PISO in icoFoam

PISO in icoFoam

- The `icoFoam` directory (`$FOAM_SOLVERS/incompressible/icoFoam`) consists of the following:

```
createFields.H  Make/  icoFoam.C
```

- The `Make` directory contains instructions for the `wmake` compilation command.
- `icoFoam.C` is the main file, and `createFields.H` is an inclusion file, which is included in `icoFoam.C`.

We have a look at a part of the description in `icoFoam.C`:

```
Description
    Transient solver for incompressible, laminar flow of Newtonian fluids.

    The solver uses the PISO algorithm ...
```

We will here discuss the PISO algorithm in general and the way it is done in `icoFoam`, for transient incompressible laminar flow of Newtonian fluids.

Also see PhD thesis by Tessa Uroic, for FOAM-extend (slightly different).

PISO in icoFoam: Governing equations

- The incompressible continuity and momentum equations are given by

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

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

  The kinematic viscosity, ν is constant for Newtonian flow.
  The kinematic pressure, p, is the static pressure divided by the constant density, ρ.
  In `icoFoam` cases we don't specify ρ, so we remember that we solve for *kinematic pressure*.
  Only pressure *gradient* affects the momentum eqs., so level of pressure is not important.
  The non-linear convection term is linearized and evaluated using Gauss' theorem, as `Foam::div(phi, U)`, where the face flux field `phi` is taken from the previous time step/iteration.

- Unknowns are u and p, but there is no pressure equation.
  The continuity equation imposes a scalar constraint on the momentum equation (since $\nabla \cdot \mathbf{u}$ is a scalar).
  We use the continuity and momentum equations to derive a pressure equation ...

PISO in icoFoam: Discretization of the momentum equation

- Discretizing the linearized momentum equation, while keeping the pressure gradient in its original form (in FOAM-extend the time term is here excluded, see Tessa Uroic's PhD thesis):

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots a_{3,n} \\ \vdots & \vdots & \vdots\ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots a_{n,n} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_n \end{bmatrix} + \begin{bmatrix} (\partial p/\partial x)_1 \\ (\partial p/\partial x)_2 \\ (\partial p/\partial x)_3 \\ \vdots \\ (\partial p/\partial x)_n \end{bmatrix}$$

Matrix decomposition

$$\underbrace{\begin{bmatrix} a_{1,1} & 0 & 0 & \dots 0 \\ 0 & a_{2,2} & 0 & \dots 0 \\ 0 & 0 & a_{3,3} & \dots 0 \\ \vdots & \vdots & \vdots\ddots & \vdots \\ 0 & 0 & 0 & \dots a_{n,n} \end{bmatrix}}_{\text{Diagonal } \mathbf{A}} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & a_{1,2} & a_{1,3} & \dots a_{1,n} \\ a_{2,1} & 0 & a_{2,3} & \dots a_{2,n} \\ a_{3,1} & a_{3,2} & 0 & \dots a_{3,n} \\ \vdots & \vdots & \vdots\ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots 0 \end{bmatrix}}_{\text{Off diagonal}} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_n \end{bmatrix} + \begin{bmatrix} (\partial p/\partial x)_1 \\ (\partial p/\partial x)_2 \\ (\partial p/\partial x)_3 \\ \vdots \\ (\partial p/\partial x)_n \end{bmatrix}$$

$$a_P^{\mathbf{u}} \mathbf{u}_P + \sum_N a_N^{\mathbf{u}} \mathbf{u}_N = \mathbf{r} - \nabla p$$

PISO in icoFoam: Derivation of the pressure equation

- Discretized linearized momentum equation, keeping the pressure gradient:

$$a_P^{\mathbf{u}}\mathbf{u}_P + \sum_N a_N^{\mathbf{u}}\mathbf{u}_N = \mathbf{r} - \nabla p$$

  Here, $\mathbf{r}$ is a source term (may have contributions from the discretized time-term).

- Introduce the $\mathbf{H}(\mathbf{u})$ operator:

$$\mathbf{H}(\mathbf{u}) = \mathbf{r} - \sum_N a_N^{\mathbf{u}}\mathbf{u}_N$$

  so that:

$$\begin{aligned} a_P^{\mathbf{u}}\mathbf{u}_P &= \mathbf{H}(\mathbf{u}) - \nabla p \\ \mathbf{u}_P &= (a_P^{\mathbf{u}})^{-1}(\mathbf{H}(\mathbf{u}) - \nabla p) \end{aligned}$$

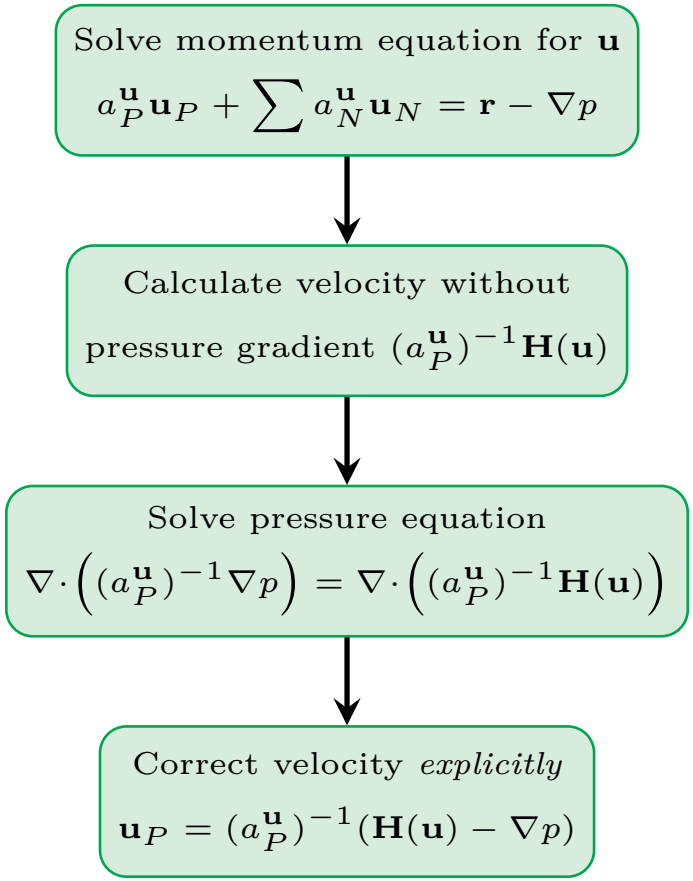
- Substitute this in the incompressible continuity equation ($\nabla \cdot \mathbf{u} = 0$) to get a pressure equation for incompressible flow:

$$\nabla \cdot \left[(a_P^{\mathbf{u}})^{-1}\nabla p\right] = \nabla \cdot \left[(a_P^{\mathbf{u}})^{-1}\mathbf{H}(\mathbf{u})\right]$$

  The R.H.S. is the continuity error of the velocity field without the pressure gradient.
  It is given by summing the fluxes through the faces, interpolated to the faces.

PISO in icoFoam: Flowchart

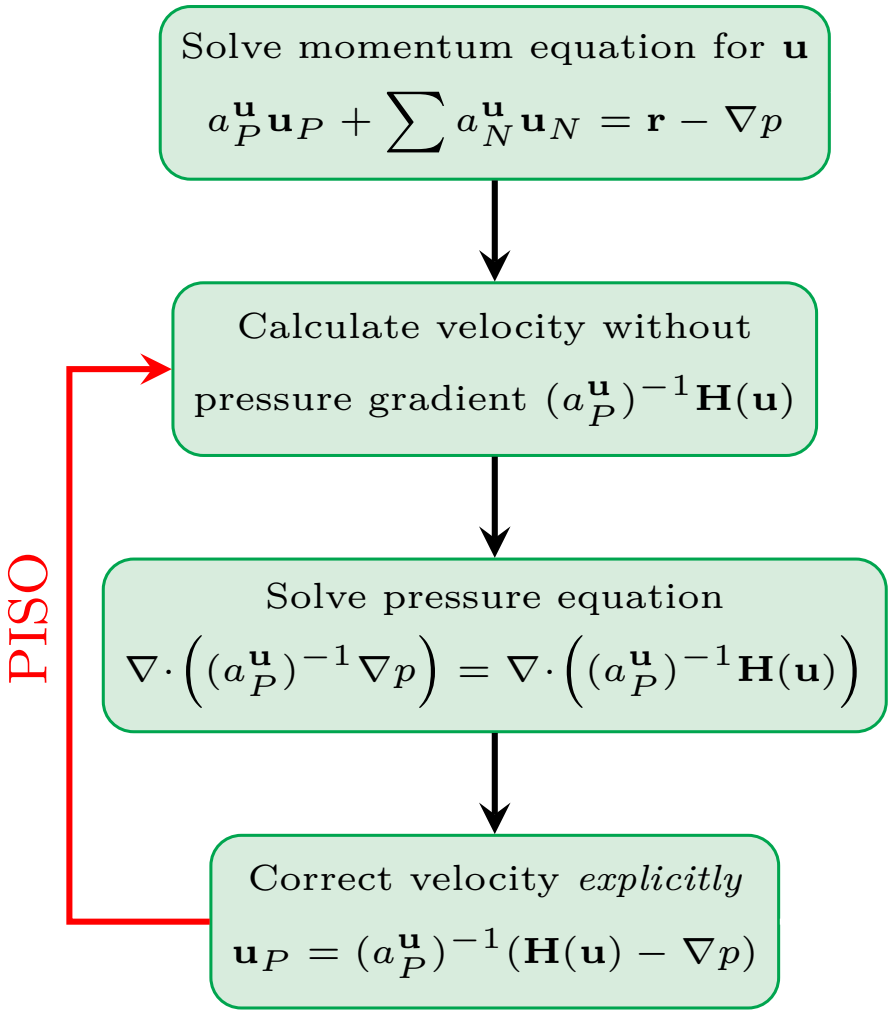
Flow chart of the main steps in both SIMPLE and PISO (in each time step)



Can we leave the current time step after the final correction and move on to the next time step?

PISO in icoFoam: Flowchart

Flow chart of the main steps in PISO (omitting details)



PISO in icoFoam: General algorithm (1/2)

(Issa, 1986; description from Versteeg and Malalasekera)

The original PISO algorithm consists of one predictor step and two corrector steps.
Here it is described with some details how it is done in OpenFOAM.

- Predictor step:
  - **Solve the discretized momentum equations using a guessed/intermediate pressure field to get intermediate velocity fields.**
    Remember: The face flux field `phi` fulfils continuity, but is frozen at the old time step.
- Corrector step 1:
  - The intermediate velocity fields will not fulfil continuity unless the guessed/intermediate pressure field used in the predictor step is correct. In OpenFOAM the pressure field should fulfil the pressure equation, derived using both the momentum and continuity equations. I.e.:, the first corrector step will:
    Solve the pressure equation and correct the velocities, including a consistent correction of the face flux field `phi`, which also represents the velocity field.
    Note: In the original algorithm there is a pressure *correction* equation (rather than a pressure equation), followed also by a pressure correction.

Continued ...

PISO in icoFoam: General algorithm (2/2)

(Issa, 1986; description from Versteeg and Malalasekera)

... continued

- Corrector step 2:
  - The first corrector step gives a new intermediate velocity field that satisfies continuity. However, the pressure equation was solved using the intermediate velocity field that did not fulfil continuity. The second corrector step should give the correct pressure field, based on a conservative velocity field, i.e.:
    Update and solve the discretized pressure equation again, using the original contributions from the discretized momentum equations, but using the updated velocities. Follow up with corrections of the velocity and face flux fields, as in the first corrector step.
    Note: The original PISO algorithm rather derives a second pressure correction equation, and also corrects the pressure a second time.

In the non-iterative PISO algorithm the velocity and pressure fields are considered solved after the two corrector steps (for sufficiently small time steps, i.e. tiny). In the iterative PISO algorithm the entire predictor-corrector procedure is repeated until convergence. This is necessary for large time steps (larger than tiny) or if the algorithm is used for steady-state.

PISO in icoFoam: Time loop

We will now have a look at how the PISO algorithm is implemented in `icoFoam.C`, looking only at the most important parts of the code. We will have a look at all of the code later.

A global perspective of the time loop, including only important parts, is given by (c.f. `pisoFoam.C`):

```
while (runTime.loop())
{
    {
        #include "UEqn.H"
        while (piso.correct())
        {
            #include "pEqn.H"
        }
    }
}
```

Here `UEqn.H` corresponds to the momentum predictor, and `pEqn.H` corresponds to the corrector step(s). The user can choose the number of corrector steps. In `cavity/system/fvSolution`:

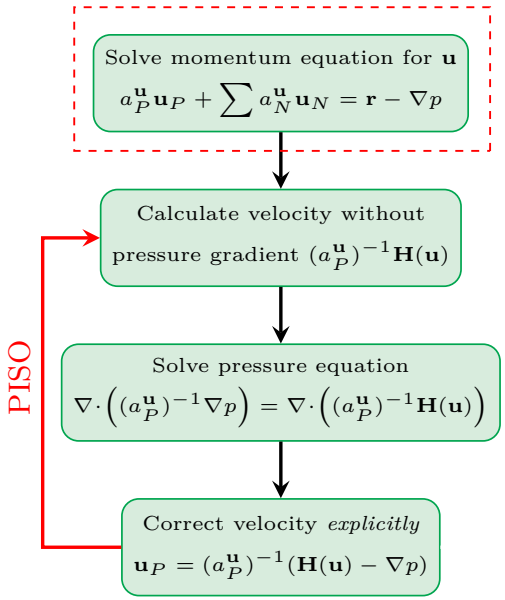
```
PISO { nCorrectors     2; }
```

This corresponds to the two corrector steps discussed before.

PISO in icoFoam: Momentum predictor

The momentum predictor step (`UEqn.H`) is implemented as:
(main difference in FOAM-extend is a 'consistent' treatment of the time term, see Tessa Uroic's PhD thesis)

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
);
if (piso.momentumPredictor())
{
    solve(UEqn == -fvc::grad(p));
}
```



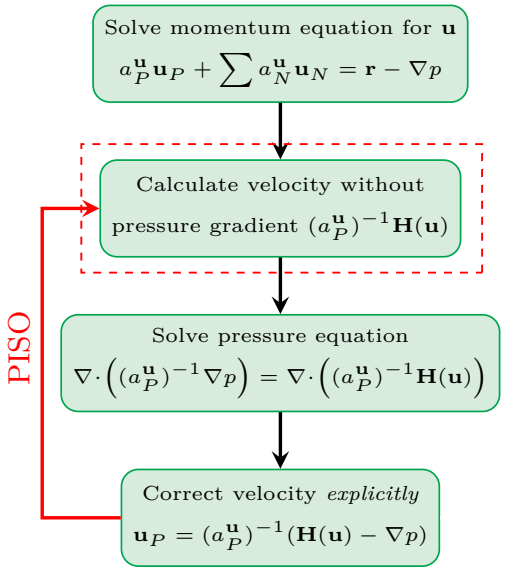
The convective term is linearized using face flux field `phi`, from previous time step.
The pressure gradient is excluded from `fvVectorMatrix UEqn`, since we later need to ask `UEqn` for the $\mathbf{H}(\mathbf{u})$ operator *without the pressure gradient*.
The user can choose if the momentum predictor should be done. It is done by default, but a switch can be added in `cavity/system/fvSolution`:

```
PISO { momentumPredictor    true; //false; //on; //off; }
```

PISO in icoFoam: Corrector step(s) (1/4)

The first part of the corrector step (`pEqn.H`) is implemented as:

```
while (piso.correct())
{
    volScalarField rAU(1.0/UEqn.A());
    volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
    surfaceScalarField phiHbyA
    (
        "phiHbyA",
        fvc::flux(HbyA)
      + fvc::interpolate(rAU)*fvc::ddtCorr(U, phi)
    );

    adjustPhi(phiHbyA, U, p);

    // Update the pressure BCs to ensure flux consistency
    constrainPressure(p, U, phiHbyA, rAU);

    //// CONTENTS IN NEXT SLIDE ////
}
```

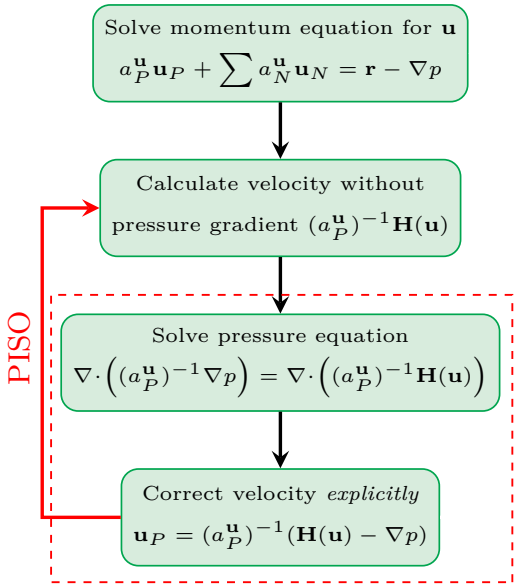


- Calculate velocity field without pressure gradient, `HbyA`, as $(a_P^{\mathbf{u}})^{-1}\mathbf{H}(\mathbf{u})$ with corrected bc's (`constrainHbyA`). The `UEqn` discretization is used (`UEqn.A()` and `UEqn.H()`).
- Calculate face flux of `HbyA` (with a time correction term which is discussed later).
- Enforce global conservation of `phiHbyA` and coherent pressure bc's.
- Continued ...

PISO in icoFoam: Corrector step(s) (2/4)

The second part of the corrector step (`pEqn.H`) is implemented as:

```
while (piso.correct())
{
    //// CONTENTS IN PREVIOUS SLIDE ////
    while (piso.correctNonOrthogonal())
    {
        fvScalarMatrix pEqn
        (
            fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
        );
        pEqn.setReference(pRefCell, pRefValue);
        pEqn.solve(mesh.solver(p.select(piso.finalInnerIter())));
        if (piso.finalNonOrthogonalIter())
        {
            phi = phiHbyA - pEqn.flux();
        }
    }
    #include "continuityErrs.H"
    U = HbyA - rAU*fvc::grad(p);
    U.correctBoundaryConditions();
}
```

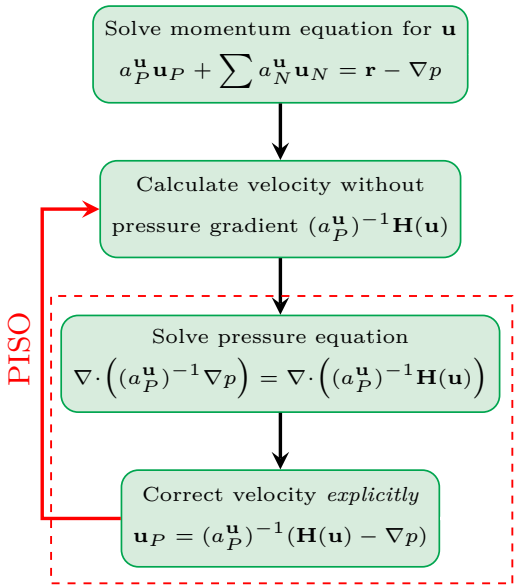


- Use the discretized mom. eq. and intermediate velocity field to solve the pressure equation: $\nabla \cdot \left[(a_P^{\mathbf{u}})^{-1}\nabla p\right] = \nabla \cdot \left[(a_P^{\mathbf{u}})^{-1}\mathbf{H}(\mathbf{u})\right]$
- Correct the face fluxes consistent with the discretized pressure equation.
- Correct the velocity field and make sure that the velocity bc's are still as set in the case.
- Continued ...

PISO in icoFoam: Corrector step(s) (3/4)

The second part of the corrector step (`pEqn.H`) is implemented as:

```
while (piso.correct())
{
    //// CONTENTS IN PREVIOUS SLIDE ////
    while (piso.correctNonOrthogonal())
    {
        fvScalarMatrix pEqn
        (
            fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
        );
        pEqn.setReference(pRefCell, pRefValue);
        pEqn.solve(mesh.solver(p.select(piso.finalInnerIter())));
        if (piso.finalNonOrthogonalIter())
        {
            phi = phiHbyA - pEqn.flux();
        }
    }
    #include "continuityErrs.H"
    U = HbyA - rAU*fvc::grad(p);
    U.correctBoundaryConditions();
}
```

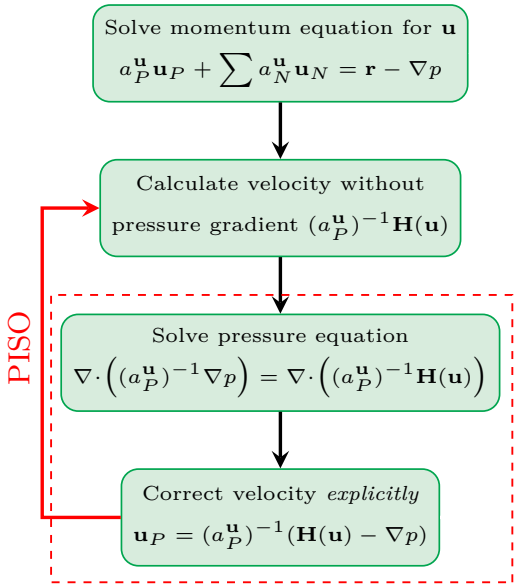


- The user can specify a number of non-orthogonal corrector steps in `cavity/system/fvSolution`:
  `PISO { nNonOrthogonalCorrectors 0; }`
  OpenFOAM has a few *explicit* implementations of discretization, such as for non-orthogonal correction and higher-order schemes. Iterations are needed to take those fully into account.
- Continued ...

PISO in icoFoam: Corrector step(s) (4/4)

The second part of the corrector step (`pEqn.H`) is implemented as:

```
while (piso.correct())
{
    //// CONTENTS IN PREVIOUS SLIDE ////
    while (piso.correctNonOrthogonal())
    {
        fvScalarMatrix pEqn
        (
            fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
        );
        pEqn.setReference(pRefCell, pRefValue);
        pEqn.solve(mesh.solver(p.select(piso.finalInnerIter())));
        if (piso.finalNonOrthogonalIter())
        {
            phi = phiHbyA - pEqn.flux();
        }
    }
    #include "continuityErrs.H"
    U = HbyA - rAU*fvc::grad(p);
    U.correctBoundaryConditions();
}
```



- Why is `phi` not corrected outside the while loop, instead of at the final loop inside the loop? Hint: Scope of variables.
- In the next slides the non-orthogonal correction loop and the consistent `phi` flux correction (`pEqn.flux()`) are explained.

PISO in icoFoam: Non-orthogonal correction loop

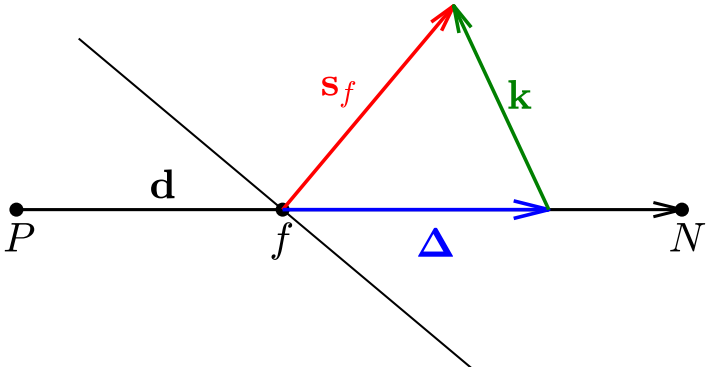
- The Laplacian term is discretized using the Gauss' theorem, as

$$\int_V \nabla \cdot (\Gamma \nabla \phi) \, \mathrm{d}V = \sum_f \Gamma_f \, \mathbf{s}_f \cdot (\nabla \phi)_f .$$

- The diffusion coefficient is always interpolated on the faces using the `linear` scheme. Therefore, it is only a matter of calculating surface normal gradients, $(\nabla \phi)_f$.

- surface normal gradient can be decomposed into *orthogonal* and *non-orthogonal* parts, as $\mathbf{s}_f = \mathbf{\Delta} + \mathbf{k}$:

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



- The orthogonal and non-orthogonal parts are treated implicitly and explicitly.
- OpenFOAM `laplacian` schemes contain a non-orthogonal correction term for calculating surface normal gradient schemes (unless the `uncorrected` scheme is employed).
- In each loop (`while (piso.correctNonOrthogonal())`) the explicit non-orthogonal correction term is updated. In other words, non-orthogonal correction loop only updates the left hand side of `pEqn`, while in the full pressure corrector loop (`while (piso.correct())`) both sides are updated.

PISO in icoFoam: Consistent conservative face fluxes

(Acknowledgements to Professor Hrvoje Jasak, and see Tessa Uroic's PhD thesis)

- Here we derive the consistent face flux calculated in OpenFOAM as:

```
phi = phiHbyA - pEqn.flux();
```

- We can discretize the continuity equation in a cell using Gauss theorem:

$$\nabla \cdot \mathbf{u} \; (= 0) = \sum_f \mathbf{s}_f \cdot \mathbf{u}_f = \sum_f F$$

where $\mathbf{u}_f$ is interpolated velocity to the faces and F is the face flux, $F = \mathbf{s}_f \cdot \mathbf{u}_f$. The face fluxes of the cell must preserve continuity!

- Substituting our previous expression for the velocity ($\mathbf{u}_P = (a_P^{\mathbf{u}})^{-1}(\mathbf{H}(\mathbf{u}) - \nabla p)$) and interpolating to the faces yields

$$F = \left((a_P^{\mathbf{u}})^{-1}\mathbf{H}(\mathbf{u})\right)_f \cdot \mathbf{s}_f - \left((a_P^{\mathbf{u}})^{-1}\nabla p\right)_f \cdot \mathbf{s}_f$$

- The first term on the R.H.S. is the face flux of `HbyA`, i.e. the already calculated `phiHbyA`, while the second term is computed by the `flux()` function.

- The second term on the R.H.S. appears during the discretization of the pressure Laplacian ($\nabla \cdot \left[(a_P^{\mathbf{u}})^{-1}\nabla p\right]$), for each face:

$$\left((a_P^{\mathbf{u}})^{-1}\nabla p\right)_f \cdot \mathbf{s}_f = \left((a_P^{\mathbf{u}})^{-1}\right)_f \frac{|\mathbf{\Delta}|}{|\mathbf{d}|}(p_N - p_P) + \left((a_P^{\mathbf{u}})^{-1}\right)_f \mathbf{k} \cdot (\nabla p)_f$$

PISO in icoFoam: Consistent conservative face fluxes

$$\left((a_P^{\mathbf{u}})^{-1}\nabla p\right)_f \cdot \mathbf{s}_f = \left((a_P^{\mathbf{u}})^{-1}\right)_f \frac{|\boldsymbol{\Delta}|}{|\mathbf{d}|}(p_N - p_P) + \left((a_P^{\mathbf{u}})^{-1}\right)_f \mathbf{k} \cdot (\nabla p)_f$$

- $|\mathbf{d}|$ is the distance between the owner and neighbour cell centers, and $a_N^P = (a_P^{\mathbf{u}})^{-1}\frac{|\boldsymbol{\Delta}|}{|\mathbf{d}|}$ is the off-diagonal matrix coefficient in the pressure Laplacian. We thus have to ask the discretized pressure equation for the term, i.e. `pEqn.flux()`, to correct the face flux field.
- The `flux()` function, defined in the `fvMatrix` class, calculate the first term through

```
lduMatrix::faceH(psi_.primitiveField().component(cmpt))
```

- which calls the `faceH` function of `lduMatrix` class:

```
for (label face=0; face<l.size(); face++)
{
    faceHpsi[face] =
        Upper[face]*psi[u[face]]
      - Lower[face]*psi[l[face]];
}
```

- The non-orthogonal correction term is also added through a pointer at the end of `flux()` function which points to the selected scheme.

```
if (faceFluxCorrectionPtr_)
{
    fieldFlux += *faceFluxCorrectionPtr_;
}
```

PISO in icoFoam: Rhie–Chow interpolation

(Versteeg & Malalasekera, Hrvoje Jasak, Tessa Uroic and Fabian Peng-Kärrholm)

- When using a colocated FVM formulation there may be unphysical pressure oscillations if the source terms of the momentum and pressure equations are determined using linear interpolation of the pressure (mom.eq.) and velocity (pres.eq.) to the faces.
- A remedy is to interpolate the velocity to the faces in a way that takes into account the pressure gradient *at* the faces, for the use in the pressure equation source term.
- According to Rhie–Chow correction, the interpolated face velocity should be corrected as

$$\mathbf{u}_f = \overline{\mathbf{u}}_f - \overline{\left((a_P^{\mathbf{u}})^{(-1)}\right)}_f \left(\nabla p_f - \overline{\nabla p}_f\right)$$

  Here, overline denotes linear interpolation of the underlying cell values at both sides of the face, and subscript f denotes face values. The correction is proportional to the difference between the pressure gradient at the face and the interpolated pressure gradient at the face.
- Although this correction is not explicitly implemented in OpenFOAM, here we show that the OpenFOAM procedure for pressure correction algorithm satisfies this interpolation type.

PISO in icoFoam: Rhie–Chow interpolation

- Now, let's recall our previous discretization of the momentum equation, we have

$$
\begin{aligned}
a_P^{\mathbf{u}}\mathbf{u}_P + \sum_N a_N^{\mathbf{u}}\mathbf{u}_N &= \mathbf{r} - \nabla p \\
\underbrace{\mathbf{r} - \sum_N a_N^{\mathbf{u}}\mathbf{u}_N}_{\mathbf{H}(\mathbf{u})} &= a_P^{\mathbf{u}}\mathbf{u}_P + \nabla p \\
(a_P^{\mathbf{u}})^{(-1)}\mathbf{H}(\mathbf{u}) &= \mathbf{u}_P + (a_P^{\mathbf{u}})^{(-1)}\nabla p
\end{aligned}
$$

- Applying the overline operator (linear interpolation on the faces) on both sides

$$
\overline{\left((a_P^{\mathbf{u}})^{(-1)}\mathbf{H}(\mathbf{u})\right)}_f = \overline{\mathbf{u}}_f + \overline{\left((a_P^{\mathbf{u}})^{(-1)}\right)}_f \overline{\nabla p}_f \tag{1}
$$

- Expanding The Rhie–Chow velocity correction

$$
\mathbf{u}_f = \overline{\mathbf{u}}_f - \overline{\left((a_P^{\mathbf{u}})^{(-1)}\right)}_f \nabla p_f + \overline{\left((a_P^{\mathbf{u}})^{(-1)}\right)}_f \overline{\nabla p}_f = \left(\overline{\mathbf{u}}_f + \overline{\left((a_P^{\mathbf{u}})^{(-1)}\right)}_f \overline{\nabla p}_f\right) - \overline{\left((a_P^{\mathbf{u}})^{(-1)}\right)}_f \nabla p_f \tag{2}
$$

- The first term on the right hand side of Eq. 2 is similar to the right hand side of Eq. 1.

PISO in icoFoam: Rhie–Chow interpolation

(Versteeg & Malalasekera, Hrvoje Jasak, Tessa Uroic and Fabian Peng-Kärrholm)

- Substituting Eq. 1 into Eq. 2 gives

$$\mathbf{u}_f = \overline{\left((a_P^{\mathbf{u}})^{(-1)}\mathbf{H}(\mathbf{u})\right)}_f - \overline{\left((a_P^{\mathbf{u}})^{(-1)}\right)}_f \nabla p_f$$

- Applying dot product with surface normal vector ($\cdot \mathbf{s}_f$)

$$\mathbf{u}_f \cdot \mathbf{s}_f = \overline{\left((a_P^{\mathbf{u}})^{(-1)}\mathbf{H}(\mathbf{u})\right)}_f \cdot \mathbf{s}_f - \overline{\left((a_P^{\mathbf{u}})^{(-1)}\right)}_f \nabla p_f \cdot \mathbf{s}_f$$

- This is exactly how the conservative fluxes are calculated, which is also equivalent to the pressure (continuity) equation we derived before.

```
phi = phiHbyA - pEqn.flux();
```

- In other words, Rhie–Chow interpolation is inherently included in OpenFOAM formulation by introducing $\mathbf{H}(\mathbf{u})$ operator and using its face interpolation.

PISO in icoFoam: Rhie–Chow interpolation

(Versteeg & Malalasekera, Hrvoje Jasak, Tessa Uroic and Fabian Peng-Kärrholm)

- In the explicit source term `fvc::div(phiHbyA)` of the pressure equation, `phiHbyA` does **not include any effect of the pressure**.
- `rAU` does **not include any effect of pressure** when solving the pressure equation and finally correcting the velocity.
- The Laplacian term, `fvm::laplacian(rAU, p)`, of the pressure equation uses the value of the gradient of p on the cell faces. The gradient is calculated using **neighbouring cells**, and not neighbouring faces.
- `fvc::grad(p)` (in mom.eq.) is calculated **from the cell face values** of the pressure.
- Can we turn off the Rhie–Chow interpolation in OpenFOAM?

Investigating the effect of Rhie–Chow interpolation

- Obviously we do not want to turn it off, but just as a practice to see its effects.
- Recall that Rhie–Chow correction states

  $$\mathbf{u}_f = \overline{\mathbf{u}}_f - \overline{\left((a_P^{\mathbf{u}})^{(-1)}\right)}_f \left(\nabla p_f - \overline{\nabla p}_f\right)$$

  which in OpenFOAM is equivalent to

  ```
  phi = phiHbyA - pEqn.flux();
  ```

- Thus calculating the face velocities just by interpolation will turn the correction off, $\mathbf{u}_f = \overline{\mathbf{u}}_f$, i.e.,

  ```
  phi = fvc::flux(U);
  ```

- The fluxes should be calculated after the explicit correction step of the velocity.

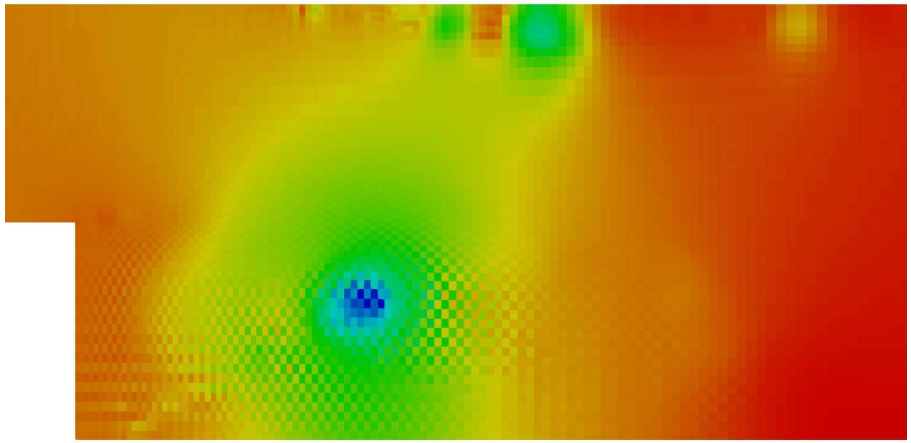
Investigating the effect of Rhie–Chow interpolation

- Turning off Rhie–Chow correction

```
while (piso.correctNonOrthogonal())
{
    ...

    pEqn.setReference(pRefCell, pRefValue);

    pEqn.solve(mesh.solver(p.select(piso.finalInnerIter())));

    //   if (piso.finalNonOrthogonalIter())
    //   {
    //       phi = phiHbyA - pEqn.flux();
    //   }
}

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

phi = fvc::flux(U);
#include "continuityErrs.H"
```

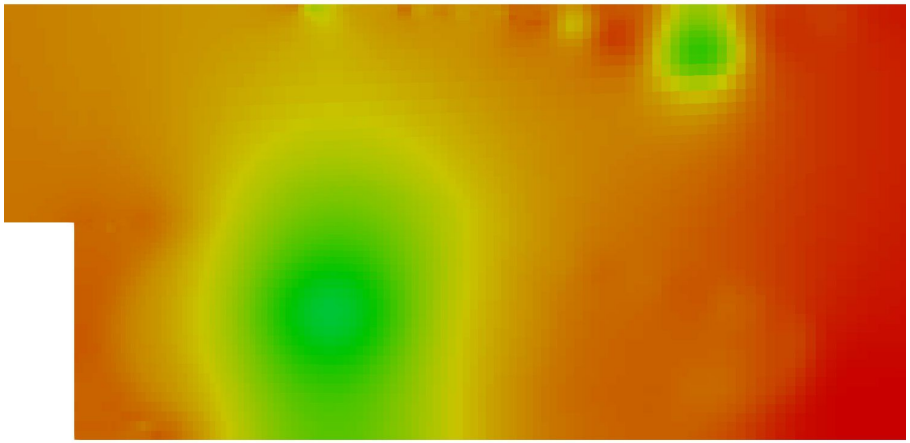
Investigating the effect of Rhie–Chow interpolation

- Running icoFoam on the pitzDaily case with and without Rhie–Chow correction
- The chequerboard oscillations are obviously seen when the correction is turned off.
- Cell value should be plotted as the point value interpolation removes the oscillations.
- Note that the continuity and momentum equations are satisfied for both cases.



Without Rhie–Chow correction

```
phi = fvc::flux(U);
```



With Rhie–Chow correction

```
phi = phiHbyA - pEqn.flux();
```