SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

Introduction to the Finite Volume Method

Simone Bnà, Cineca

09/2022

Univerza v Ljubljani

TECHNISCHE UNIVERSITÄT WIEN

CINECA

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Co-funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission.
This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Disclaimer and Acknowledgements

Simone Bnà, Cineca

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

"This material is based on *An introduction to Computational Fluid Dynamics using OpenFOAM with advanced topics* by Riccardo Rossi, Head and Founder of RED Fluid Dynamics, March/April 2021".

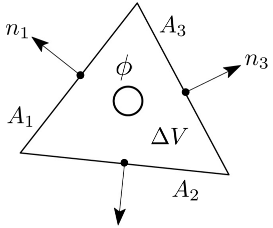
The scalar transport equation

We use the **finite-volume method** (FVM) to solve the flow governing equations. The **integral form** of the scalar transport equation (STE) must be discretized and solved:

$$\int_V \frac{\partial \phi}{\partial t}\, dV + \int_A (u_j \phi) n_j \, dA = \int_A D \frac{\partial \phi}{\partial x_j} n_j \, dA + \int_V (S_\phi)\, dV$$

Discretization steps:

- Numerical integration
- Time-advancement schemes
- Differentiation schemes
- Interpolation schemes



The semi-discrete transport equation

Simone Bnà, Cineca

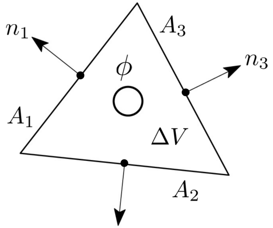
SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

We use the **midpoint rule** as numerical method for approximating the surface and volume integrals that appear in the scalar transport equation and is valid for a generic **polyhedral cell**. In formula we have:

$$\frac{\partial}{\partial t}(\phi \Delta V) + \sum_f (u_j \phi)_f A_{fj} = \sum_f D_f \left.\frac{\partial \phi}{\partial x_j}\right|_f A_{fj} + S_\phi \Delta V$$

where A_{fj} is the **face area vector**:

$$A_{fj} = \Delta A_f n_j$$



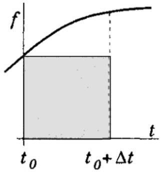
Time integration schemes I

Simone Bnà, Cineca

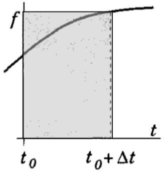
Time integration schemes provides a numerical approximation to the **time rate of change** of a field variable:

$$\int_{t_n}^{t_{n+1}} \frac{\partial \phi}{\partial t} dt = [\phi^{n+1} - \phi^{n}] = \frac{1}{\Delta V} \int_{t_n}^{t_{n+1}} f(t, \phi(t))\, dt$$

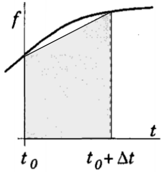
Forward Euler



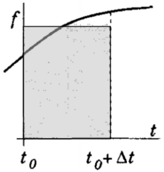
Backward Euler



Trapezoidal rule



Midpoint rule



Time integration schemes II

Simone Bnà, Cineca

The backward Euler and the trapezoidal rule, or Crank-Nicolson scheme, are **implicit schemes** and they are **unconditionally stable**, whereas the forward Euler scheme is only conditionally stable.

The stability condition for the forward (explicit) Euler scheme in convection dominated flows is associated with the **Courant-Friedrichs-Lewis** number (CFL):

$$CFL = \frac{u\Delta t}{\Delta x} \leq 1$$

where Δx and Δt are the grid spacing and time-step size, respectively.

The discrete transport equation

Simone Bnà, Cineca

SCtrain SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

If we **integrate in time** the semi-discrete transport equation and we apply the unconditionally stable backward Euler scheme we get:

$$(\phi^{n+1} - \phi^n)\Delta V + \sum_f (u_j^n \phi^{n+1})_f A_{fj} = \sum_f D_f \frac{\partial \phi^{n+1}}{\partial x_j}\bigg|_f A_{fj} + S_\phi^{n+1}\Delta V$$

Does the **CFL** condition need to be satisfied?

OpenFOAM software

Simone Bnà, Cineca

OpenFOAM is a **free, open source CFD software package** written in C++ and released free and open-source under the GNU General Public License by the OpenFOAM Foundation.

```
/*---------------------------------*- C++ -*-----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2012                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*-----------------------------------------------------------------------------*/
```

OpenFOAM features I

Simone Bnà, Cineca

OpenFOAM is based on the same features of **state-of-the-art commercial CFD** software(Ansys Fluent, Star-CCM+, ...):

- Finite Volume framework
- Collocated unstructured polyhedral meshes
- Second-order accurate in space and time
- Massive parallel computations via domain decomposition and MPI

OpenFOAM features II

Simone Bnà, Cineca

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

OpenFOAM uses **equation mimicking** to perform field algebra and discretisation, which resembles mathematical notation. For the Navier-Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} + (\nabla \cdot \mathbf{u}\mathbf{u}) - \nu \Delta \mathbf{u} = -\nabla \left(\frac{p}{\rho} \right)$$

the following code is used:

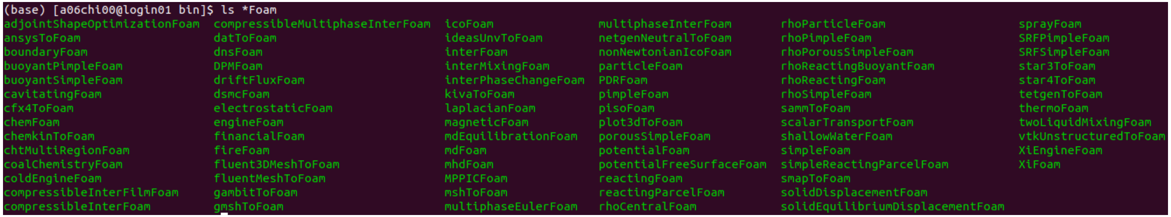
```
  fvm::ddt(U)
+ fvm::div(phi, U)
- fvm::laplacian(nu, U)
== -fvc::grad(p))
```

OpenFOAM solvers

Simone Bnà, Cineca

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

OpenFOAM provides a **wide list of solvers** for complex fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics, ...



```
(base) [a06chi00@login01 bin]$ ls *Foam
adjointShapeOptimizationFoam  compressibleMultiphaseInterFoam  icoFoam               multiphaseInterFoam       rhoParticleFoam                   sprayFoam
ansysToFoam                   datToFoam                        ideasUnvToFoam        netgenNeutralToFoam       rhoPimpleFoam                     SRFPimpleFoam
boundaryFoam                  dnsFoam                          interFoam             nonNewtonianIcoFoam       rhoPorousSimpleFoam               SRFSimpleFoam
buoyantPimpleFoam             DPMFoam                          interMixingFoam       particleFoam              rhoReactingBuoyantFoam            star3ToFoam
buoyantSimpleFoam             driftFluxFoam                    interPhaseChangeFoam  PDRFoam                   rhoReactingFoam                   star4ToFoam
cavitatingFoam                dsmcFoam                         kivaToFoam            pimpleFoam                rhoSimpleFoam                     tetgenToFoam
cfx4ToFoam                    electrostaticFoam                laplacianFoam         pisoFoam                  sammToFoam                        thermoFoam
chemFoam                      engineFoam                       magneticFoam          plot3dToFoam              scalarTransportFoam               twoLiquidMixingFoam
chemkinToFoam                 financialFoam                    mdEquilibrationFoam   porousSimpleFoam          shallowWaterFoam                  vtkUnstructuredToFoam
chtMultiRegionFoam            fireFoam                         mdFoam                potentialFoam             simpleFoam                        XiEngineFoam
coalChemistryFoam             fluent3DMeshToFoam               mhdFoam               potentialFreeSurfaceFoam  simpleReactingParcelFoam          XiFoam
coldEngineFoam                fluentMeshToFoam                 MPPICFoam             reactingFoam              smapToFoam
compressibleInterFilmFoam     gambitToFoam                     mshToFoam             reactingParcelFoam        solidDisplacementFoam
compressibleInterFoam         gmshToFoam                       multiphaseEulerFoam   rhoCentralFoam            solidEquilibriumDisplacementFoam
```

icoFoam solver

Simone Bnà, Cineca

The icoFoam solver is a **pressure-based** solver for **time-dependent** and **incompressible laminar flows**:

$$\nabla \cdot \mathbf{u} = 0$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \nu \Delta \mathbf{u} = -\nabla \left(\frac{p}{\rho} \right)$$

Segregated solver: solve for $\mathbf{u}$ and p separately.
Since the continuity equation provides a **kinematic constraint** only on the velocity field, an **evolution equation** for pressure has to be derived.
OpenFOAM uses the so-called **pressure-velocity coupling** algorithms.

Deriving the pressure equation I

The **momentum equation(s)** in general **matrix form** is:

$$\boldsymbol{M}\mathbf{u} = -\nabla p$$

where $\boldsymbol{M} = \boldsymbol{M}(\mathbf{u})$ is the matrix of coefficients arising from **time and spatial discretization**.

$$\begin{pmatrix} M_{1,1} & M_{1,2} & M_{1,3} & \dots & M_{1,n} \\ M_{2,1} & M_{2,2} & M_{2,3} & \dots & M_{2,n} \\ M_{3,1} & M_{3,2} & M_{3,3} & \dots & M_{3,n} \\ \dots & \dots & \dots & \dots & \dots \\ M_{n,1} & M_{n,2} & M_{n,3} & \dots & M_{n,n} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \dots \\ \mathbf{u}_n \end{pmatrix} = - \begin{pmatrix} (\nabla p)_1 \\ (\nabla p)_2 \\ (\nabla p)_3 \\ \dots \\ (\nabla p)_n \end{pmatrix}$$

Note that the coefficients matrix is a function of the velocity field due to the **non linearity** of the momentum equation.

Deriving the pressure equation II

Simone Bnà, Cineca

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

Using the available pressure from previous calculation step or from initial conditions, the velocity field can be computed by solving the momentum equation, where the **pressure gradient** is an **explicit source term** collocated on the right hand side (RHS).

In the framework of implicit schemes, however, this velocity field **will not satisfy the continuity equation** because the pressure gradient is associated with the previous calculation step. The **computed, or predicted, velocity** field will only represent a **guess**. This stage of the pressure-velocity coupling algorithm is called **momentum predictor**.

$$\boldsymbol{M}\mathbf{u}^{*} = -\nabla p$$

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

Deriving the pressure equation III

Simone Bnà, Cineca

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

In order to obtain a diverge-free velocity field a pressure equation is derived using the continuity constraint.
Firstly, a diagonal matrix is extracted from the matrix of coefficients of the momentum equation, which can be easily inverted:

$$\boldsymbol{A} = \begin{pmatrix} M_{1,1} & 0 & 0 & \dots & 0 \\ 0 & M_{2,2} & 0 & \dots & 0 \\ 0 & 0 & M_{3,3} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & M_{n,n} \end{pmatrix} \quad \boldsymbol{A}^{-1} = \begin{pmatrix} 1/A_{1,1} & 0 & 0 & \dots & 0 \\ 0 & 1/A_{2,2} & 0 & \dots & 0 \\ 0 & 0 & 1/A_{3,3} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1/A_{n,n} \end{pmatrix}$$

The corresponding code providing the reciprocal of the diagonal component is:

```
    volScalarField rUA = 1.0/UEqn().A()
```

Deriving the pressure equation IV

Simone Bnà, Cineca

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

In the second step, the residual vector ***H*** is computed by subtracting the diagonal component from the momentum equations:

$$\boldsymbol{H} = \boldsymbol{A}\mathbf{u} - \boldsymbol{M}\mathbf{u}$$

$$\boldsymbol{H} = -\begin{pmatrix} 0 + M_{1,2}\mathbf{u} + M_{1,3}\mathbf{u} + \ldots + M_{1,n}\mathbf{u} \\ M_{2,1}\mathbf{u} + 0 + M_{2,3}\mathbf{u} + \ldots + M_{2,n}\mathbf{u} \\ M_{3,1}\mathbf{u} + M_{3,2}\mathbf{u} + 0 + \ldots + M_{3,n}\mathbf{u} \\ \ldots \\ M_{n,1}\mathbf{u} + M_{n,2}\mathbf{u} + M_{n,3}\mathbf{u} + \ldots + 0 \end{pmatrix}$$

The residual vector will represent a source term for the pressure equation.

Deriving the pressure equation V

Simone Bnà, Cineca

From the definition of the residual vector and recalling that $\boldsymbol{M}\mathbf{u} = -\nabla p$, the **velocity field** can be calculated as follows:

$$\boldsymbol{H} = \boldsymbol{A}\mathbf{u} + \nabla p \longrightarrow \boldsymbol{A}\mathbf{u} = \boldsymbol{H} - \nabla p$$
$$\boldsymbol{A}^{-1}\boldsymbol{A}\,\mathbf{u} = \boldsymbol{A}^{-1}\boldsymbol{H} - \boldsymbol{A}^{-1}\nabla p \longrightarrow \mathbf{u} = \boldsymbol{A}^{-1}\boldsymbol{H} - \boldsymbol{A}^{-1}\nabla p$$

Using the continuity equation, the **pressure equation** is finally obtained:

$$\nabla \cdot \mathbf{u} = 0 \longrightarrow \nabla \cdot (\boldsymbol{A}^{-1}\boldsymbol{H} - \boldsymbol{A}^{-1}\nabla p) = 0$$
$$\nabla \cdot (\boldsymbol{A}^{-1}\nabla p) = \nabla \cdot (\boldsymbol{A}^{-1}\boldsymbol{H})$$

the solution of the PDE provides the new pressure field obeying to the continuity equation.

Pressure equation in OpenFOAM

Simone Bnà, Cineca

SCtrain SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

In the OpenFOAM literature, the pressure equation is often reported as follows:

$$\nabla \cdot (\boldsymbol{A}^{-1} \nabla p) = \nabla \cdot (\boldsymbol{A}^{-1} \boldsymbol{H}) \equiv \nabla \cdot \left(\frac{1}{a_p} \nabla p \right) = \nabla \cdot \left(\frac{\boldsymbol{H}(\mathbf{u})}{a_p} \right)$$

The pressure equation represents a **Poisson-like equation** where the source term is given by the residual vector of the momentum equation.

The corresponding OpenFOAM code to assemble the pressure equation is the following:

```
volScalarField rUA = 1.0/UEqn().A()
HbyA = rUA*UEqn().H()
fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
```

Pressure-velocity coupling

Simone Bnà, Cineca

SCtrain SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

The complete solution procedure for the continuity and momentum equations is the following:

- Solve the **momentum equation** in the **predictor step** $\longrightarrow \boldsymbol{M}\mathbf{u} = -\nabla p$
- Compute the **residual vector** $\longrightarrow \boldsymbol{H} = \boldsymbol{A}\mathbf{u} - \boldsymbol{M}\mathbf{u}$
- Solve the **pressure equation** $\longrightarrow \nabla \cdot (\boldsymbol{A}^{-1}\nabla p) = \nabla \cdot (\boldsymbol{A}^{-1}\boldsymbol{H})$
- Compute the new velocity field in the **corrector step** $\longrightarrow \mathbf{u} = \boldsymbol{A}^{-1}\boldsymbol{H} - \boldsymbol{A}^{-1}\nabla p$

The new velocity field satisfies the continuity equation. However, the pressure equation is no longer satisfied since the source term represented by the residual vector depends on the velocity field and it has changed.

An **iterative procedure** is needed to solve the coupled pressure-velocity system of equations.

The PISO algorithm

Simone Bnà, Cineca

SCtrain SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

In the PISO (**Pressure Implicit with Splitting of Operator**) algorithm the momentum predictor step is performed once per time-step:

1. Start the time loop or a new time step
2. Solve the **momentum equation** in the **predictor step** $\longrightarrow \boldsymbol{M}\mathbf{u} = -\nabla p$
3. Compute the **residual vector** $\longrightarrow \boldsymbol{H} = \boldsymbol{A}\mathbf{u} - \boldsymbol{M}\mathbf{u}$
4. Solve the **pressure equation** $\longrightarrow \nabla \cdot (\boldsymbol{A}^{-1}\nabla p) = \nabla \cdot (\boldsymbol{A}^{-1}\boldsymbol{H})$
5. Compute the new velocity field in the **corrector step** $\longrightarrow \mathbf{u} = \boldsymbol{A}^{-1}\boldsymbol{H} - \boldsymbol{A}^{-1}\nabla p$
6. If pressure converged go to 1, if not converged go to 3

The **inner loop** (3-6) performed to update the residual vector with the new velocity field is called **pressure-correction loop**. In OpenFOAM, the number of correctors (*nCorrectors*) to be performed per time-step is specified in the *fvSolution* dictionary in the *system* folder.

Courant number limit I

Simone Bnà, Cineca

The Navier-Stokes equations contains **two non-linearity**: the **convection term** and the **pressure velocity coupling**.
The non-linear convection term can be treated via an iterative approach using the Picard's method:

$$(\mathbf{u}^{n+1} \cdot \nabla)\mathbf{u}^{n+1} \approx (\mathbf{u}^{n} \cdot \nabla)\mathbf{u}^{n+1} + \mathcal{O}(\Delta t^2)$$

where $\mathbf{u}^n$ and $\mathbf{u}^{n+1}$ are the previous (available) and the new (estimated) solutions, respectively.

Courant number limit II

Simone Bnà, Cineca

The characteristic of the **PISO algorithm** is that the momentum equation is updated only once per time-step. Since the coefficients of the matrix $\boldsymbol{M} = \boldsymbol{M}(\mathbf{u})$ contains the convection velocity, this means that **fluxes are kept frozen** during the correction steps.

This is only possible at low Courant numbers (very small time steps):

$$CFL = \frac{u\Delta t}{\Delta x} \leq 1$$

although implicit in time!

a condition where the **pressure-velocity coupling** has a much **stronger impact** on the momentum equation than the non-linearity arising from the convection term.

Steady-state simulations

Simone Bnà, Cineca

In a **steady-state problem**, a **large time-step** size could be used because there is no need for obtaining accurate (i.e. divergence-free velocity field) solutions at the end of each time step.

However, the **stability constraint** given by the CFL condition (CFL $\leq$ 1) associated with the frozen convection velocity in the **PISO loop** does not allow for the use of a large time-step size. Therefore, a different algorithm should be used when the transient solution is not of interest.
The **SIMPLE** [1] (Semi-Implicit Method for Pressure-Linked Equations) is the most popular algorithm to obtain a steady-state solution to the Navier-Stokes equations.

[1] S.V. Patankar, D.B. Spalding, A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows, Int. J. Heat Mass Transfer 15 (10) (1972) 1787-1806.

The SIMPLE algorithm

Simone Bnà, Cineca

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

In the **SIMPLE** algorithm, the **momentum predictor** step is performed at **every pseudo-time iteration**:

1. Start the time loop or a new time step
2. Solve the **momentum equation** in the **predictor step** $\longrightarrow \boldsymbol{M}\mathbf{u} = -\nabla p$
3. Compute the **residual vector** $\longrightarrow \boldsymbol{H} = \boldsymbol{A}\mathbf{u} - \boldsymbol{M}\mathbf{u}$
4. Solve the **pressure equation** $\longrightarrow \nabla \cdot (\boldsymbol{A}^{-1}\nabla p) = \nabla \cdot (\boldsymbol{A}^{-1}\boldsymbol{H})$
5. Compute the new velocity field in the **corrector step** $\longrightarrow \mathbf{u} = \boldsymbol{A}^{-1}\boldsymbol{H} - \boldsymbol{A}^{-1}\nabla p$
6. Go to 1

Furthermore, since a **divergence-free velocity field** is expected only at the end of the pseudo-time loop, the corrector or **inner loop is dropped** (see PISO algorithm) and only the **outer loop is performed** at each every iteration.

Under-relaxation I

Simone Bnà, Cineca

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

Since the **time-derivative is dropped** in the SIMPLE algorithm the **diagonal-dominance** of the momentum equation is **reduced**, potentially leading to an **unstable solution**.
Under relaxation is thus used in the momentum and other scalar equations to artificially increase the diagonal-dominance and improve stability:

$$a_p \mathbf{u}_p^{n+1} + \sum_{nb} a_{nb} \mathbf{u}_{nb}^{n+1} = b_p$$

$$\frac{1-\alpha}{\alpha} a_p \mathbf{u}_p^{n+1} + a_p \mathbf{u}_p^{n+1} + \sum_{nb} a_{nb} \mathbf{u}_{nb}^{n+1} = b_p + \frac{1-\alpha}{\alpha} a_p \mathbf{u}_p^{n}$$

$$\frac{1}{\alpha} a_p \mathbf{u}_p^{n+1} + \sum_{nb} a_{nb} \mathbf{u}_{nb}^{n+1} = b_p + \frac{1-\alpha}{\alpha} a_p \mathbf{u}_p^{n}$$

where $0 < \alpha \leq 1$ is the **under-relaxation factor**.

Under-relaxation II

Simone Bnà, Cineca

In OpenFOAM, the **under-relaxation factors** are set in the *fvSolution* dictionary located in the *system* directory, where both fields and equations factors can be used.

The **equations under-relaxation factors**, are used to manipulate the coefficients matrix and improve its **diagonal-dominance**.
Fields under-relaxation factors can be also specified to limit the change in the velocity and pressure fields during the **pressure velocity-coupling**:

$$p^{n+1} = p^n + \alpha_p p^{'}$$
$$u^{n+1} = \alpha_u \mathbf{u}^* + (1 - \alpha_u)\mathbf{u}^n$$

where α_p and α_u are the **fields under-relaxation factor** and $p^{'}$ and $\mathbf{u}^*$ are the **corrected pressure and velocity values**.

The simpleFoam solver

Simone Bnà, Cineca

The **simpleFoam solver** is a **pressure-based** solver for **steady-state** computations of **incompressible laminar** or **turbulent flows** within the framework of RANS modeling:

$$\nabla \cdot \mathbf{u} = 0$$

$$(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla \cdot \boldsymbol{R} - \nu \Delta \mathbf{u} = -\nabla \left(\frac{p}{\rho} \right)$$

where ***R*** is the **Reynolds stress tensor**, accounting for the turbulent component of the flow and requiring a **turbulence model** to be estimated.

Time stepping vs iterating

Simone Bnà, Cineca

The **icoFoam** and **simpleFoam** solvers have similar behavior because there is a **strong similarity** in the system of algebraic equations solved by the two methods.

For a generic transport equation, it can be shown that the **time step-size** in an implicit time-marching technique and the **under-relaxation factor** in the iterative solution technique relate each other as follows:

$$\Delta t = \frac{\alpha \Delta V}{a_p(1-\alpha)}, \qquad \alpha = \frac{a_p \Delta t}{a_p \Delta t + \Delta V}$$

where a_p and ΔV are the central coefficient of the algebraic equation and the cell volume, respectively.

Thank you for your attention!