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 :

$$u \cdot \nabla u = -\nabla p + \mu \nabla^2 u + F \tag{1}$$

$$\nabla \cdot u = 0 \tag{2}$$

Not that from working in the H_{div} space the second equation is naturally satisfied globally.
First we will describe the classic Finite Element Method, its advantages and disadvantages, hence explaining why we are using a modified method. Secondly we will explain a number of definitions and tools used in the project and which may be unfamiliar to some readers. Then we will show how to find the weak form of 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.

Classic Finite Elements Method

The finite element method (FEM) is a numerical method which approximates the solution to an exact partial differential equation [5]. 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, industry and this project.

We will now consider the case of applying the FEM to the stokes equations using the more classic lagrange finite element. We will specifically consider a test case created by the method of manufactured solution.

$$F = -\nabla p + \mu \nabla^2 u \tag{3}$$
$$\nabla \cdot u = 0 \tag{4}$$

Using the boundary conditions are $u = (u_0, u_1) = (e^{xy}, e^{xy})$ on the edges of the unit square, our domain.
It can easily be verified that the solution to this equation is :

$$\begin{aligned} u_x &= -cos(2\pi x)sin(2\pi y) \\ u_y &= sin(2\pi x)cos(2\pi y) \\ p &= 0 \\ F_x &= -4\pi^2 cos(2\pi x)cos(2\pi y) + 4\pi^2 sin(2\pi x)sin(2\pi y) \\ F_y &= 4\pi^2 cos(2\pi x)cos(2\pi y) - 4\pi^2 sin(2\pi x)sin(2\pi y) \end{aligned} \tag{5}$$

As seen in Common and Unusual Finite Elements [7], the lagrange finite element approximates a function on a cell using polynomials of degree q defined by their pointwise values at an array of points (such as a uniform lattice).
For instance let us approximate $f = sin(x)$ from 0 to 2π using 1D first order lagrange finite elements. We approximate the function on each of the four cells by a linear function. This linear function is defined on each cell using two points whithin it, the two edge points. Hence we get figure . In practice one can consider f to be the sum of these linear functions. Note that this element is continuous across the "edges" of a cell, since regardless of cell the value of the interpolated function at the "edge" will be identical. This concept can be extended to 2D.

It follows that we first cut up our domain into a mesh. Commonly these meshes are made up of triangles for simplicity, although squares or even curved elements for simulations on a sphere are also used. Then we approximate u and p using the lagrange element. Hence we get that for instance u is the sum of a set of known piecewise linear functions (as they are determined by the mesh) but with unknown coeffecients. The aim of the rest of the method is to determine these unknown coefficients.
For that end we need to find what is called the weak form of the equation. We multiply (3) by an arbitrary function v. Hence we get :

$$\int_\omega -v \cdot \nabla p + \mu v \cdot \nabla^2 u dx = \int_\omega v \cdot F dx$$

Since our domain ω is cut out into triangles we can cut the integral into triangles and work on each one individually. Hence for each cell c in ω we get :

$$\int_c -p\nabla \cdot v + \mu v \cdot \nabla^2 u dx = \int_c v \cdot F dx$$

However since u and v will be approximated by functions which may be discontinuous in their derivatives ∇^2 may not be defined. Hence we need to use integration by parts to get :

$$\int_c -p\nabla \cdot v + \mu \nabla v \cdot \nabla u dx = \int_c v \cdot F dx$$

Finally multiplying (4) by a similar pressure test function q and adding it to the left hand side of the above we get :

$$\int_c -p\nabla \cdot v + \mu \nabla v \cdot \nabla u dx + \int_c q\nabla \cdot u dx = \int_c v \cdot F dx \tag{6}$$

Now we use the fact that q, v, F, u and p are a sum of known linear functions and coefficients.
Pulling out and removing the coefficients of v we get that :

$$\int_c \sum_i \sum_j (-p_j \phi_j \nabla \cdot \Phi_i + \mu \nabla \Phi_i \cdot u_j \nabla \Phi_j + \phi_i u_j \nabla \Phi_j) dx = \int_c \sum_i \phi_i F_i \phi_i dx$$

Combining all the triangles we can eventually turn into a matrix equation of the form :

$$Mx = b \tag{7}$$

Where x is a vector of coefficients of u and p.
We can solve this easily using a variety of linear systems solvers. Note in particular that since the majority of basis functions Φ and ϕ are 0 in any given triangle the matrix M will be sparse.
While we locally enforced the divergence-free equation by including it in the weak form, the global flow may not be divergence-free. In order to achieve that more complex mathematical tools would be required. Using the above and implementing it in firedrake we get figures .

Tools and Definitions

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.
This finite element, combined with the DG element for pressure, discretizes

the function space H(div). A property we thus get from simply working in BDM is that our solution will be exactly divergence free globally and locally. Additionally we will be able to employ an enhanced lagrangian preconditioner thanks to the global divergence free property [2].

However unlike the lagrange elements we can no longer set a dirichelet boundary condition's tangential component. Therefore we need to find alternative approaches to enforce these conditions. Additionally some operations (such as the curl) are undefined in this space. Due to possible discontinuities at the boundaries we also have to be carefull with all boundary integrals.

In general the weak form will be far more involved.

Discontinous Galerkin

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

Newton's Method

Newton's method is a generalized method for solving non-linear matrix equations of the form :

$$F(x, y) = 0$$

Where F is a vector function and x is vector of unknowns. Note that F may be non-linear and have constant components. For instance we could have $F = x^T Ax + Mx - b$, where A and M are constant matrices and b is a constant vector.
We start with a guess $x = x_0$ and then determine the Jacobian of $F(x)$, call it $DF(x)$.
We determine the jacobian by calculating the Gateaux derivative, defined as follows [12] :

$$DF_{x_n}(\Delta x) = lim_{t\to 0^+} \frac{F(x_n + \epsilon\Delta x) - F(x_n)}{\epsilon} \tag{8}$$

We then use the vector taylor expansion :

$$F(x_{n+1}) = F(x_n) + DF_{x_n}(\Delta x)\Delta x$$

Hence we get the algorithm :

$$DF_{x_n}(\Delta x)\Delta x = -F(x_n) \tag{9}$$

$$x_{n+1} = x_n + \Delta x \tag{10}$$

We solve for Δx using x_n and the taylor expansion approximation and then use this result to determine x_{n+1}. This is a linear problem which can be solved via an Lu Factorization for instance.
Note that the Newton's method only converges and is unique if it is within a ball around the solution [10-11].

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, assumed for simplicity to be symmetric, x and unknown 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, the ratio of largest and smallest eigenvalue which describes the behaviour of the system [13], 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 :

$$0 = -\nabla p + \mu \nabla^2 u \tag{11}$$

$$\nabla \cdot u = 0 \tag{12}$$

Stokes Equation

We start off by cutting our computational domain Ω into a set of cells e. As discussed above it is now required to find the weak form of equations (9) and (10). For this purpose let us introduce two test functions, w in BDM for velocity and q in DG for pressure. Let us multiply equation (9) by w and integrate over the domain as seen in the first section.

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

For simplicity and sparsness of the resulting matrix we will work on each cell e individually :

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

Focusing on the viscosity, or second, term we get via integration by parts :

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

The first term can be summed up to an integral over Ω and the second one is asymmetrical. We can add a term to make it symetrical. This helps with numerical stability of the final matrix equation. Additionally some linear solver take advantage of this structure.

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

u should be continous in the solution space thus the $\{u_i n_j\}$ term will cancel out. Hence 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 α is larger than 10 numerical 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 cell-volume over the length of the facettes between them for each individual facette element Γ.
The $J(u_i)$ term makes this integral 0 in continous space so this does not change the solution.

As mentioned in the BDM definition section we cannot set the tangential component of a dirichelet boundary condition in the conventional finite element manner. So instead we use penalty terms, which should be 0 and thus

push the solution towards respecting the boundary conditions. In practice this just means taking all the terms over the edge of a cell and considering them on the edge of the domain instead.
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_μ.
Back to our original equation we get :

$$\int_\Omega w \cdot F dx = \int_\Omega (w \cdot \nabla p)dx - v_\mu \tag{13}$$

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.
We need this since we want to turn the entire system into a single matrix equation which includes all the conditions.
Additionally we can use some basic vector identities to get our weak equation for the stokes problem :

$$\int_\Omega w \cdot F dx = \int_\Omega (\nabla \cdot (pw))dx - \int_\Omega (p\nabla \cdot (w))dx + \int_\Omega q(\nabla \cdot u)dx - v_\mu \tag{14}$$

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.
We solve for A^{-1}, or to be exact the operation $A^{-1}v$ via an Lu Factorization. The Lu factorization is a direct solver, and thus is innefficient. The algorithm would have to be performed in a sequential manner, preventing usage of parallelisation. We did mitigate this through the usage of the "mumps" subroutine. Additionally there was an However more efficient implementations would require this to be modified.
However S^{-1}, the inverse of the shur complement is difficult to determine because it is usually dense. However research has shown [2 & 9] that we can solve this issue by using a "grad-div" stabilization method. We add the term $\gamma \nabla \cdot v \, \nabla \cdot w$ to the equation (!). Note that this term is 0 in continuous space. We thus get :

$$\int_\Omega w \cdot F dx = \int_\Omega (\nabla \cdot (pw)) dx - \int_\Omega (p \nabla \cdot (w)) dx + \int_\Omega q(\nabla \cdot u) dx - v_\mu + \int_\Omega \gamma \nabla \cdot v \, \nabla \cdot w dx \tag{15}$$

This doesn't change the equation, thus should result in the same solution. However acoording to the same papers [2 & 9] we can then approximate S by the pressure mass matrix. We then perform ILu in order to solve for S^{-1}. ILu performs the Lu Factorization on various blocks of the matrix, allowing it to be parallelised.

The Continuation Method

Now we include a modified advection term :

$$0 = LHS - RHS - c(u \cdot \nabla u) \tag{16}$$

Where LHS and RHS are the Left hand and Right hand side of equation (5). Note that this is equation is not linear, hence we will need to use Newton's method.
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 :

$$F(u) = 0 \tag{17}$$

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 :

$$\frac{\partial u}{\partial t} + F(u, p) = 0 \tag{18}$$

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 :

$$u^{n+1} - u^n + \Delta t(\frac{1}{2}G(u^n, p^n) + \frac{1}{2}G(u^{n+1}, p^{n+1}) + (u^n \cdot \nabla u^n)) = 0 \tag{19}$$

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 could afterwards use the method we described first, Newton's method and Midpoint Rule, to get a more accurate result after a few Picards iterations.

$$u^{n+1} - u^n + \Delta t(\frac{1}{2}F(u^n, p^n) + \frac{1}{2}F(u^{n+1}, p^{n+1})) = 0 \tag{20}$$

Stability Analysis

When interested in the stability of a solution analytically we commonly first determine the solution. We would then consider the normal modes form of the equation before perturbing it. After some algebra we determin the most unstable mode and deduce from that.
While there are methods to determine these unstable modes numerically we will not use either of these methods here.
Instead we will use a far simpler algorithm. We will consider random perturbations and then use the time stepping method to see if they grow.

Test Cases

Stokes Convergence

Once again we will primarily focus on the stokes equation (9-10).
For this we will use the manufactured solution $u_x = cos(y)exp(x)$, $u_y = -sin(y)exp(x)$ for which we get $p = 0$.
Figures and show us the convergence graph in velocity and pressure for the H_{div} solver, while figures and show us the convergence graphs obtained in the first section of the report.

Navier-Stokes Convergence

As is often the case for numerical schemes we will first test it's effectiveness with a manufactured solution.
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 Cylinder

Known Case : Cylinder flow Stability

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.

[5]Ibrahimbegovic, Adnan. *Nonlinear Solid Mechanics*. Vol. 160. Dordrecht: Springer Netherlands, 2009. Solid Mechanics and Its Applications. Web.

[6]

[7] Kirby R.C., Logg A., Rognes M.E., Terrel A.R. (2012) *Common and unusual finite elements*. In: Logg A., Mardal KA., Wells G. (eds) Automated Solution of Differential Equations by the Finite Element Method. Lecture Notes in Computational Science and Engineering, vol 84. Springer, Berlin, Heidelberg

[8]

[9]

[10] Huan Zhengda, *The convergence ball of Newton's method and the uniqueness ball of equations under Hölder-type continuous derivatives*, Computers

& Mathematics with Applications, Volume 47, Issues 2–3, 2004, Pages 247-251, ISSN 0898-1221, https://doi.org/10.1016/S0898-1221(04)90021-1.

[11] X Wang, *Convergence of Newton's method and uniqueness of the solution of equations in Banach space*, IMA Journal of Numerical Analysis, Volume 20, Issue 1, January 2000, Pages 123–134, https://doi.org/10.1093/imanum/20.1.123

[12] Bae, Jong Sook, and Sangsuk Yie. *Range of Gateaux Differentiable Operators and Local Expansions* Pacific J. Math. 125.2 (1986): 289-300. Web.

[13] Zi-Cai Li, Cheng-Sheng Chien, Hung-Tsai Huang, *Effective condition number for finite difference method*, Journal of Computational and Applied Mathematics, Volume 198, Issue 1, 2007, Pages 208-235, ISSN 0377-0427, https://doi.org/10.1016/j.cam.2005.11.037.