$
Under-relax U Eqn
Iterative marching
Momentum predictor
$\mathbf{U}\,\text{Eqn} = -\nabla p$
Pressure equation
$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left(\frac{\mathbf{H(U)}}{A}\right) + f(\nabla p)$
nNonOrthogonalCorrectors
Under-relax p
Non-orthogonal corrections loop
Update flux $\phi = \mathbf{S}_f \cdot \left[(\mathbf{H}/A)_f - (1/A)_f (\nabla p)_f\right]$
nCorrectors
nOuterCorrectors
Momentum corrector
$\mathbf{U} = \frac{\mathbf{H(U)}}{A} - \frac{1}{A}\nabla p$
PISO loop
Solve additional transport equations
SIMPLE loop convergence?
No
SIMPLE loop
Yes
End
Time marching
Time loop - Iterates over time

PISO-ITA (PIMPLE in OpenFOAM®)

What else we did not cover?

The PIMPLE loop in OpenFOAM®

- The **PIMPLE** loop in OpenFOAM is controlled using the sub-dictionary *fvSolution*.
- 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 3.
- Have in mind that increasing the number of **nOterCorrectors** will highly increase the computational cost.

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

What else we did not cover?

The PIMPLE loop in OpenFOAM®

- You can use under-relaxation factors (URF) with the **PIMPLE** solvers.
- 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 for steady simulation.
- 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 can assign URF to all variables (including **.*Final**), to only the intermediate field variables (**U**, **p**, **k**, and so on), or to only the **.*Final** variables (**UFinal**, **pFinal**, **kFinal**, and so on).
- We recommend to use URF in all variables.

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

What else we did not cover?

The PIMPLE loop in OpenFOAM®

- You can use the following guidelines to define the URF with the **PIMPLE** family of solvers.
- These guidelines also applies for the **SIMPLE** 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).

What else we did not cover?

The PIMPLE loop in OpenFOAM®

- You can use the following guidelines to define the URF with the **PIMPLE** family of solvers.
- These guidelines also applies for the **SIMPLE** 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).

This page intentionally left blank

Roadmap

1. ~~Important concepts to remember~~
2. ~~The Finite Volume Method: An overview~~
3. ~~The FVM in OpenFOAM®: some implementation details and computational pointers~~
4. ~~Some kind of conclusion~~
5. ~~What else we did not cover?~~
6. Goodbye

This is the end

That was only the tip of the iceberg

Now the rest is on you

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. 2006, Elsevier Science
  - **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.

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.

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.

FYI – Useful links

- You can download the tutorials at this link:
  - http://www.wolfdynamics.com/training/OF_WS2020/OF2020training.tar.gz

- The tutorials are based on OpenFOAM® version 7.

- To run some of the postprocessing script you will need Python version 3.7, and the following Python libraries:
  - Numpy
  - Pandas
  - Matplotlib.

Be collaborative, be innovative, be cloud

wolf dynamics

multiphysics simulations,
optimization & data analytics

Let's connect

guerrero@wolfdynamics.com

www.wolfdynamics.com