Solving the unsteady Navier-Stokes Equation
An introduction to how to handle time

Gunnar A. Staff

29th November 2004

Abstract

This note gives a brief introduction to some of the available schemes for computing the unsteady Navier-Stokes. We do not discuss spatial discretization, or linear/nonlinear solvers. Schemes like fully implicit, convection-Stokes splitting and velocity-pressure splitting are discussed. No firm conclusions are drawn regarding which approach is best.

Contents

1 Navier-Stokes as a Differential Algebraic problem

We write Navier-Stokes on the familiar form

$$\frac{\partial u}{\partial t} + u \cdot \nabla u + \frac{1}{\rho}\nabla p = \nu \nabla^2 u + f$$
$$\nabla \cdot u = 0$$

We notice immediately that the first equation has a time-derivative, while the second equation doesn't. A natural question arising is how will this effect our choice of solvers in time?

To answer this we start by investigating a singular perturbation problem on the form

$$y' = f(y,z) \tag{1}$$
$$\epsilon z' = g(y,z) \tag{2}$$

As $\epsilon \to 0$, (2) becomes more and more stiff. For the limit $\epsilon = 0$, the corresponding reduced problem is the Differential Algebraical Equation (DAE)

$$\begin{aligned} y' &= f(y,z) \\ 0 &= g(y,z) \end{aligned} \tag{3}$$

The name Differential Algebraic Equation comes from the fact that the first equation is a differential equation, while the second is a pure algebraic equation. We notice that this is also true for Navier-Stokes, where the incompressible constraint serves as a pure algebraic constraint (Lagrangian Multiplier).

There is however an important thing to notice from this. The incompressibility constrain $\nabla \cdot u = 0$ is a stiff term in the Navier-Stokes! This means that regardless of the diffusion operator, the NS is stiff i the ode-sense.

We investigate this further by solving (1)-(2) with a general Runge-Kutta scheme, and then put $\epsilon = 0$.

$$Y_{ni} = y_n + \Delta t \sum_{j=1}^{s} a_{ij} f(Y_{nj}, Z_{nj}) \tag{4}$$
$$\epsilon Z_{ni} = \epsilon z_n + \Delta t \sum_{j=1}^{s} a_{ij} g(Y_{nj}, Z_{nj}) \tag{5}$$
$$y_{n+1} = y_n + \Delta t \sum_{i=1}^{s} b_i f(Y_{nj}, Z_{nj}) \tag{6}$$
$$\epsilon z_{n+1} = \epsilon z_n + \Delta t \sum_{i=1}^{s} b_i g(Y_{nj}, Z_{nj}) \tag{7}$$

We need to define z_{n+1} from (7) independent of ϵ. This is possible by assuming that the coefficient matrix of the Runge-Kutta scheme (a_{ij}) is invertible. This is true for e.g. the Radau schemes. We then write (5) as

$$\Delta t g(Y_{ni}, Z_{ni}) = \epsilon \sum_{j=1}^{s} \omega_{ij} (Z_{nj} - z_n)$$

where ω_{ij} is the inverse of (a_{ij}). This is now inserted into (7), and we put $\epsilon = 0$.

$$
\begin{aligned}
Y_{ni} &= y_n + \Delta t \sum_{j=1}^{s} a_{ij} f(Y_{nj}, Z_{nj}) && (8)\\
0 &= g(Y_{ni}, Z_{ni}) && (9)\\
y_{n+1} &= y_n + \Delta t \sum_{i=1}^{s} b_i f(Y_{nj}, Z_{nj}) && (10)\\
z_{n+1} &= \underbrace{\left(1 - \sum_{i,j=1}^{s} b_i \omega_{ij}\right)}_{=R(\infty)} z_n + \sum_{i,j=1}^{s} b_i \omega_{ij} Z_{nj} && (11)
\end{aligned}
$$

We notice that we may not be able to guarantee that the solution (y_{n+1}, z_{n+1}) satisfies the constraint $g(y, z)$. If we however replace (11) with the condition

$$0 = g(y_{n+1}, z_{n+1}), \tag{12}$$

we will. We notice that if the chosen Runge-Kutta scheme is stiffly accurate, i.e. the method satisfy

$$a_{si} = b_i \quad \text{for } i = 1, \dots s,$$

(12) is automatically satisfied. The Radau schemes with the endpoint as one of the quadrature nodes, is an example of a class of schemes that satisfies this property. Obviously this is a favorable property. It has been shown [13] that the global error for this Radau scheme is of order $2s - 1$ for y, and s for z.

A similar investigation is possible for multi-steps schemes as well. It shows that the BDF schemes is also a good choice for DAE problems.

There exist projected Runge-Kutta Methods, which makes non-stiffly accurate schemes applicable for problems like the Navier-Stokes equations. Assume that the problem is advanced one step in time. The solution of u_{n+1} is then corrected so as to satisfy

$$
\begin{aligned}
\hat{y}_{n+1} &= y_{n+1} + f_z(t_{n+1}, y_{n+1}, z_{n+1})\lambda_{n+1}\\
0 &= g(t_{n+1}, \hat{y}_{n+1})
\end{aligned}
$$

The extra variable λ_{n+1} is only needed for the projection. Then we set $u_{n+1} \leftarrow \hat{u}_{n+1}$, and advance with the next time-step. Note that the solution of z can be determined from the solution for y, and to the same order of accuracy, via a post-processing step.

It can be shown that Navier-Stokes can be a pure index 2 problem, or a Hessenberg Index-2 problem. This is in fact dependent of the chosen solution method, discretization in space etc. A further discussion of this is outside the scope of this note, and the interested reader is referred to [13, 1, 25, 20].

2 Splitting Schemes

There are in general two approaches for constructing splitting-schemes. One is to discretize in time first, yealding pde whith only spatial derivatives. We then

perform aplitting on the operator level. The advantage is that we usually satisfies the inf-sup condition for normal spatial discretization (no need for mixed elements or staggered grid). A disadvantage is that we introduce a boundary layer error on the pressure field, due to the introduction of non-physical boundary conditions for the pressure.There has been presented schemes that claims to avoid this boundary layer errors. More on this splitting schemes can be found in [10].

The other approach is to discretize in space, and incooperate boundary conditons first. Than the splitting is done on the matrix-blocks. The advantage is that we do not have to impose artificial boundarys, leading to no boundary layer errors. But we do end up with a sadle-point problem, which has to satisfy the inf-sup condition through mixed elements, staggered grid or other stabilization techniques. We will only present this type of splitting schemes.

We discretize the Navier-Stokes equation in space using a stable scheme, and incooperate the boundary conditions, ending up with a system of DAE's. We put all coefficients inside the matrixes, and write the system as

$$\begin{aligned} Mu_t + C(u)u + Au - D^T p &= Mf \\ -Du &= 0 \\ u(t=0) &= u_0 \end{aligned}$$

We have in general three approaches for solving this problem:

1. Discretize "everything" using an implicit scheme
2. Use an implicit scheme and then linearize the convection using extrapolation
3. Solving the convection explicit, and the rest implicit

At the moment it seems like a good idè to compute the convection explicitly (because C(u)u is nonlinear and non-symmetric), and the remaining implicitly (due to the stiff diffusion operator and the stiffness generated by the algebraic term). We try with a straight forward discretization using third order Adams-Bashforth for the explicit convection treatment, and implicit Euler for the rest. We end up with the following scheme.

$$\begin{aligned} \left(\frac{M}{\Delta t} + A\right) u^{n+1} - D^T p^{n+1} &= \frac{M}{\Delta t} u^n + Mf^{n+1} \\ &\quad - \left(\frac{23}{11} C(u^n)u^n - \frac{4}{3} C(u^{n-1})u^{n-1} + \frac{5}{12} C(u^{n-2})u^{n-2}\right) \\ -Du^{n+1} &= 0 \end{aligned}$$

This scheme is first order in time due to time splitting error between the explicit and implicit part. It means that changing implicit Euler with BDF2 will not increase the order. In [2, 3] there are presented explicit-implicit schemes of higher order, both for AB-BDF multistep and for ERK-DIRK Runge-Kutta schemes. Note that all this schemes have a stability restriction for the explicit convection. This means that may we have to choose a timestep Δt for stability reasons, and not for accuracy reasons, and we are therefor forced to do unnecessary implicit steps which is costly.

We try a new intuitive approach. We discretize with an implicit scheme, and then extrapolate the nonlinear term. By using the third order BDF3, we get

$$\left(\frac{11}{6\Delta t}M + A\right)u^{n+1} - D^T p^{n+1}$$
$$= \frac{M}{\Delta t}\left(3u^n - \frac{3}{2}u^{n-1} + \frac{1}{3}u^{n-2}\right) + Mf^{n+1} - (3C(u^n)u^n - 3C(u^{n-1})u^{n-1} + C(u^{n-2})u^{n-2})$$
$$-Du^{n+1} = 0$$

This scheme has no splitting error, and is therefore third order in time. Notice however that by introducing the extrapolation for the convection, we alter the stability of the BDF3 scheme, and we can not guarantee that it will be stable for highly imaginary-dominant eigenvalues (high Reynolds numbers). Again we may choose small timestep to achieve stability, and not because of the required accuracy.

2.1 An Operator-Integration-Factor Splitting Method

[18] gives a general framework for how to split the integration in time for a problem on the form

$$\frac{du}{dt} = A(t)u + B(t)u + f(t), \quad u(t=0) = u_0 \tag{13}$$

The approach is called the Operator Integrating Factor method (OIF). Assume that we for some reason wants to split the time-integration of A and B from (13). We introduce the integrating factor $Q(t)$ satisfying the differential equation

$$\frac{dQ}{dt} = -AQ, \quad Q(t^*) = I \quad \text{for some } t^* \tag{14}$$

By multiplying (13) with Q, we get

$$Q\left(\frac{du}{dt} - Au\right) = Q(Bu + f) \tag{15}$$

By using (14), we can write (15) as

$$\frac{d}{dt}(Qu) = Q(Bu + f) \tag{16}$$

We assume that B is a stiff operator (e.g. a diffusion operator), and discretize (16) using a appropriate stiff integrator like e.g. implicit Euler.

$$\frac{Q^{n+1}u^{n+1} - Q^n u^n}{\Delta t} = Q^{n+1}Bu^{n+1} \tag{17}$$

From (14) we choose $t^* = t^{n+1}$, which results in $Q^{n+1} = 1$. (16) is then simplified to

$$\frac{u^{n+1} - Q^n u^n}{\Delta t} = Bu^{n+1} \tag{18}$$

The only unknown beside from u^{n+1}, is our initial value $Q^n u^n$.

We consider now the initial value problem

$$\frac{d\tilde{u}}{d\tau} = A\tilde{u}, \quad \tilde{u}(\tau = 0) = u^n, \quad \tau = t - t^n. \tag{19}$$

Then we can write

$$\frac{d}{dt}(Q\tilde{u}) = \frac{dQ}{dt}\tilde{u} + Q\frac{d\tilde{u}}{dt} = -AQ\tilde{u} + QA\tilde{u} = 0,$$

where we assume that A and Q commute. This means that $Q\tilde{u} = const$, leading to

$$(Q\tilde{u})|_{t=t^{n+1}} = (Q\tilde{u})|_{t=t^n}$$

But from (19) we have that

$$(Q\tilde{u})|_{t=t^n} = Q^n\tilde{u}^n = Q^n\tilde{u}(\tau = 0) = Q^n u^n$$

We end up with

$$Q^{n+1}\tilde{u}^{n+1} = \tilde{u}^{n+1} = Q^n u^n$$

Consequently we can find $Q^n u^n$ from (18) from the solution of $\tilde{u}(\tau)$ at $\tau = \Delta t$, which corresponds to $t = t^{n+1}$.

$$\tilde{u}^{n+1} = Q^n u^n = \tilde{u}(\tau = \Delta t)$$

Remarks

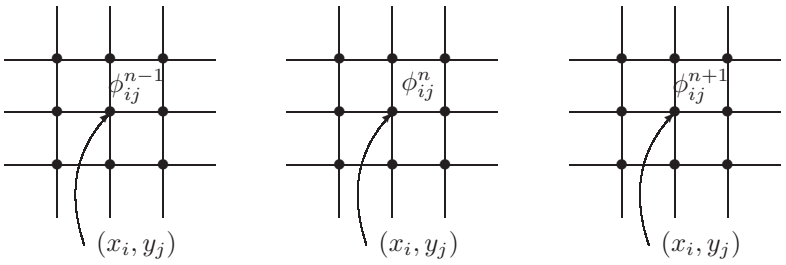
- $Q(t)$ is the exponential function if A and B are not dependent of time, otherwise not.
- In order to compute $Q^n u^n$, we solve the initial value problem (19).
- The integrating factor Q is never explicitly constructed. Only the action of Q is evaluated.
- The problem is now totally decoupled, meaning that (19) only involves the operator A, and (18) only involves the operator B.
- We integrate (19) until $\tau = \Delta t$. Notice that the numerical methods for (19) and (18) can be different. The timesteps for the different schemes can also be different, meaning that $\Delta\tau$ may be a fraction of Δt. By assuming that the chosen scheme for (19) has continuous output, it does not even have to be a fraction.

2.2 A convection-Stokes splitting method

By assuming that the viscosity coefficient is linear, and not dependent on time, it can be shown that the diffusion operator normally lead to a symmetric positive definite matrix. This is a problem where there exists a lot of effective linear solvers. The convection on the other hand often leads to a non-symmetric nonlinear system. Here there do not exist many effective solvers. It would be preferable if we could handle this two systems differently. In other words, we want a convection-Stokes splitting where the convection is handled explicitly while the Stokes is solved implicitly. This can be done in general using the Operator Integration Factor apporach.

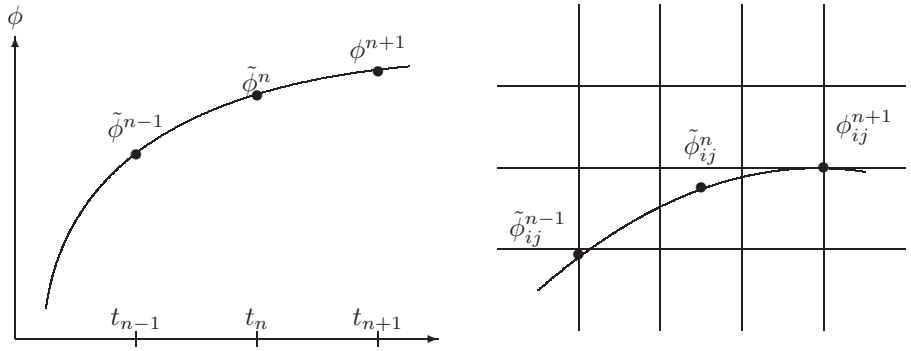
We write Navier-Stokes as

$$M\frac{du}{dt} + C(u)u = \underbrace{-Au + D^T p,\ Du = 0}_{\equiv S\ \text{(Stokes)}} \tag{20}$$



(a) t_{n-1} (b) t_n (c) t_{n+1}

Figure 1: Eulerian representation.



(a) Lagrangian representation (b) Semi-Lagrangian representation

Figure 2: Lagrangian representation

(20) can be written in both Eulerian and Lagrangian form (Figure 1 and Figure 2 shows the difference between Eulerian and Lagrangian representation).

$$M\frac{du}{dt} + C(u)u = Su, \qquad \text{Eulerian} \tag{21}$$

$$M\frac{Du}{Dt} = Su, \qquad \text{Lagrangian} \tag{22}$$

where $\frac{D}{Dt} = \frac{\partial}{\partial t} + \nabla$ is the total derivative, or material derivative.

The vector $\tilde{u}^{n+1}$ represents the values of the velocity at time t^n of those fluid particles which at time t^{n+1} coincide with the grid points. This is called the semi-Lagrangian formulation, showed in Figure 2(b). Note that the positions $x(t)$ of the fluid particles at time t^n will not necessary coincide with the grid points. Also note that we do not have to find the location the fluid particles had at time t^n. We only need the velocity value at this locations. We find this values by integrating forward in time the equation

$$M\frac{d\tilde{u}}{d\tau} = -C(\tilde{u})\tilde{u}, \quad \tilde{u}(0) = u^n. \tag{23}$$

We only use information at the grid points, which means that we don't have to interpolate (which can be expensive).

Note also that the solution of the initial value problem (23) may need different boundary conditions then the original Navier-Stokes problem. (23) is a pure hyperbolic problem, and the prescribed velocity can only be specified along the part of the boundary where the flow is going into the domain ($u \cdot n < 0$).

(22) is discretized using a suitable scheme, e.g. a BDF scheme, or an stiffly accurate Runge-Kutta scheme. We will in this case show it for implicit Euler.

$$M\left(\frac{u^{n+1} - \tilde{u}^{n+1}}{\Delta t}\right) = Su^{n+1}$$

where

$$M\frac{d\tilde{u}}{d\tau} = -C(\tilde{u})\tilde{u}, \quad \tilde{u}(0) = u^n$$

and $\tilde{u}^{n+1} = \tilde{u}(\tau = \Delta t)$. It may be shown that the eigenvalues of $C(u)$ is normally highly imaginary dominant and grows linearly with the number of elements in space. This means that fourth order explicit Runge-Kutta, or the fifth order adaptive Dormand-Prince (explicit Runge-Kutta) is a good choice since a large part of the imaginary axis is included in the stability domain. The explicit solver do have to satisfy the stability condition (often referred to as the CFL condition)

$$|\lambda_{max}(J)|\Delta t < \gamma$$

where J is the Jacobian of $M^{-1}C(u)$, and γ is a the intersection-point of the stability domain and the imaginary axis for the specified scheme. For the standard fourth order explicit Runge-Kutta, $\gamma = 2\sqrt{2}$. But we are only forced to do extra explicit time-steps in order to achieve stability, and not the implicit steps as for standard explicit-implicit splitting.

Notice that $C(u)$ is a nonlinear operator, meaning that we have to re-evaluate it for every quadrature point of the Runge-Kutta scheme. As long as (22) is only first order (implicit Euler), we only have to evaluate $C(u^n)$. This changes the scheme to first order, but the stability domain is preserved (actually the

entire stability analysis is based on the assumption that $M^{-1}C(u)$ is equal for all the quadrature nodes). Using a lower order scheme (first or second) is not likely a good choice because of the very limited stability domain for imaginary eigenvalues!

A possible problem for this approach is the purely inviscid nature of (23). For to large Δt, or for to coarse spatial discretization, spatial oscillation (sometimes called wiggles) may accure. These wiggles then lead to breakdeown through the nonlinear coupling of the convection operator. This places a fundamental limitation on the computational savings possible for this convection-Stokes splitting compared to a normal implicit-explicit computation.

2.2.1 Second order convection-Stokes splitting

We will briefly describe a second order convection-Stokes splitting scheme. From this it will be easy to extend it to even higher order.

We discretize the Lagrangian based equation (22) using a second order BDF

$$M\frac{1}{\Delta t}(\frac{3}{2}u^{n+1} - 2\tilde{u}^{n+1} + \frac{1}{2}\tilde{\tilde{u}}^{n+1}) = Su^{n+1}$$

We need to solve two initial value problems in order to determine $\tilde{u}^{n+1}$, and $\tilde{\tilde{u}}^{n+1}$.

$$\begin{aligned} M\frac{d\tilde{u}}{d\tau} &= -C(\tilde{u})\tilde{u}, \quad \tilde{u}(0) = u^n \\ \tilde{u}^{n+1} &= \tilde{u}(\tau = \Delta t) \end{aligned} \tag{24}$$

$$\begin{aligned} M\frac{d\tilde{\tilde{u}}}{d\tau} &= -C(\tilde{\tilde{u}})\tilde{\tilde{u}}, \quad \tilde{\tilde{u}}(0) = u^{n-1} \\ \tilde{\tilde{u}}^{n+1} &= \tilde{\tilde{u}}(\tau = 2\Delta t) \end{aligned} \tag{25}$$

Note that for the ERK4 to be of at least second order, we need to extrapolate the velocity with a second order extrapolation between t^n and t^{n+1}. Between t^{n-1} and t^n we use a second order interpolation.

2.3 Stokes solvers and pressure-velocity splitting approaches

To this point we have only assumed that we were able to solve the Stokes problem . We will here present a coupled approach based on Uzawa, and use this to motivate the need for pressure-velocity splitting schemes.

2.3.1 Stokes solver using Uzawa

For simplicity we discretize using implicit Euler in time. It is trivial to extend this to higher order multi-step or Runge-Kutta schemes.

$$\begin{aligned} M\left(\frac{u^{n+1} - \tilde{u}^{n+1}}{\Delta t}\right) &= -Au^{n+1} + D^T p^{n+1} + F^{n+1} \\ -Du^{n+1} &= 0 \end{aligned}$$

This is easily transformed into matrix form

$$\begin{bmatrix} H & -D^T \\ -D & 0 \end{bmatrix} \begin{bmatrix} u^{n+1} \\ p^{n+1} \end{bmatrix} = \begin{bmatrix} \left(F^{n+1} + \frac{1}{\Delta t}M\tilde{u}^{n+1}\right) \\ 0 \end{bmatrix}$$

where the Helmholtz operator H is defined as

$$H = \left(A + \frac{1}{\Delta t} M \right).$$

We notice that for smaller timestep Δt, the Helmholtz operator will be more similar to the mass matrix M, and the condition number will decrease. Notice that this is not the case for D, which is not time-dependent. This is due to the Differential Algebraic properties, and is a sign of the stiffness introduced by the algebraic term.

We use the Uzawa procedure (which is only to use Gauss elimination on the block system) to decouple the pressure and velocity into two equations

$$\begin{aligned} DH^{-1}D^T p^{n+1} &= -DHF^* \\ Hu^{n+1} &= F^* + D^T p^{p+1} \\ F^* &= \left(F^{n+1} + \frac{1}{\Delta t} M \tilde{u}^{n+1} \right) \end{aligned}$$

We notice that the pressure solver implies a nested iteration because of H^{-1} on the right-hand side. This is obviously expensive (even though there exists efficent Helmholtz solvers), and we will search for less computationally intensive schemes through a pressure-velocity split.

2.3.2 Pressure-velocity split

For simplicity, and without loss in generality, we write the Stokes on the well known form (minus the force term)

$$Mu_t = -Au + D^T p \tag{26}$$

$$-Du = 0, \tag{27}$$

We now discretize, using implicit Euler as an example, (26) for the intermediate velocity field $\hat{u}^{n+1}$, keeping $p = p^n$ explicit.

$$M \left(\frac{\hat{u}^{n+1} - u^n}{\Delta t} \right) = -A\hat{u}^{n+1} + D^T p^n \tag{28}$$

We can of course not guarantee that $\hat{u}^{n+1}$ satisfies the incompressibility condition (27). We therefore correct this velocity field by computing the necessary change in the pressure in order to produce a final field u^{n+1} which are incompressible. This is done by solving

$$\begin{aligned} M \left(\frac{u^{n+1} - \hat{u}^{n+1}}{\Delta t} \right) &= D^T (p^{n+1} - p^n) \\ -Du^{n+1} &= 0 \end{aligned} \tag{29}$$

By combining (28) and (29), we end up with

$$\begin{aligned} M \left(\frac{u^{n+1} - u^n}{\Delta t} \right) &= -A\hat{u}^{n+1} + D^T p^{n+1} \\ -Du^{n+1} &= 0 \end{aligned}$$

which, apart from $\hat{u}^{n+1}$, is similar to discretizing (26)-(27) using implicit Euler. By evaluating the velocity in $\hat{u}^{n+1}$ instead of u^{n+1}, we introduce a splitting error of order $\mathcal{O}(\Delta t)$, which is of the same order as the implicit Euler.

But how do we solve the saddle-point problem (29)? By writing $\Delta p^{n+1} = p^{n+1} - p^n$, and applying Uzawa, we end up with the system

$$\begin{aligned} DM^{-1}D^T\Delta p^{n+1} &= -\frac{1}{\Delta t}D\hat{u}^{n+1} \\ u^{n+1} &= \hat{u}^{n+1} + \Delta t M^{-1}D^T\Delta p^{p+1} \end{aligned}$$

Again we have a nested iteration because of M^{-1}. But remember that M is the mass matrix. For finite difference and finite volume methods $M = I$, resulting in a non-nested iteration. For spectral element methods of the Galerkin type, M is diagonal and therefore trivial to invert exact, also resulting in a non-nested iteration. For other element discretizations it is a matrix with a small (much smaller then the Helmholtz operator) condition number, and is relatively cheep to handle, compared to a Helmholtz solver. Another approach is to "lump" the mass-matrix, making it diagonal and trivial to invert exactly.

Note that this approach can be derived in the framework of the Operator-Integration-Factor method.

2.4 A Second order convection-pressure-velocity splitting scheme

We will now present a second order accurate (in time) scheme which splits the Navier-Stokes into a convection-Stokes problem, and Stokes into a pressure-velocity problem.

$$M\frac{Du}{Dt} = Su, \tag{30}$$

$$M\frac{du}{d\tau} = -C(u)u \tag{31}$$

Algorithm We assume that (30)-(31) has been semi-discretized using a stable spatial discretization. The only solvers needed here are a Helmholtz solver and a solver for a Poisson like problem.

Convection step Solve (31) for $\tilde{u}^{n+1}$ using fourth order explicit Runge-Kutta as shown in (24).
Solve (31) for $\tilde{\tilde{u}}^{n+1}$ using fourth order explicit Runge-Kutta as shown in (25)

Viscous step Solve

$$\begin{aligned} H\hat{u}^{n+1} &= \frac{1}{\Delta t}M\left(2\tilde{u}^{n+1} - \frac{1}{2}\tilde{\tilde{u}}^{n+1}\right) + D^T\hat{p}^{n+1} + F^{n+1} \\ H &= \left(A + \frac{3}{2\Delta t}M\right) \\ \hat{p}^{n+1} &= 2p^n - p^{n-1} \end{aligned}$$

using a Helmholtz solver

Pressure step Solve

$$DB^{-1}D^T\Delta p^{n+1} = -\frac{1}{\Delta t}D\hat{u}^{n+1}$$

using a solver for a symmetric positive definite system.

Correction step Correct the velocity and update the pressure with

$$u^{n+1} = \hat{u}^{n+1} + \frac{2\Delta t}{3}M^{-1}D^T\Delta p^{n+1}$$
$$p^{n+1} = p^n + \Delta p^{n+1}$$

3 A Parallel approach in time – The Parareal algorithm

In space there exist a variety of parallel schemes. This is in general schemes for evaluating functions, or inverting matrixes in parallel. A parallel scheme in time seems less intuitive because of the very sequential nature of time.

We will here present an approach for parallelizing in time, using a recent proposed scheme called the Parareal algorithm.

3.1 The algorithmic idè

The Parareal algorithm was first presented in [17]. An improved version of the algorithm was presented in [6]. Further improvements and understanding, as well as new applications of the algorithm, were presented in [4] and [19]; our point of departure is the version of the Parareal algorithm presented in these papers.

We consider a set of ordinary differential equations that we would like to integrate from an initial time $t_0 = 0$ to a final time T. The time interval is first decomposed as

$$t_0 = T_0 < T_1 < \cdots < T_n = n\Delta T < T_{n+1} < T_N = T.$$

If we for some reason were given the initial values of every interval in time $[T_i, T_{i+1}]$, $i = 0 \ldots, N-1$, the problem would obviously be parallel. We denote the initial value λ_i, where λ_0 is the initial value given for our sequential problem. The Parareal algorithm is then given as the predictor-corrector scheme

$$\lambda_n^k = \mathcal{F}_{\Delta T}(\lambda_{n-1}^{k-1}) + \mathcal{G}_{\Delta T}(\lambda_{n-1}^k) - \mathcal{G}_{\Delta T}(\lambda_{n-1}^{k-1}), \tag{32}$$

where subscript n refers to the time sub-domain number, superscript k refers to the (global) iteration number, and λ_n^k represents an approximation to the solution at time level n at iteration number k. The fine propagator $\mathcal{F}_{\Delta T}$ represents a fine time discretization of the differential equations, with the property that

$$\lambda_n = \mathcal{F}_{\Delta T}(\lambda_{n-1}) \ , \ n = 1, ..., N,$$

while the coarse propagator $\mathcal{G}_{\Delta T}$ represents an approximation to $\mathcal{F}_{\Delta T}$. Note that since $\mathcal{G}_{\Delta T}$ only have to be an approximation of $\mathcal{F}_{\Delta T}$, they can be discretized using different time-steps, different spatial discretization parameters, different

numerical schemes, and also different equations (e.g. removing high-oscillating terms that is under-sampled by $\mathcal{G}_{\Delta T}$).

Notice that $\mathcal{F}_{\Delta T}$ operates on initial conditions λ_{n-1}^{k-1}, which are known. This implies that $\mathcal{F}_{\Delta T}(\lambda_{n-1}^{k-1})$ can be implemented in parallel. The coarse propagator $\mathcal{G}_{\Delta T}$, on the other hand, operates on initial conditions λ_{n-1}^{k} from the current iteration, and is therefore strictly sequential.

It is sometimes easier to understand how the Parareal algorithm works by rewriting (32) as

$$\lambda_n^k = \mathcal{G}_{\Delta T}(\lambda_{n-1}^k) + \sum_{i=1}^{k-1}(\mathcal{F}_{\Delta T}(\lambda_{n-1}^i) - \lambda_n^i)$$

We see that the algorithm predicts the solution with the coarse propagator $\mathcal{G}_{\Delta T}$, and then corrects it with the difference between the fine propagator and the "exact" solution. Obviously the Parareal algorithm can be seen as a multi-step scheme for the iteration counter k.

3.2 Convergence and stability analysis

The first obvious question is how will this converge, and most important of all, what will it converge to.

The Parareal algorithm will converge to a serial computation using $\mathcal{F}_{\Delta T}$ on the entire domain in time. We denote $|e_n| = |u(t) - u(t_n)|$ as the error between the exact result and a serial computation using $\mathcal{F}_{\Delta T}$. Further we denote $|\epsilon_n^k| = u(t_n) - u^k(t_n)|$ as the error between the Parareal algorithm at the kth iteration and the serial version. Obviously it will be pointless to iterate further then $|\epsilon_n^k| = \mathcal{O}(e_n)$.

It can be shown [5] that the error at time-domain n, for iteration k, can be written as

$$|\epsilon_n^k| \leq C(\Delta T)^{k(m+1)}\binom{n}{k}|u_0|,$$

where m is the order of the scheme in time for the coarse propagator $\mathcal{G}_{\Delta T}$. Notice that this assumes that we use the same spatial discretization (and the same model) for $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$. We notice that the error increases for larger n, which should come as no surprise. We therefore look at the error at the end ($n = N$), and assume only a few iterations ($k \ll N$). Then we write the error as

$$|\epsilon_N^k| \leq C(\Delta T)^{km}|u_0|.$$

We have exponential convergence for the iteration number k.

The stability has proven to be a more complicated aspect of the Parareal algorithm. It can be shown [22] that the stability function of the Parareal algorithm can be described as

$$H = \sum_{i=0}^{k}\binom{n}{i}(\bar{r} - R)^i R^{n-i}$$

where R is the stability function of $\mathcal{G}_{\Delta T}$, and $\bar{r} = r^s$ is the stability function of $\mathcal{F}_{\Delta T}$ over the entire subdomain.

It can be shown that by assuming large real negative eigenvalues, the chosen scheme in time for $\mathcal{G}_{\Delta T}$ has to be strong A-stable with the asymptotic property

$\lim_{z\to-\infty} R(z) \leq \frac{1}{2}$. This is not a problem since a stiff problem should be approximated using a stiffly accurate, or at least a L-stable scheme.

For complex eigenvalues $\mu_i = x_i + \mathbf{i}y_i$, $x_i < 0$, the stability restrictions are more severe. We know that since both $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$ are stable, we can describe the stability functions as

$$\bar{r} = e^{x_{\bar{r}}} e^{\mathbf{i}\theta}$$
$$R = e^{x_R} e^{\mathbf{i}(\theta+\varepsilon)}$$

where θ represents the angle on the complex circle, ε represents the phase-difference between $\bar{r}$ and R, and both $\bar{r}$ and R lies on or inside the complex unity-circle. We can now write the stability restrictions as

$$e^{x_R} \leq \frac{1}{2} \frac{1 - e^{2x_{\bar{r}}}}{1 - e^{x_{\bar{r}}} \cos\varepsilon}$$

Obviously we run into problems when μ_i is pure imaginary or imaginary dominant. Note however that this is a worst case estimate, and the restrictions are not that severe for a smaller number of iterations k.

3.3 Parareal applied to the Navier-Stokes equation

In [9], the Naiver-Stokes is solved using the Parareal algorithm. In the previous section we have showed that for problems where the eigenvalues are highly imaginary dominant, the Parareal algorithm may be unstable. It is believed that this can be improved by computing Naiver-Stokes using a convection-Stokes splitting as described in Section 2.2, but this has not been thoroughly analyzed yeat.

4 General Remarks

In this note we have presented some of the available discretization schemes in time for Navier-Stokes. It is not possible to give a general rule of what is the best solution method. It depends e.g. on

- The chosen spatial discretization
- Your grid
- How efficient your available linear and non-linear solvers are for the given spatial discretization and grid
- The required error tolerance
- The boundary conditions, initial data and coefficients (Reynolds number)
- Should the code be a black-box Navier-Stokes solver, or a highly optimized solver for e specified problem

In general it is safe to say that computing everything implicit and coupled is most robust and stable, while doing a convection-Stokes and a pressure-velocity splitting is faster.

We refer to [7] and [23] for a comparison of different solvers and solution strategies.

References

[1] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998.

[2] Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. Implicit-explicit runge-kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics*, 25:151–167, 1997.

[3] Uri M. Ascher, Steven J. Ruuth, and B. Wetton. Implicit-explicit methods for time-dependent pde's. *SIAM J. Numer. Anal*, 32:797–823, 1999.

[4] L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah. Parallel in time molecular dynamics simulations. *Physical Review. E*, 66, 2002.

[5] Guillaume Bal. Parallelization in time of (stochastic) ordinary differential equations. submitted, 2002.

[6] Guillaume Bal and Yvon Maday. A "parareal" time discretization for non-linear pde's with application to the pricing of an american put. In Luca F. Pavarino and Andrea Toselli, editors, *Recent Developments in domain Decomposition Methods*, volume 23 of *Lecture Notes in Computational Science and Engineering*, pages 189–202. Springer, 2002.

[7] Mark H. Carpenter, Sally A. Viken, and Eric J. Nielsen. The efficiency of high order temporal schemes. Technical report, NASA, 2003.

[8] M.O. Deville, P.F. Fischer, and E.H. Mund. *High-Order Methods for Incompressible Fluid Flow*. Cambridge Monographs in Applied and Computational Mathematics. Cambridge, 2002.

[9] Paul F. Fischer, Frèdèric Hecht, and Yvon Maday. A parareal in time semi-implicit approximation of the navier-stokes equations. In R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, and J. Xu, editors, *Domain Decomposition Methods in Science and Engineering*, volume 40 of *Lecture Notes in Computational Science and Engineering*, pages 433–440. Springer, 2004.

[10] J.L. Guermond, P. Minev, and Jie Shen. An overview of projection methods for incompressible flows. Submitted to International Journal of Numerical Methods in Engineering, 1990.

[11] J.L. Guermond and Jie Shen. A new class of truly consistent splitting schemes for incompressible flows. *Journal of Computational Physics*, 192:262–276, 2003.

[12] E. Hairer, S.P Nørsett, and G. Wanner. *Solving Ordinary Equations I*, volume 8 of *Springer Series in Computational Mathematics*. Springer, 2nd edition, 2000.

[13] E. Hairer and G. Wanner. *Solving Ordinary Equations II*, volume 14 of *Springer Series in Computational Mathematics*. Springer, 2nd edition, 2002.

[14] Ernst Hairer and Gerhard Wanner. Stiff differential equations solved by radau methods. *Journal of Computational and Applied Mathematics*, 111:93–111, 1999.

[15] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge, 1996.

[16] Hans Petter Langtangen, Kent-Andre Mardal, and Ragnar Winther. Numerical methods for incompressible viscous flow.

[17] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Résolution d'edp par un schéma en temps pararéel. *C.R.Acad Sci. Paris Sér. I Math*, 332:661–668, 2001.

[18] Yvon Maday, Anthony T. Patera, and Einar M. Rønqusit. An operator-integration-factor splitting method for time-dependent problems: Application to incompressible fluid flow. *Journal of Scientific Computing*, 5:263–292, 1990.

[19] Yvon Maday and Gabriel Turinici. A parareal in time procedure for the control of partial differential equations. *C.R.Acad Sci. Paris Sér. I Math*, 335:387–392, 2002.

[20] Wade S. Martinson and Paul I. Barton. A differentiation index for partial differential-algebraic equations. *SIAM Journal of Scientific Computing*, 21:2295–2315, 2000.

[21] Alfio Quarteroni and Alberto Valli. *Numerical Approximation of Partial Differential Equations*, volume 23 of *Springer Series in Computational Mathematics*. Springer, 2nd edition, 1997.

[22] Gunnar Staff and Einar Rønquist. Stability of the parareal algorithm. In R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, and J. Xu, editors, *Domain Decomposition Methods in Science and Engineering*, volume 40 of *Lecture Notes in Computational Science and Engineering*, pages 449–456. Springer, 2004.

[23] Stefan Turek. A comparative study of some time-stepping techniques for the incompressible navier-stokes equations: From fully implicit nonlinear schemes to semi-implicit projection methods. Technical report, University of Heidelberg, 1995.

[24] Stefan Turek. *Efficient Solvers for Incompressible Flow Problems*, volume 6 of *Lecture Notes in Computational Science and Engineering*. Springer, 1999.

[25] Jörg Weickert. Navier-stokes equations as a differential-algebraic system. Technical report, Technische Universität Chemnitz-Zwickau, 1996.