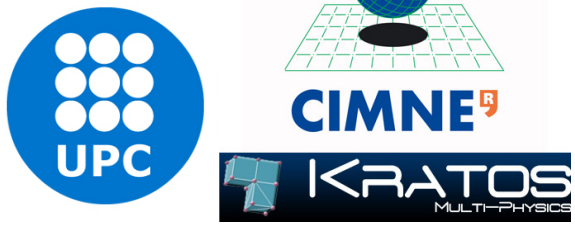
The Finite Element Method for Fluid-Structure Interaction with open source software - Solution Procedures

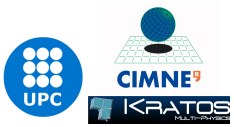
Riccardo Rossi Pooyan Dadvand

Kratos Team

Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE)



July, 2014, TU Munich



Outline

Once again the monolithic problem

Once discretized (and stabilized), the NS equations give rise to a linear system of equations of the type

$$\begin{pmatrix} \mathbf{K} & \mathbf{G} \\ \mathbf{D} & \mathbf{S} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \tag{1}$$

Where we are abusing of the symbols $\mathbf{K}, \mathbf{G}, \mathbf{D}, \mathbf{S}$ by assuming that they include contributions stemming out of the stabilization terms.
Formulations that arrive to define a problem which solves "at once" for velocities and pressure are denominated ''monolithic".

Once again the monolithic problem

As a matter of fact, the term at position 11 is typically in the form $\mathbf{K} \approx \frac{c\mathbf{M}}{\Delta t} + \mathbf{C}(\mathbf{u}) + \mathbf{L}_\nu + \mathbf{S}_\mathbf{u}$ and is constituted by:

1. an inertia term $\frac{c\mathbf{M}}{\Delta t}$ which is symmetric and well behaved
2. a dissipative term $\mathbf{L}_\nu$ which has essentially the structure of a laplacian and is symmetric and positive definite
3. a non-linear convective term $C(\mathbf{u})$ which is non symmetric
4. a stabilizing term which essentially adds directional dissipation

Once again the monolithic problem

The K term is relatively well behaved when dissipation dominates, which is normally the case of flows at low Re. Unfortunately for the flows of typical interest the convective part is much more important and hence the matrix tends to become skew-symmetric.
The inertia term on the other hand is very well behaved (it is typical to use a diagonal approximation for it) and its importance can be adjusted by varying δt. For sufficiently low values of the time step it dominates over all the other terms in the momentum equation. This fact ultimately justifies explicit time integration schemes for the momentum equation.

remark

In order to guarantee stability as $\Delta t \rightarrow 0$ the stabilization terms **do not** disappear for small time steps. The overall incompressible system CAN NOT be solved explicitly

Some properties of matrix K

Let's now focus for a second on the block K. As we observed before it is obtained by assembling various contributions. Interestingly each of the contributions have some properties in the discrete:

1. dissipative term $\mathbf{L}_\nu$is such that $\sum_J \mathbf{L}_{\nu I} J = 0$
2. convective term $C(\mathbf{u})$ is also such that $\sum_J \mathbf{C}(\mathbf{u})_I J = 0$

One could convince of this by considering that there is no dissipation and no convective effect if we apply a constant velocity to all of the nodes. This very same argument also holds for the stabilization part (apart for its time dependent part)
This arguments intrinsically imply that the *lumped* inertia contribution can be obtained by summing up the terms in the rows, that it
$\frac{c\mathbf{M}_{II}}{\Delta t} = \sum_J K_{IJ}$
which is a property that may have practical applications

A linear algebra point of view

From the point of view of the linear algebra the monolithic problem is UGLY. The system matrix is:

1. non-symmetric
2. non positive-definite
3. blocks have different scaling since they are dimensionally not consistent

exactly all the features that iterative solvers do not like.

Suitable methods for the solution

Since the Monolithic problem naturally rises from the discretization of the Navier-Stokes equations, a huge amount of research has been spent over the years to find optimal methods for the solution of such system.
In any case basically all of the techniques fit in one of the following cathegories

1. Direct methods
2. Iterative methods $\longrightarrow$ research is really concentrated on preconditioners
3. Multigrid?

Direct Methods

Direct Methods represent the solution of choice for small problems. They are by far the most robust techniques as they can solve basically any invertible problems, (almost) independently on the conditioning.
The basic idea is to compute the LU decomposition of the system.
Their fundamental problem is related to

1. Algorithmic Complexity above $O(N^2)$ even taking into account the sparsity of the FE matrices
2. Memory cost which becomes prohibitive for large systems due to the storage needs of the L and U factors.

Parallelization is possible but very hard. Implementation of sparse direct solvers is very very technical ... do not try to implement one on your own!!!!

Direct Methods

A number of alternatives are available on the internet.

1. SuperLU it is a freely available (and nicely licensed) library which comes with a Direct solver as well as with preconditioners. It has a distributed memory version (SuperLUdist) and a Multithreaded one (SuperLU MT). I was never able to use the MT one... also works in Windows
2. ParDiSo (Parallel Direct Solver). As the name says it is parallel and from my own experience it is the fastest solver i found. The Intel MKL provides a license of it. Non-commercial licenses are available in linux.
3. MUMPS (Multifrontal Massively Parallel Solver) ...no experience but heard good about it
4. Pastix . Also a parallel multi-frontal solver with a nice interface and a group behind. Very promising but could not try it out yet

Iterative Solvers

The characteristics of the linear problem limit the possible choice of the iterative solver to basically two possibilities

1. BiCGStab (BiConjugate Gradient Stabilized) - possibly BiCGStab(l) versions could be appealing
2. GMRES (Generalized Minimum Residual)

The **GMRES** method guarantees achievement of the solution (in exact arithmetics) after N iterations, with N being the number of dofs in the system. This guarantee is not really useful since to do this N*N storage would be needed as well as a prohibitive computational cost. Nevertheless GMRES is the krylov method known that requires less iterations to converge.
The **BICGStab** method does not have a firm convergence proof ... however it does work. It needs more iterations (possibly even many more) than GMRES but each iteration costs much less than GMRES, particularily if many iterations are needed.

GMRES algorithm

ALGORITHM 6.9: GMRES

1. Compute $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, and $v_1 := r_0/\beta$
2. Define the $(m+1) \times m$ matrix $\bar{H}_m = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$. Set $\bar{H}_m = 0$.
3. For $j = 1, 2, \ldots, m$ Do:
4. Compute $w_j := Av_j$
5. For $i = 1, \ldots, j$ Do:
6. $h_{ij} := (w_j, v_i)$
7. $w_j := w_j - h_{ij} v_i$
8. EndDo
9. $h_{j+1,j} = \|w_j\|_2$. If $h_{j+1,j} = 0$ set $m := j$ and go to 12
10. $v_{j+1} = w_j / h_{j+1,j}$
11. EndDo
12. Compute y_m the minimizer of $\|\beta e_1 - \bar{H}_m y\|_2$ and $x_m = x_0 + V_m y_m$.

Preconditioning

While the choice of the krylov solver to use is somewhat reduced, a whole world of possibilites rises for "preconditioning strategies".
The idea of preconditioning is that instead of solving a system of the type

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2}$$

it may be easier to solve

$$\mathbf{M}\mathbf{A}\mathbf{x} = \mathbf{M}\mathbf{b} \tag{3}$$

provided that $\mathbf{M} \approx \mathbf{A}^{-1}$
or even

$$\mathbf{M}_1\mathbf{A}\mathbf{M}_2\mathbf{M}_2^{-1}\mathbf{x} = \mathbf{M}_1\mathbf{b} \tag{4}$$

ILU type preconditioners

ILU type preconditioners have their origin in the idea of LU decomposition, and hence have similarities with sparse direct solver technology. The basic idea is that instead of performing a complete factorization of the system matrix, an *Incomplete Factorization* is performed, by dropping terms which are "less important" .
A key issue is how to control effectively this dropping so to get to a method that is both robust and fast.
Many different flavors of ILU preconditioning exist, but still no clear winner exists. As a matter of fact, although some concepts taken of Multigrid are now incorporated in modern multilevel solvers, ILU type preconditioners are normally not very scalable.
For use on small computers or with OpenMP they may however be very effective.
They have some potential as **black box solvers**

ILU type preconditioners

Many libraries of ILU preconditioners are available in the literature. Some are MPI parallel. Very few are OpenMP parallel.

- SuperLU - starting from version 4.0 it provides a ILU preconditioner. NOT PARALLEL
- ITSOL - a library by Yousef Saad which implements a number of ILU flavors. NOT PARALLEL
- IFPACK - a library of preconditioners implemented in Trilinos - MPI PARALLEL
- PASTIX - implementes an ilu preconditioner - MPI PARALLEL but also OPENMP PARALLEL (may be very interesting for OpenMP!!)

Block preconditioners

The ILU preconditioners described above DO NOT take advantage of the structure of the NS matrix nor of its natural division in blocks.
A wide class of preconditioners, which is quite in fashion, starts exactly from this consideration and divides the monolithic matrix in the 4 blocks of which it is constituted. This knowledge is then used to perform a symbolic LU decomposition.
If we start with the system

$$\begin{pmatrix} \mathbf{K} & \mathbf{G} \\ \mathbf{D} & \mathbf{S} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \tag{5}$$

we can solve symbolically for $\mathbf{u}$ from the first equation to get

$$\mathbf{u} = \mathbf{K}^{-}1\mathbf{r}_u - \mathbf{K}^{-1}\mathbf{G}\mathbf{p} \tag{6}$$

by substituting in the second line we obtain

$$\left(\mathbf{S} - \mathbf{D}\mathbf{K}^{-1}\mathbf{G}\right)\mathbf{p} = \mathbf{r}_p - \mathbf{D}\mathbf{K}^{-1}\mathbf{r}_u \tag{7}$$

It is clear that if we were able to solve exactly the pressure schur complement in 7 and then substitute it into 6 we could obtain the exact solution of the original system

Block preconditioners

interestingly the statement expressed just now could be expressed as the successive solution of two systems:

$$\begin{pmatrix} \mathbf{K} & \mathbf{0} \\ \mathbf{D} & \mathbf{S} - \mathbf{D}\mathbf{K}^{-1}\mathbf{G} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{p}} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \tag{8}$$

followed by

$$\begin{pmatrix} \mathbf{I} & \mathbf{K}^{-1}\mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{p}} \end{pmatrix} \tag{9}$$

if we group the two we get

$$\begin{pmatrix} \mathbf{K} & \mathbf{0} \\ \mathbf{D} & \mathbf{S} - \mathbf{D}\mathbf{K}^{-1}\mathbf{G} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{K}^{-1}\mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \tag{10}$$

which shows how we decomposed the original system in a (block) lower triangular one and an upper triangular one

Block preconditioners

remark

The decomposition shown is **exact**. If we could solve the system exactly we would get convergence in one iteration

the problem

Unfortunately solving the modified block LU problem is NOT easier than solving the original one. The problem roots in the computation of $\mathbf{K}^{-1}$. Note that for solving the block $\mathbf{S} - \mathbf{D}\mathbf{K}^{-1}\mathbf{G}$ a matrix-free method could be applied, but the cost would be unaffordable.

the solution

The solution is to *approximate* the term $\mathbf{S} - \mathbf{D}\mathbf{K}^{-1}\mathbf{G}$. A lot of research is currently being done on how to find an acceptable approximation

SIMPLE algorithm

We begin by observing that SIMPLE stands for "Semi Implicit Method for Pressure-Linked Equations".
In the "simple" family of methods $\mathbf{K}^{-1}$ is approximated by a diagonal approximation, that is $\mathbf{K}^{-1} \approx (diag\,(\mathbf{K}))^{-1}$
the diagonalizing operator can be defined in different ways for example

- $(diag\,(\mathbf{K}))_{II} := \mathbf{K}_{II} \longrightarrow$ SIMPLE
- $(diag\,(\mathbf{K}))_{II} := \sum_J \mathbf{K}_{IJ}$
- $(diag\,(\mathbf{K}))_{II} := \sum_J |\mathbf{K}_{IJ}| \longrightarrow$ SIMPLEC
- $(diag\,(\mathbf{K}))_{II} := \sqrt{\sum_j \mathbf{K}_{IJ}^2}$

At the limit for $\Delta t \to 0$ the inertia term dominates over the others, hence all choices tend to coincide, and the diagonal represents an excellent approximation of $\mathbf{K}^{-1}$.
Independently of the diagonalization technique chosen, the algorithm will finally take a form of the type

1. solve $\mathbf{K}\hat{\mathbf{u}} = \mathbf{r}_u$ for $\hat{\mathbf{u}}$
2. solve $\left(\mathbf{S} - \mathbf{D}\,(diag\,(\mathbf{K}))^{-1}\,\mathbf{G}\right)\mathbf{p} = \mathbf{r}_p - \mathbf{D}\hat{\mathbf{u}}$ for pressure
3. solve $\mathbf{u} = \hat{\mathbf{u}} - (diag\,(\mathbf{K}))^{-1}\,\mathbf{G}\mathbf{p}$

remark

The SIMPLE algorithm can be used alone iterating the steps above until convergence or used as a preconditioner for example for a GMRES iteration

error of SIMPLE procedure

Let's now make an attempt to evaluate the error of the simple procedure. The idea is that at each SIMPLE step, we are *exactly* solving the problem

$$\mathbf{A}_{SIMPLE} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_v \\ \mathbf{r}_p \end{pmatrix} \tag{11}$$

with

$$\mathbf{A}_{SIMPLE} := \begin{pmatrix} \mathbf{K} & \mathbf{0} \\ \mathbf{D} & \mathbf{S} - \mathbf{D} diag\left(\mathbf{K}\right)^{-1} \mathbf{G} \end{pmatrix} \begin{pmatrix} \mathbf{I} & diag\left(\mathbf{K}\right)^{-1} \mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} = \begin{pmatrix} \mathbf{K} & \mathbf{K} diag\left(\mathbf{K}\right)^{-1} \mathbf{G} \\ \mathbf{D} & \mathbf{S} \end{pmatrix} \tag{12}$$

instead of the original system

$$\mathbf{A} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_v \\ \mathbf{r}_p \end{pmatrix} \tag{13}$$

with

$$\mathbf{A} := \begin{pmatrix} \mathbf{K} & \mathbf{G} \\ \mathbf{D} & \mathbf{S} \end{pmatrix} \tag{14}$$

the error in such approximation can be assessed as $\mathbf{E} := \mathbf{A} - \mathbf{A}_{SIMPLE}$ to give

$$\mathbf{E} := \begin{pmatrix} \mathbf{0} & \mathbf{G} - \mathbf{K} diag\left(\mathbf{K}\right)^{-1} \mathbf{G} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{15}$$

error of SIMPLE procedure

remark

If we accept that for $\Delta t \to 0$ than $\mathbf{K} diag\,()^{-1} \to \mathbf{I}$, which is definitely true if we use lumped mass matrix, than this allows to conclude that the SIMPLE preconditioner tends to the ideal one as the time step tends to zero

remark

UNFORTUNATELY for practical application inertia terms DO NOT dominate everywhere. For the range of time steps we would like to use in the practice the time step may be small in some parts of the domain but large in others, for example in the Boundary Layer.
The SIMPLE preconditioner IS NOT optimal for such cases.

In the practice it is accepted that at the opposite limit, when the viscous forces dominate over inertia (that is in the STOKES case) the ideal preconditioner is a different one:

$$\mathbf{D}\mathbf{K}^{-1}\mathbf{G} \to \nu \mathbf{M}_p \tag{16}$$

where the new symbol $\mathbf{M}_p$ is the "pressure mass matrix" $\mathbf{M_p}_{IJ} := \int_\Omega N_I N_J d\Omega$

other block preconditioners

A wide number of variations of the SIMPLE scheme were proposed in the literature with the aim of obtaining a convergence which is

- *scalable* meaning that the number of iterations is not (too) affected by the mesh size
- *independent on the Reynolds number* ... meaning in fact that do not depend too much on the time step (most of the arguments used in the development of such methods are devoted to the stationary case)

While i have no intention to provide a complete list here, one first possibility is to modify the SIMPLE scheme so to make it "doubly asympdotic" that is "optimal" both at the Stokes limit and at the small-time-step limit. One possibility is to solve:

1. solve $\mathbf{K}\hat{\mathbf{u}} = \mathbf{r}_u$ for $\hat{\mathbf{u}}$
2. solve $\left(\mathbf{S} - \mathbf{D}\left(diag\left(\mathbf{K}\right)\right)^{-1}\mathbf{G}\right)\mathbf{p} = \mathbf{r}_p - \mathbf{D}\hat{\mathbf{u}}$ for pressure
3. solve $\nu\mathbf{M}_p\delta\mathbf{p} = \mathbf{r}_p - \mathbf{D}\hat{\mathbf{u}}$ for a pressure correction $\delta\mathbf{p}$
4. correct the pressure as $\mathbf{p} = \mathbf{p} + \delta\mathbf{p}$
5. solve $\mathbf{u} = \hat{\mathbf{u}} - \left(diag\left(\mathbf{K}\right)\right)^{-1}\mathbf{G}\mathbf{p}$

MSIMPLER approach

The MSIMPLER approach was proposed by Vuik and others and reported to have Re independent behaviour and to be fairly independent on the mesh size

1. solve $\left(\mathbf{S} - \mathbf{D}\mathbf{M}_\mathbf{u}^{-1}\mathbf{G}\right)\mathbf{p}^* = \mathbf{r}_p - \mathbf{D}\mathbf{M}_\mathbf{u}^{-1}\mathbf{r}_\mathbf{u}$ for $\mathbf{p}^*$
2. solve $\mathbf{K}_{static}\hat{\mathbf{u}} = \mathbf{r}_u - \mathbf{G}\mathbf{p}^*$ for $\hat{\mathbf{u}}$
3. solve $\left(\mathbf{S} - \mathbf{D}\mathbf{M}_\mathbf{u}^{-1}\mathbf{G}\right)\delta\mathbf{p} = \mathbf{r}_p - \mathbf{D}\hat{\mathbf{u}}$ for $\delta\mathbf{p}$
4. update $\mathbf{u} = \mathbf{u}^* - \mathbf{M}_\mathbf{u}^{-1}\mathbf{G}\delta\mathbf{p}$
5. update $\mathbf{p} = \mathbf{p}^* + \delta\mathbf{p}$

Without entering in the discussion of the derivation of this preconditioner, we would like to highlight that it has some interesting features:

- velocity mass matrix $\mathbf{M}_\mathbf{u}$ is required
- The "static part" of the "**uu**" term is needed.

Fractional Step

The idea of the "fractional-step" (FS) algorithm is very close to the idea of the SIMPLE algorithm.
The classical presentation of the FS is as follows (considering here Backward Euler for simplicity):

$$\mathbf{M_u}\frac{\mathbf{u}_{n+1}-\mathbf{u}_n}{\Delta t}+\mathbf{C}\left(\mathbf{u}\right)\mathbf{u}+\mathbf{L}_\nu+\mathbf{G}\mathbf{p}_{n+1}=\mathbf{f} \tag{17}$$

$$D\mathbf{u}=\mathbf{0} \tag{18}$$

the first equation can be written symbolically as

$$\mathbf{M_u}\frac{\mathbf{u}_{n+1}-\hat{\mathbf{u}}}{\Delta t}+\mathbf{M}\frac{\hat{\mathbf{u}}-\mathbf{u}_n}{\Delta t}+\mathbf{C}\left(\mathbf{u}n+1\right)\mathbf{u}_{n+1}+\mathbf{L}_\nu+\mathbf{G}\mathbf{p}_n+\mathbf{G}\left(\mathbf{p}_{n+1}-\mathbf{p}_n\right)=\mathbf{f} \tag{19}$$

if we now accept that $\mathbf{C}\left(\mathbf{u}n+1\right)\mathbf{u}_{n+1}=\mathbf{C}\left(\hat{\mathbf{u}}\right)\hat{\mathbf{u}}$ we can split the equation above in two to obtain

$$\mathbf{M_u}\frac{\hat{\mathbf{u}}-\mathbf{u}_n}{\Delta t}+\mathbf{C}\left(\hat{\mathbf{u}}\right)\hat{\mathbf{u}}+\mathbf{L}_\nu+\mathbf{G}\mathbf{p}_n=\mathbf{f} \tag{20}$$

$$\mathbf{M_u}\frac{\mathbf{u}_{n+1}-\hat{\mathbf{u}}}{\Delta t}\mathbf{G}\left(\mathbf{p}_{n+1}-\mathbf{p}_n\right)=\mathbf{0} \tag{21}$$

Fractional Step

to follow we can now obtain $\mathbf{u}_{n+1}$ from the last equation

$$\mathbf{u}_{n+1} = \hat{\mathbf{u}} - \Delta t \mathbf{M}_{\mathbf{u}}^{-1} \mathbf{G} \left(\mathbf{p}_{n+1} - \mathbf{p}_n \right) \tag{22}$$

and substitute it into the divergence constraint $\mathbf{D}\mathbf{u}_{n+1} = \mathbf{0}$ to get

$$\mathbf{D}\mathbf{u}_{n+1} = \mathbf{0} = \mathbf{D}\hat{\mathbf{u}} - \Delta t \mathbf{D}\mathbf{M}_{\mathbf{u}}^{-1} \mathbf{G} \left(\mathbf{p}_{n+1} - \mathbf{p}_n \right) \tag{23}$$

up to this point the FS appears to be basically coincident with the SIMPLE. However we can now approximate the matrix that multiplies the pressure as $\mathbf{L} \approx \mathbf{D}\mathbf{M}_{\mathbf{u}}^{-1}\mathbf{G}$
an heuristic argument may help to understand this approximation: a term $\pi = \mathbf{M}_{\mathbf{u}}^{-1}\mathbf{G}\mathbf{p}$ can be seen as the *nodal gradient of the pressure*. Hence $\mathbf{D}\pi$ is the divergence of a gradient which in turn is a laplacian. For this reason the term $\mathbf{D}\mathbf{M}_{\mathbf{u}}^{-1}\mathbf{G}$ is known in the literature as "discrete laplacian"

Fractional Step

The final form of the fractional step is the following:

- Solve until convergence the **non-linear** problem

$$\mathbf{M_u}\frac{\hat{\mathbf{u}}-\mathbf{u}_n}{\Delta t}+\mathbf{C}\left(\hat{\mathbf{u}}\right)\hat{\mathbf{u}}+\mathbf{L}_\nu+\mathbf{G}\mathbf{p}_n=\mathbf{f} \tag{24}$$

- Solve for pressure the problem

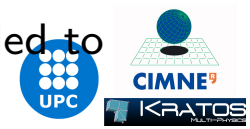
$$\Delta t\mathbf{L}\left(\mathbf{p}_{n+1}-\mathbf{p}_n\right)=\mathbf{D}\hat{\mathbf{u}} \tag{25}$$

- compute the end of step velocity as

$$\mathbf{u}_{n+1}=\hat{\mathbf{u}}-\Delta t\mathbf{M}_\mathbf{u}^{-1}\mathbf{G}\left(\mathbf{p}_{n+1}-\mathbf{p}_n\right) \tag{26}$$

- go to the next time step!! NOT a preconditioner

It can be shown that, if the skew-symmetric form of the convection term is used, the fractional step algorithm described is **unconditionally stable**. If BDF2 is used instead of BE second order accuracy in velocity is possible since the error in pressure is of the second order
The use of a continuum laplacian instead of the discrete laplacian provides some degree of stability for *sufficiently high values of* Δt. Pressure stabilization is needed to be able to use small time steps (which we need for accuracy).
similiarily convection should be stabilized.



remarks on Fractional Step

remark 1

A first remark is that the inverse of the velocity mass matrix $\mathbf{M_u^{-1}}$ should take into account velocity boundary conditions. That is, it should be zero on all of the nodes where velocity is imposed.

remark 2

The continuum laplacian $\mathbf{L}$ is not invertible per-se but needs pressure to be fixed on the neumann boundary. Furthermore solving a laplacian instead of a discrete laplacian implies imposing the spurious boundary condition $\frac{\partial \mathbf{p}_{n+1} - \mathbf{p}_n}{\partial \mathbf{n}} = 0 \Rightarrow \frac{\partial \mathbf{p}_{n+1}}{\partial \mathbf{n}} = \frac{\partial \mathbf{p}_n}{\partial \mathbf{n}}$ on all of the walls were the velocity is imposed.

Solving the pressure laplacian

A common feature of most of the methods found until now is that a laplacian-like equation needs to be solved for the pressure.
Provided that the system is symmetric and positive definite (SPD) the iterative methods of choice are

- Conjugate Gradient method (CG)
- Multigrid, either geometric (MG) or algebraic (AMG)

Multigrid approaches have optimal complexity and necessarily win for large problems. Nevertheless they tend to have high setup times and hence CG may win for small and middle-sized problems.
The possibility of using "deflation" may give some of the advantages of multigrid to the CG, leading to a scheme that is very simple to implement.
let's have a look...

Standard CG method

Let's consider a linear SPD system of the form $\mathbf{Ax} = \mathbf{b}$ which can be written in residual form as $\mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k$
the basic idea of the CG is to look for a correction $\mathbf{p}_k$ defined by a recurrence formula of the type

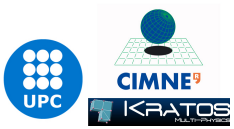
$$\mathbf{p}_k = \alpha_k \left(\mathbf{r}_k + \beta_{k-1} \mathbf{p}_{k-1} \right) = \alpha_k \mathbf{v}_k \tag{27}$$

and impose at each iteration that

$$\mathbf{p}_k \cdot \mathbf{Ax}_{k-1} = 0 \implies \beta_{k-1} = - \frac{\mathbf{r}_{k-1} \cdot \mathbf{Ap}_{k-1}}{\mathbf{p}_{k-1} \mathbf{Ap}_{k-1}} \tag{28}$$

and, by some manipulation considering orthogonality $\beta_j := \frac{\mathbf{r}_{j+1} \cdot \mathbf{r}_{j+1}}{\mathbf{r}_j \cdot \mathbf{r}_j}$. The second relation we need to impose is that

$$\mathbf{p}_k \left(\mathbf{b} - \mathbf{A} \left(\mathbf{x}_{k-1} + \mathbf{p}_k \right) \right) = \mathbf{0} \implies \alpha_k = \frac{\mathbf{v}_k \cdot \mathbf{r}_k}{\mathbf{v}_k \cdot \mathbf{Av}_k} \tag{29}$$



Standard CG algorithm

taking into account the "derivation" in the last slide, the CG algorithm assumes the form

1. Compute $\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$; $\mathbf{p}_0 := \mathbf{r}_0$
2. for j=0 ... until convergence
3. $\alpha_j := \frac{\mathbf{r}_j \cdot \mathbf{r}_j}{\mathbf{p}_j \cdot \mathbf{A}\mathbf{p}_j}$
4. $\mathbf{x}_{j+1} := \mathbf{x}_j + \alpha_j \mathbf{p}_j$
5. $\mathbf{r}_{j+1} := \mathbf{r}_j - \alpha_j \mathbf{A}\mathbf{p}_j$
6. $\beta_j := \frac{\mathbf{r}_{j+1} \cdot \mathbf{r}_{j+1}}{\mathbf{r}_j \cdot \mathbf{r}_j}$
7. $\mathbf{p}_{j+1} := \mathbf{r}_{j+1} + \beta_j \mathbf{p}_j$

Deflated CG method

The deflated version of the CG modifies the search direction of the CG by adding a number of basis vectors **W**

$$\mathbf{p}_k = \alpha_k \left(\mathbf{r}_k + \beta_{k-1} \mathbf{p}_{k-1} - \sum_m \mathbf{W}_m \lambda_m \right) = \alpha_k \mathbf{v}_k \tag{30}$$

Aside of the two orthogonality requirement of the standard CG, the deflated version adds the additional requirement that

$$\mathbf{W}_m \cdot \mathbf{A}\mathbf{p}_j = 0 \, \forall m \tag{31}$$

Deflated CG method

The resulting algorithm is

1. Define a deflated matrix $\mathbf{A}_{def} = \mathbf{W}^t\mathbf{A}\mathbf{W}$
2. Compute $\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
3. Solve by direct solver $\mathbf{A}_{def}\lambda = \mathbf{W}^t\mathbf{r}_0$
4. modify $\mathbf{p}_0 = \mathbf{r}_0 - \mathbf{W}\lambda$
5. for j=0 ... until convergence
6. $\alpha_j := \frac{\mathbf{r}_j \cdot \mathbf{r}_j}{\mathbf{p}_j \cdot \mathbf{A}\mathbf{p}_j}$
7. $\mathbf{x}_{j+1} := \mathbf{x}_j + \alpha_j \mathbf{p}_j$
8. $\mathbf{r}_{j+1} := \mathbf{r}_j - \alpha_j \mathbf{A}\mathbf{p}_j$
9. $\beta_j := \frac{\mathbf{r}_{j+1} \cdot \mathbf{r}_{j+1}}{\mathbf{r}_j \cdot \mathbf{r}_j}$
10. Solve by direct solver $\mathbf{A}_{def}\lambda = \mathbf{W}^t\mathbf{r}_{j+1}$
11. $\mathbf{p}_{j+1} := \mathbf{r}_{j+1} + \beta_j \mathbf{p}_j - \mathbf{W}\lambda$

Deflated CG method

An appealing choice for the vectors which form the deflated base, is to take them as constans on one portion of the domain and 0 on all the other parts.
This is very similar in concept to what is done in "Non-smoothed" aggregation for multigrid. The idea is that if we distribute evenly over the geometry such patches, the low modes of the solution will be represented quite well by this basis. Hence the deflation will take care of the "long waves" and the CG will work where it is effective ... in the spirit of multigrid methods.

...


UPC
CIMNE
KRATOS