INTERNATIONAL CENTRE FOR
NUMERICAL METHODS IN ENGINEERING

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER OF SCIENCE IN COMPUTATIONAL MECHANICS

Finite Element in Fluids

Homework 6B - Navier Stokes Problem (Picard, Newton-Raphson)

Eugenio José Muttio Zavala

April 3, 2019

Submitted To:
Prof. Antonio Huerta
Prof. Matteo Giacomini
Prof. Pablo Saez

1 NAVIER STOKES PROBLEM

PROBLEM STATEMENT

Consider the steady Navier-Stokes equations 1.1, in which it is included the convective term, then in comparison with the Stokes equations, the viscous expression is not the only term that can affect the motion of the fluid particles in the momentum equation. As the Reynolds number increases boundary layers are more obvious and the variations in the velocity profile become sharper.

$$\begin{cases} -\nu\nabla^2\mathbf{v} + (\mathbf{v}\cdot\nabla)\nabla p = \mathbf{b} & \text{in } \Omega \\ \nabla\cdot\mathbf{v} = 0 & \text{in } \Omega \\ \mathbf{v} = \mathbf{v}_D & \text{on } \Gamma_D \\ \mathbf{n}\cdot\sigma = \mathbf{t} & \text{on } \Gamma_N \end{cases} \tag{1.1}$$

Where:

- $\mathbf{v}$ - is the velocity.
- p - is the pressure.
- ν - is kinematic viscosity.
- $\mathbf{b}$ - is a source term.
- σ - is the Cauchy stress tensor.
- $\mathbf{t}$ - is the tractions vector.

The weak form of the Navier-Stokes equations 1.1 in reduced notation is:

$$\begin{cases} a(\mathbf{w},\mathbf{v}) + c(\mathbf{w},\mathbf{v},\mathbf{v}) - b(\mathbf{w},p) = (\mathbf{w},\mathbf{f}) \\ b(\mathbf{v},q) = 0 \end{cases} \tag{1.2}$$

With,

$$a(\mathbf{w},\mathbf{v}) = \int_\Omega (\nabla\mathbf{w}) : (\nu\nabla\mathbf{v})\,d\Omega \tag{1.3}$$

$$b(\mathbf{w},p) = \int_\Omega q\nabla\cdot\mathbf{v}\,d\Omega \tag{1.4}$$

$$(\mathbf{w},\mathbf{f}) = \int_\Omega \mathbf{w}\cdot\mathbf{f}\,d\Omega \tag{1.5}$$

Which corresponds to the previous implementation of Stokes problem, but now it should be considered the convective term, which it is needed to compute carefully to include in both iterative solver schemes (Picard and Newton-Raphson):

$$c(\mathbf{w},\mathbf{v},\mathbf{v}) = \int_\Omega \mathbf{w}\cdot(\mathbf{a}\cdot\nabla)\mathbf{v}\,d\Omega \tag{1.6}$$

The discretization of the above expressions yields a system of nonlinear equations:

$$\begin{pmatrix} \mathbf{K}+\mathbf{C}(\mathbf{v}) & \mathbf{G}^T \\ \mathbf{G} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{v} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix} \tag{1.7}$$

Then an iterative algorithm has to be used to solve the system of non-linear equations arising from the discretization of Navier-Stokes equations.

1.1 Picard Iterative Method

This methodology is applicable to non-linear systems that posses the form:

$$\mathbf{A}(\mathbf{x}^k)\mathbf{x}^{k+1} = \mathbf{b}(\mathbf{x}^k) \tag{1.8}$$

Initializing the process with a first guess $\mathbf{x}^{\mathbf{0}}$, then iteratively solving the system as:

$$\Delta\mathbf{x}^{k+1} = \mathbf{A}(\mathbf{x}^k)^{-1}\left(\mathbf{b}(\mathbf{x}^k) - \mathbf{A}(\mathbf{x}^k)\mathbf{x}^k\right) \tag{1.9}$$

Implementing this methodology to the set of equations 1.7 related to the Navier-Stokes problem, will be necessary to discretize each term and code each shape function and derivatives of shape functions. But as the class program contains this expressions already coded in the file *StokesSystem.m*, then a copy of this file with the name *ConvectiveMatrix.m* is done. This file will contain the computation of the convective term of the equation 1.6.

The discretization of the convective $\mathbf{C}(\mathbf{v})$ term can be written in compact form as:

$$C = [mat\ N]^T \begin{bmatrix} a_x & 0 & a_y & 0 \\ 0 & a_x & 0 & a_y \end{bmatrix} [grad\ N] \tag{1.10}$$

Code Implementation 1

The implementation inside a the file *ConvectionMatrix.m* is carried out by using the lines already coded (which are omitted in the next code) related to the computing of the shape functions and its derivatives for each Gauss point. Then, by considering the equation 1.17:

```
function [C] = ConvectionMatrix(X,T,referenceElement,velo)
for ielem = 1:nElem
  %Initializing the Ce Matrix
  Ce = zeros(nedofV,nedofV);
  for ig=1:ngaus
    v_igaus = N_ig*u_e; %Velocity from the BC's included in the function
    Ce = Ce + Ngp'*(v_igaus(1)*Nx+v_igaus(2)*Ny)*dvolu; % Convective term
  end
    C(Te_dof,Te_dof) = C(Te_dof,Te_dof) + Ce;
end
```

1.2 Newton-Raphson Method

Newton-Rapshon formulation attempts to reduce a value denominated residual less than a tolerance specified by the user. The residual is considered as the nonlinear system, for instance the Navier Stokes equations 1.7 can be transformed in a vector as:

$$\mathbf{r} = \begin{bmatrix} \Big(\mathbf{K} + \mathbf{C}(\mathbf{v})\Big)\mathbf{v} + \mathbf{G}^T\mathbf{p} - \mathbf{f} \\ \mathbf{Gv} \end{bmatrix} \tag{1.11}$$

Then, the iterative method starts with a initial guess $\mathbf{x^0}$. The iterations will continue until convergence is achieved, then the linear system of equations is:

$$\mathbf{J}(\mathbf{x}^k \Delta x^{k+1} = -\mathbf{r}(x^k) \tag{1.12}$$

Updating the solution

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}^{k+1} \tag{1.13}$$

This numerical methodology can performs better than Picard's method, but the main issue is the computation of the derivatives in order to complete the formulation of the Jacobian, which for Navier-Stokes equation has the general form:

$$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial \mathbf{r}_1}{\partial \mathbf{v}} & \dfrac{\partial \mathbf{r}_1}{\partial \mathbf{p}} \\ \dfrac{\partial \mathbf{r}_2}{\partial \mathbf{v}} & \dfrac{\partial \mathbf{r}_2}{\partial \mathbf{p}} \end{pmatrix} \tag{1.14}$$

Where for almost all the values is trivial:

$$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial \mathbf{r}_1}{\partial \mathbf{v}} & G^T \\ G & 0 \end{pmatrix}$$

Except for the $J(1,1)$ component, which can be expressed by the chain rule as:

$$\frac{\partial \mathbf{r}_1}{\partial \mathbf{v}} = \mathbf{K} + \mathbf{C}(\mathbf{v}) + \frac{\partial \mathbf{C}(\mathbf{v})}{\partial \mathbf{v}} \tag{1.15}$$

Thus, the above expression can be divided in two to computing in a simpler way. The first to terms corresponds to a variable that it will called $\mathbf{C}_1$ (which is the same term used with Picard), and the last term related to the derivative will be $\mathbf{C}_2$:

$$C_1 = [mat\ N]^T \begin{bmatrix} a_x & 0 & a_y & 0 \\ 0 & a_x & 0 & a_y \end{bmatrix} [grad\ N] \tag{1.16}$$

$$C_2 = [mat\ N]^T \begin{bmatrix} \frac{dV_x}{dx} & \frac{dV_x}{dy} \\ \frac{dV_y}{dx} & \frac{dV_y}{dy} \end{bmatrix} [mat\ N] \tag{1.17}$$

Code Implementation 2

In order to have the option of using Picard and Newton-Raphson method, a new file named *ConvectiveMatrixNR.m* is copied from the file *ConvectiveMatrix.m* used before in Picard. This file uses the same procedure to compute the shape functions depending the chosen element and its derivatives, then

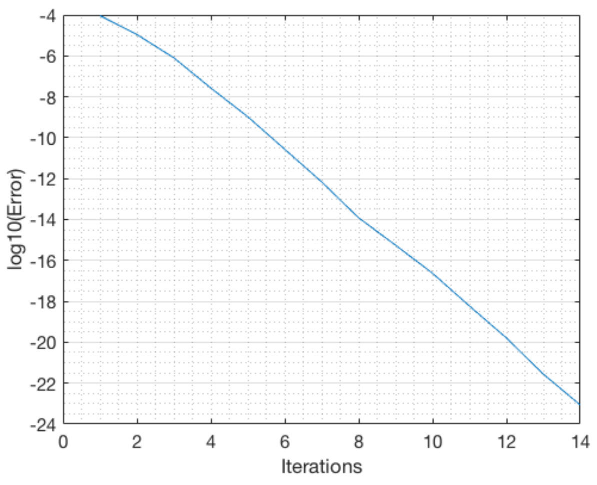
```
function [C1,C2] = ConvectionMatrixNR(X,T,referenceElement,velo)
for ielem = 1:nElem
  %Initializing the Ce Matrix
  Ce1 = zeros(nedofV,nedofV);
  Ce2 = zeros(nedofV,nedofV);
  for ig=1:ngaus
    v_igaus = N_ig*u_e; %Velocity from the BC's included in the function
    Ce1 = Ce1 + Ngp'*(v_igaus(1)*Nx+v_igaus(2)*Ny)*dvolu;
    Ce2 = Ce2 + Ngp'*([nx ; ny]*u_e)'*Ngp*dvolu;
  end
  C1(Te_dof,Te_dof) = C1(Te_dof,Te_dof) + Ce1;
  C2(Te_dof,Te_dof) = C2(Te_dof,Te_dof) + Ce2;
end
```

In the main file an option to choose the iterative method is implemented, then the corresponding modifications inside a while loop used before in the Picard method are (the repeated lines from the class code are omitted):

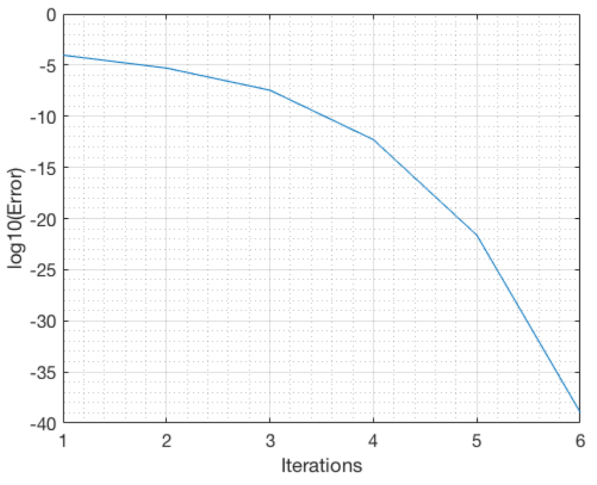
```
while iter < 100
    %Convective terms for Jacobian and System
    [C1,C2] = ConvectionMatrixNR(X,T,referenceElement,velo);
    Cred1 = C1(dofUnk,dofUnk);
    Cred2 = C2(dofUnk,dofUnk);

    %Navier-Stokes System
    A = [Kred + Cred1   Gred';
     Gred     zeros(nunkP)];
    Atot = A;
    btot = [fred - (C1(dofUnk,dofDir))*valDir; zeros(nunkP,1)];

    % Jacobian definition
    J = [Kred + Cred1 + Cred2   Gred';
     Gred     zeros(nunkP)];

    F = Atot*sol0 - btot; %Residual Function

    % Computation of velocity and pressure increment
    solInc =-J\F;

    % Update the solution (omitted)
     [...]
    % Check convergence lines (omitted)
     [...]
    % Update variables for next iteration (omitted)
     [...]
end
```

1.3 Cavity Flow Problem - Solver Scheme Comparison

In order to compare both iterative nonlinear solver schemes, the cavity flow problem is solved using a 10x10 Q2Q1 quadrilateral discretization in the domain for the velocity and pressure. In order to compare the behavior of both schemes a convergence comparison is shown in the figure 1.1. Using the tolerance error of $0.5e-08$, the same value is achieved. Newton-Raphson needs five iterations meanwhile Picard's method requires 13 iterations to converge. Moreover, it is noticeable the linear behavior of Picard, and the quadratic behavior of NR which helps to find the solution using less iterations, but it is needed to comment that computing the Jacobian can be a cost problem for larger systems.



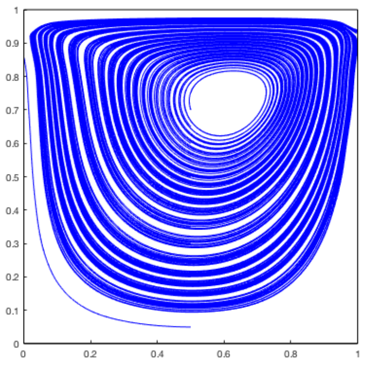
(a)



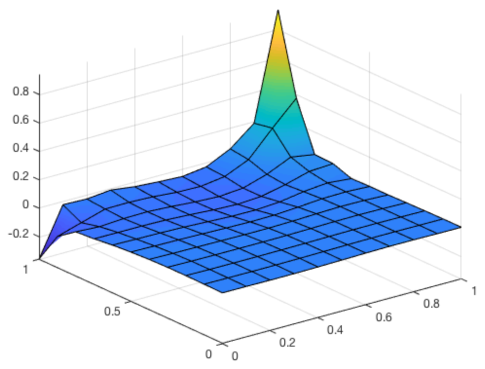
(b)

Figure 1.1: a) Picard and b) Newton-Rapshon convergence comparison.

Furthermore, both solutions are practically the same. The streamlines of velocity and the pressure distribution graph are similar and coherent with the expected results of a cavity flow problem.

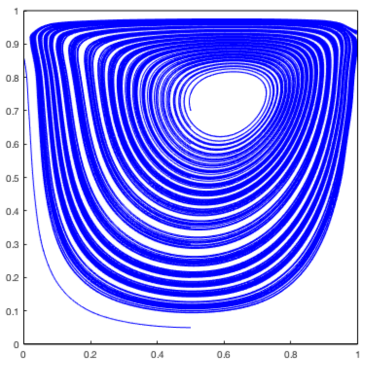


(a)

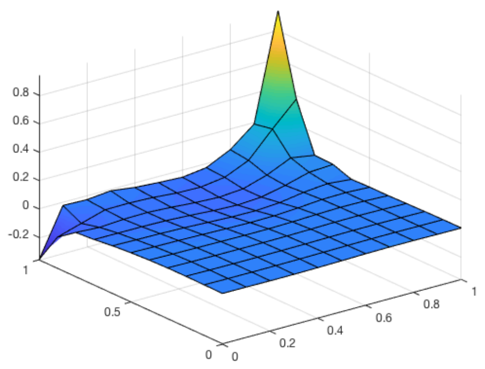


(b)

Figure 1.2: a) Velocity and b) pressure solutions using Picard Method.



(a)



(b)

Figure 1.3: a) Velocity and b) pressure solutions using Newton-Rapshon Method.