The PISO (Pressure-Implicit with Splitting of Operators) algorithm

We have the same setup (1), (2) of the SIMPLE algorithm
PISO's generally considered an extension of SIMPLE as it
is based on non-iterative computations of unsteady fluid flow
It involves one predictor step followed by 2 correction steps

We actually rewrite the decomp. in (6) as

$$\bar{\bar{M}}\,\bar{u}_h^{(k)} = \bar{\bar{D}}\,\bar{u}_h^{(k)} + \bar{\bar{\mathcal{H}}} \implies \bar{u}_h^{(k)} = \bar{\bar{D}}^{-1}\bar{\bar{\mathcal{H}}} - \bar{\nabla} p_h^{(k)} \quad (7)$$

where $\bar{\bar{\mathcal{H}}}$ incorporates $\bar{u}_h^{(k-1)}$.
As a result the pressure equation becomes:

$$\bar{\nabla}\cdot\bar{u}_h^{(k)} = \bar{\nabla}\cdot\left(\bar{\bar{D}}^{-1}\bar{\bar{\mathcal{H}}} - \bar{\nabla} p_h^{(k)}\right) = 0$$

$$\Downarrow$$

$$\Delta p_h^{(k)} = \bar{\nabla}\cdot\left(\bar{\bar{D}}^{-1}\bar{\bar{\mathcal{H}}}\right) \quad (8)$$

Matrix $\bar{\bar{\mathcal{H}}}$ will be refered as the residual since

$$\bar{\bar{\mathcal{H}}} = \bar{\bar{M}}\,\bar{u}_h^{(k)} - \bar{\bar{D}}\,\bar{u}_h^{(k)} = (\bar{\bar{M}} - \bar{\bar{D}})\,\bar{u}_h^{(k)} \quad (9)$$

and, as in (8), it can be interpreted as a source term for pressure.

Therefore one can proceede to compute prediction $\overset{*}{\bar{u}}{}_h^{(k)}$ in (7), use it to derive (9) and $p_h^{(k)}$ from (8) and then use both $\bar{\bar{\mathcal{H}}}$ and $p_h^{(k)}$ to compute the updated, divergence-free, velocity field $\bar{u}_h^{(k)}$.

Once the velocity field gets updated also $\bar{\bar{\mathcal{H}}}$ would change and thus also $p_h^{(k)}$; therefore in the PISO algorithm, instead of closing the loop at calculation of $p_h^{(k+1)}$, as in SIMPLE,

we use the flux-corrected $U_h^{(k)}$ to update $\bar{\bar{\mathcal{H}}}$ directly.
This essentially translated of having two nested corrector loops:
: the outer loop is the standard pressure correction which, as in SIMPLE, directly follows the momentum predictor; the inner loop updates the pressure until (8) converges

PISO PSEUDOCODE

```
set BCs and K=0
guess p^(0)
/*outer*/ while toll < 1e-10
            compute H from (9)
            solve (7) to obtain U*_h^(K)
            solve (8) to obtain p*_h^(K)
            /*inner*/ while toll < 1e-04
                        update U*_h^(K) using p*_h^(K) in (7)
                        derive updated H* from (9)
                        update p*_h^(K) using H* in (8)
            end
end
```

The adjustment of iterating in a smaller loop rather than updating the whole system every time has the advantage that, if the time step is small enough ($Co < 1$), no under-relaxation is needed for the linear system to converge.

PISO IN OF

```
fvVectorMatrix UEqn
(
    fvm::ddt(U) + fvm::div(phi, U) - fvm::laplacian(mu, U)
);
solve (UEqn == - fvc::grad(p));
volScalarField D = UEqn().A();
U = 1.0/D * UEqn().H();
phi = (fvc::interpolate(U) & mesh.Sf()) +
        + fvc::ddtPhiCorr(1.0/D, U, phi);
adjustPhi(phi, U, p);

fvScalarMatrix pEqn
(
 fvm::laplacian(1.0/D, p) == fvc::div(phi);
)

pEqn.setReference(pRefCell, pRefValue);
pEqn.solve();
if (nonOrth == nNonOrthCorr)
{
    phi = - pEqn.flux();
}

U -= 1.0/D * fvc::grad(p);
U.correctBoundaryConditions();
```