# M4R

Anthony Webster CID 01051827

March 2nd, 2019

## Introduction

We seek to create an efficient solver of the steady Navier stokes equations in an arbitrary domain in H(div) space. The steady Navier Stokes equation are as follows :

$$u \cdot \nabla u = -\nabla p + \mu \nabla^2 u + F \qquad (1)$$
$$\nabla \cdot u = 0 \qquad (2)$$

Since we are working in an arbitrary domain we will be using finite element. Hence we will be approximating $u$ by a function in a different space. In our case we will use Brezzi-Douglas-Marini element (BDM) for $u$ and discontinuous galerkin (DG) for $p$. BDM2 is continuous at the boundary in the normal but not necessarily otherwise. Hence some operations (especially in the viscosity and convective part) will be ill defined if we just attempted to find a weak form. While this does make the discretisation harder we in exchange have a number of advantages, such as being globally exactly divergence free due to using the $H(div)$ space.

We will first quickly introduce some general concepts and methods which students and professors not directly familiar with finite elements may find usefull to follow the remainder of the project as well as a description of how this project would be tackled utilising a more conventional finite element. Then we will show how discretized the equations and built different parts of our solvers. Lastly we will consider some example cases and deduce that the algorithm converges to the right result at satisfactory rates.

## Tools and Definitions

### Finite Elements Method

The finite element method (FEM) is a numerical method which approximates the solution to an exact partial differential equation. The property that it works on arbitrary domains, as opposed to the finite difference

method which finds the exact solution to an approximated discretized partial differential equation, is one of the reasons it is widely used in research and industry.

We will now consider the case of applying the FEM to the navier stokes equations using the more classic lagrange finite element.
As seen in Common and Unusual Finite Elements [7], the lagrange finite element approximates a function using polynomials of degree q defined by their pointwise values at an array of points (such as a uniform lattice).
For instance take a $q = 1$ on a 1D problem cut into 3 cells. This is effectively a linear interpolation, we approximate the function on each of the three cells by a linear function. This linear function is defined on each cell using two points whithin it, such as the two edge points.

### Brezzi-Douglas-Marini Element

The paper Common and Unusual Finite Elements [7] states that the BDM space on one triangle $K$ is composed by polynomials of order $q$ defined by the normal component on each edge. If $q > 1$ it is also defined using the integration against gradients of the polynomials on the triangle. If $q > 2$ we then also use the integration against curls of $b_K P_{q-2}(K)$ where $b_{q-2}$ is the cubic bubble function associated with $K$ and $P_q(K)$ is the set of polynomials defined on $K$.

### Discontinous Galerkin

The discontinous Galerkin Element space is the space of piecewise linear continuous functions.

### Newton's Method

Up until now we have described how to setup the stokes system, then precondition it and finally added the advection term to the precondititioned system. However we did not yet touch the topic of how, in practice, we actually solve the equations.
We use newton's iterations to this effect.
We will first describe it in a general fashion. Take the system :

$$F(x) = 0$$

Where $F$ is a vector function and $x$ is vector of unkowns.
Then similarly to the 1D newton we start with a guess $x = x_0$ we then

perform the following :

$$DF(x_n)\Delta x = -F(x_n)x_{n+1} = x_n + \Delta x \qquad (3)$$

Where $DF(x_n)$ is the jacobian.

## Preconditioners

We will first describe how preconditioners, such as the Schur Preconditioner work in general. We see this in the book Multilevel Block Factorization Preconditioners [4] on pages 49 to 51.
Take the system :

$$Mx = b$$

Where $M$ is a very large matrix, $x$ and unkown vector we need to solve for (in our case this will be a vector of coefficients for both $u$ and $p$) and $b$ a constant vector. If $M$ is a symmetric positive definite sparse matrix then $M$ applied to a vector is cheap to compute. Hence we can easily find $r$ the residual such that :

$$r = b - Mx$$

We then use the resulting $r$ in order to provide a correction on our result. However $M$ may not be well-behaved. In fact in our problem we know it is usually not.
In this case we use a preconditioning matrix, $P$ to map the system in such a way that $P^{-1}$ is cheap to compute, this operation can easily be parallelized and the condition number, which describes how accurate this operation is (for more details refer to page 50 of the book), is lowered. We then numerically solve the transformed and hopefully better behaved system :

$$P^{-1}(Mx - b) = 0$$

# Implementing the H(div) Solver

Since we have two problem terms we will deal with each separately. Hence we will first consider the stokes equation, ignoring the advection term and focusing on the viscosity term for now :

$$0 = -\nabla p + \mu \nabla^2 u \qquad (4)$$
$$\nabla \cdot u = 0 \qquad (5)$$

3

## Stokes Equation

First we start from the stokes equation and determine its weak form.

For this purpose let us introduce two test functions, $w$ in BDM and $q$ in DG. Let us multiply the stokes equation by $w$ and integrate over the domain.

$$F = \nabla p - \mu \nabla^2 u$$

$$\int_\omega w \cdot F dx = \int_\omega (w \cdot \nabla p) dx - \int_\omega (\mu w \cdot \nabla^2 u) dx$$

Focusing on the viscosity term we get via integration by parts over an individual element $e$ :

$$\int_e (w \cdot \nabla^2 u) dx = - \sum_e \int_e \nabla_h(w) : \nabla_h(u) dx + \int_\Gamma 2\{w_i n_j\}\{\frac{\partial u_i}{\partial x_j}\} dS$$

The first term can be summed up to an integral over $\Omega$ and the second one is asymmetrical. We can add a term to make it symetrical, which helps numerical stability.

$$- \int_\Omega \nabla_h(w) : \nabla_h(u) dx + \int_\Gamma 2\{w_i n_j\}\{\frac{\partial u_i}{\partial x_j}\} dS + \int_\Gamma 2\{u_i n_j\}\{\frac{\partial w_i}{\partial x_j}\} dS$$

Since our added term is 0 in continous space, $u$ should be continous thus the $\{u_i n_j\}$ term 0, this does not change the equation.

Now we also add the term $\alpha \int_\Gamma \frac{1}{h} J(w_i) J(u_i) dS$. According to sources [3] provided that $\alpha$ is about 10 stability is increased. $J$ indicates a jump between the two cells. The numerical value of $h$ is the average distance of a side. In our case we defined it as the average cellvolume over the length of the facettes between them for each individual facette element $\Gamma$.

The $J(u_i)$ term makes this integral 0 in continous space so this does not change the solution.

Lastly we need to add in all the exterior boundary terms corresponding to each of the above terms we added. This is allows us to set up penalty terms so that our solution tends to those boundary values. Hence our final viscous term becomes :

$$- \int_\Omega \nabla_h(w) : \nabla_h(u) dx + \int_\Gamma 2\{w_i n_j\}\{\frac{\partial u_i}{\partial x_j}\} dS + \int_\Gamma 2\{u_i n_j\}\{\frac{\partial w_i}{\partial x_j}\} dS$$

$$+ \alpha \int_\Gamma \frac{1}{h} j(w_i) j(u_i) dS + \int_{\partial\Omega} w_i n_j \frac{u_i - u_i^0}{x_j} + (u_i - u_i^0) n_j \frac{w_i}{x_j} ds + \alpha \int_{\partial\Omega} \frac{1}{h} w_i (u_i - u_i^0) ds$$

Where $u_i^0$ are the boundary values. We will call this term $v_\mu$.

Back to our original equation we get :

$$\int_\Omega w \cdot F dx = \int_\Omega (w \cdot \nabla p) dx - v_\mu \tag{6}$$

Let us consider the continuity equation again :

$$\nabla \cdot u = 0$$

Multiplying by $q$ and integrating we get :

$$\int_\Omega q \nabla \cdot u \, dx = 0$$

This is 0, so we add it to the weak formulation (3) we obtained, getting one weak formulation which includes both pressure and velocity. Additionally we can use some basic vector identities to get our final equation for the stokes problem :

$$\int_\Omega w \cdot F dx = \int_\Omega (\nabla \cdot (pw)) dx - \int_\Omega (p \nabla \cdot (w)) dx + \int_\Omega q (\nabla \cdot u) dx - v_\mu \quad (7)$$

## Applying the Schur Preconditioner to our problem

In an effort to get our iteration count to be independent of mesh-size, we will apply a Schur preconditioner similarly described in chapter 3 of Multilevel Block Factorization Preconditioners [4] as well as in source [2].

This is a well-known method, however as we will be modifying it, we will quickly describe it as well. Our sytem effectively amounts to solving a matrix equation of the form :

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

Where $B$ is the discretized divergence operator and $B^T$ the discrete gradient operator. A contains the viscous term. Setting $S = -BA^{-1}B^T$, we can factorize the above and get an expression whose inverse is :

$$\begin{bmatrix} I & -A^{-1}B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} A^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -BA^{-1} & I \end{bmatrix}$$

Hence, if we can find $A^{-1}$ and $S^{-1}$ we can easily solve the equation. We solve both via a Lu factorization.

From [2] we add the term $\gamma \nabla \cdot v \, \nabla \cdot w$ to the equation (4). We thus get :

$$\int_\Omega w \cdot F dx = \int_\Omega (\nabla \cdot (pw)) dx - \int_\Omega (p \nabla \cdot (w)) dx + \int_\Omega q (\nabla \cdot u) dx - v_\mu + \int_\Omega \gamma \nabla \cdot v \, \nabla \cdot w dx \quad (8)$$

This added term is 0 in continous space and therefore doesn't change it. However the modified system is more easily well approximated and this approximation improves for larger values of $\gamma$, from [2]. We also now use an incomplete Lu (ILu) factorization for $S^{-1}$. The ILu factorization is faster than the full Lu factorization. However in some cases, such as for $A^{-1}$ our testing has shown that ILu fails. This is because $A$ is a dense matrix which includes the viscosity term.

## The Continuation Method

Now we include a modified advection term :

$$0 = LHS - RHS - c(u \cdot \nabla u) \tag{9}$$

Where LHS and RHS are the Left hand and Right hand side of equation (5).

Given the navier stokes equations we could try to naively set $c = 1$ and then use Newton's method applying the above preconditioning and discretization methods. However at high Reynolds number this approach quickly fails unless the mesh is refined, leading to high computational cost.

Hence, in such cases, we will initially set $c$, referred to as the advection switch, to 0. Applying the solver to $c = 0$ we should easily find a solution. Using this solution as a starting point we then calculate the derivative of (6) with respect to the advection switch.

We then solve again with the non-linear solver using the previously obtained solution for $c = 1$. If this fails we reduce $c$ until it works. We then repeat this procedure until $c = 1$.

This allows us to rapidly solve simple problems, for which the advection switch is unecessary, as well as more complex problems, since we can use more efficient step-size in $c$ initially saving iterations, automatically.

In some circumstances however even with a variable step-size in $c$ we fail to obtain a solution. In our code if the step-size is below $10^{-3}$ our solver stops and fails.

## Stability Analysis

While we are mainly interested in the steady-state solution of the navier-stokes equation we also need to consider whether or not the solution is stable. To that end we need to solve a full navier stokes, including the time derivative. This will be used to determine if small perturbations of the solution die down.

### Time stepping method

In practice our problem amounts to the following, very generalized, equation :

$$F(u) = 0 \tag{10}$$

Where F is some function. this is in practice how the firedrake non-linear solver solves the equation.

If we add in the time-terms we get :

$$\frac{\partial u}{\partial t} + F(u, p) = 0 \tag{11}$$

While we could try and solve this equation directly using Backwards Euler or Midpoint Rule testing shows that this does not converge. Instead we use Picards iteration along with the Midpoint rule as described in the following equation :

$$u^{n+1} - u^n + \Delta t(\frac{1}{2}G(u^n, p^n) + \frac{1}{2}G(u^{n+1}, p^{n+1}) + (u^n \cdot \nabla u^n)) = 0 \quad (12)$$

Here $G = RHS - LHS$. We applied the midpoint rule to the linear terms of the equation while the non-linear term, the advection term, was evaluated at the previous time step. Provided the time step is low enough this will always converge, regardless of inital guess. Optionally we can afterwards use the method we described first, Newton's method and Midpoint Rule, to get a more accurate result after a few Picards iterations.

$$u^{n+1} - u^n + \Delta t(\frac{1}{2}F(u^n, p^n) + \frac{1}{2}F(u^{n+1}, p^{n+1})) = 0 \quad (13)$$

## Test Cases

### Manufactured Solution

As is often the case for numerical schemes we will first test it's effectiveness with a manufactured solution. We will thus find a solution to the equations. In our case we used $u_x = \mu + e^y$, $u_y = \mu + e^x$ and $p = e^{xy}$.
Plugging this into the equations we get that these do indeed solve the navier-stokes equations. We now adjust boundary conditions so that on $\delta\Omega$ $u$ and $p$ have the values given above.
We can thus use this to determine the error of our solution.

### Known Case : Flow Past a Pipe

## Citations

[1] Imperial College London *Firedrake Project* Available from : https://www.firedrakeproject.org [Accessed througout 2018-2019]


[2] Patrick E. Farrell, Lawrence Mitchell, Florian Wechsung An Augmented Lagrangian Preconditioner For The 3D Stationary Incompressible Navier–Stokes Equation At High Reynolds Number *SIAM J. Sci Comput 28(6)*2006


[3]Arnold, Douglas N., et al. "Unified analysis of discontinuous Galerkin methods for elliptic problems." SIAM journal on numerical analysis 39.5 (2002): 1749-1779.

[4] Vassilevski, Panayot S. Multilevel Block Factorization Preconditioners: Matrix-based Analysis and Algorithms for Solving Finite Element Equations. New York, NY: Springer New York, 2008. Web.