Pontifícia Universidade Católica
do Rio de Janeiro

Elisa Demori Lacombe Penna da Rocha

Determinação da permeabilidade absoluta em meios porosos através da Solução da Equação de Navier-Stokes na escala de poros

Projeto de Graduação

Projeto de Graduação apresentado ao Departamento de Engenharia Mecânica da PUC-Rio

Orientador: Márcio da Silveira Carvalho
Coorientador: Dr. Sergio Ribeiro

Rio de Janeiro

Novembro de 2020

AGRADECIMENTOS

Primeiramente, gostaria de agradecer aos meus pais, André e Claudia, por todo carinho, amor e suporte que proporcionaram durante minha trajetória.

Aos meus irmãos, Manoela, Bruna, Luiz Guilherme e Gabriela, por todo o companheirismo e amor.

E aos meus orientadores, Márcio Carvalho e Sergio Ribeiro, por todos os conselhos, toda ajuda e paciência com a qual me guiaram.

RESUMO

Determinação da permeabilidade absoluta em meios porosos através da Solução da Equação de Navier-Stokes na escala de poros

A análise do escoamento de fluidos Newtoniano e não-Newtonianos em meio porosos e a estimativa de sua permeabilidade absoluta sem o uso de técnicas experimentais ainda é um grande desafio para a engenharia. A dificuldade se dá devido à natureza complexa dos diferentes meios porosos existentes e a heterogeneidade do meio poroso. No presente trabalho realizou-se uma simulação numérica do escoamento no espaço poroso de um meio poroso, aproximado como um meio bidimensional, de fluidos Newtonianos e não-Newtonianos, através da solução das equações de conservação de massa e quantidade de movimento por elementos finitos. Uma ferramenta computacional foi criada em linguagem Python, com a utilização do pacote FEniCS, com a finalidade de realizar modelagens de escoamentos bidimensionais e analisar o comportamento de velocidade, pressão e os efeitos de viscosidade não-Newtoniana no escoamento. Os resultados obtidos de pressão e velocidade do escoamento Newtoniano foram utilizados para determinar a permeabilidade absoluta com a utilização da lei de Darcy para meios porosos.

Palavras-chave: Escoamento em meios porosos; Equações de Navier-Stokes; Permeabilidade Absoluta

ABSTRACT

Determination of absolute permeability in porous media using the Navier-Stokes Equation Solution on the pore scale

The analysis of the Newtonian and non-Newtonian flows in porous media and the estimation of the absolute permeability without running experiments is still a major challenge in engineering. The difficulty is due to the complex nature of the different existing porous media and the media matrices' heterogeneity of the pore space. In the present work, numerical simulation of the Newtonian and non-Newtonian pore scale flow through a porous media 2D model was performed by solving the mass and momentum conservation equations using the finite element method. A computational tool was created in python language, using the FEniCS package, to perform two-dimensional flow modeling and analyze the behavior of velocity, pressure, and the effect of non-Newtonian viscosity on the flow. The results obtained from pressure and velocity of the Newtonian flow were used to determine the absolute permeability with the use of Darcy's law for porous media.

Keywords: Flow in porous media; Navier-Stokes equations; absolute permeability

SUMÁRIO

Lista de figuras

Lista de tabelas

1 – Introdução

A determinação da permeabilidade de um meio poroso é de suma importância nos campos da engenharia, petrofísica, estudos ambientais e hidrogeologia. É um parâmetro altamente dependente de microestrutura complexa que quantifica a capacidade de um meio poroso de transmitir fluidos.

A permeabilidade de um meio poroso é usualmente determinada experimentalmente. Porém, os experimentos são longos e custosos. Idealmente, seria muito importante determinar o valor da permeabilidade de um meio sem a necessidade de análise experimental. Diferentes correlações são usadas para relacionar a permeabilidade com a porosidade de um meio. Porém, essas correlações apresentam um alto grau de imprecisão nos valores estimados.

O uso de imagens tridimensionais dos meios porosos facilitou a execução de modelos que representam o escoamento de fluidos e a geometria das rochas, tornando possível a obtenção de dados sem o uso de métodos experimentais tradicionais. As abordagens numéricas para o cálculo das propriedades dos espaços porosos podem ser classificadas em duas classes: simulação direta, através da solução das equações que governam escoamento através do espaço poroso, que é determinado através das imagens obtidas (geralmente por micro tomografia de raio X); e modelo de redes capilares, onde o espaço poroso é representado por uma rede de poros e gargantas de poros e as equações que descrevem o escoamento são simplificadas [1]. Entretanto, a extração de redes envolve ambiguidades, devido à dificuldade de diferenciar poros e gargantas nos diferentes algoritmos. Dessa forma, a forma mais precisa é determinar a permeabilidade através da solução da equação de Navier-Stokes no espaço poroso tridimensional, obtido por uma imagem [2].

O método Lattice Boltzmann é uma popular abordagem para a modelagem de escoamentos em geometrias complexas; porém é um alto custo computacional. No final do século XX e começo do século XXI alguns estudos compararam o método de diferenças finitas com o de Lattice Boltzmann. Os estudos mostraram que o método de Lattice Boltzmann requeria 2,5 mais memória comparada com a utilizada pelo método de diferenças finitas. Em 2012 pesquisadores usaram o método de diferenças finitas com imagens de micro tomografia para a estimativa da

permeabilidade de diferentes tipos de rochas e novamente compararam com os resultados com o método de Lattice Boltzmann. Esses indicavam que ambos os métodos apresentavam permeabilidade que diferiam pouco um do outro em menos de 6%, enquanto o método de diferenças finitas utilizou somente 17% do tempo de CPU utilizado pelo método de Lattice Boltzmann [2].

A primeira análise do uso do método de elementos finitos para a solução das equações de Navier-Stokes ocorreu na década de 70, introduzido por Babuška (1971) e Brezzi (1974). No entanto, a monografia estendida de Girault e Raviart (1986) tornou-se a obra de referência clássica no uso de elementos finitos para solução de Navier-Stokes [4].

O comportamento de fluidos não-Newtonianos é estudado na Reologia, área que estuda escoamento e deformações de materiais. Diferentemente dos fluidos Newtonianos, esses não apresentam uma relação linear entre tensão cisalhante e taxa de deformação, com isso, o valor da viscosidade não é constante e varia em função da taxa de deformação. Por não apresentarem uma viscosidade constante, o uso da equação de Darcy deve ser feito através de uma viscosidade característica, que represente o comportamento do fluido em todo o espaço poroso. Como existe um grande faixa de taxa de cisalhamento durante o escoamento de um fluido não-Newtoniano em um meio poroso, o valor de viscosidade característica não é simples de ser especificado. Simulação do escoamento de fluidos não-Newtonianos na escala de poros pode também auxiliar nessa estimativa.

O principal objetivo deste trabalho é desenvolver uma ferramenta para determinação da permeabilidade de um meio poroso através da solução da equação de Navier-Stokes em ambiente Python. A ferramenta será testada para determinar a permeabilidade de um meio poroso modelo bidimensional, utilizado em diferentes experimentos para análise de diferentes fenômenos na escala de poros e estudar o escoamento de fluidos não-Newtonianos através do espaço poroso. A ferramenta desenvolvida também foi ampliada para permitir o cálculo de escoamento não-Newtonianos, permitindo assim uma análise do efeito da variação da viscosidade no escoamento através de meios porosos.

2 – Formulação Numérica

Este capítulo apresenta a modelagem numérica do escoamento em um meio poroso para a determinação de sua permeabilidade absoluta. Com a solução das equações de Navier-Stokes pelo método de elementos finitos, através do pacote FEniCS, será possível determinar os campos de velocidade e pressão para então aplicar a Lei de Darcy para o cálculo da permeabilidade.

2.2 – Lei de Darcy

A lei de Darcy é uma relação empírica que descreve o escoamento de um fluido através de um meio poroso. Recebe esse nome devido ao seu formulador, o engenheiro francês Henry Darcy, que, em 1856, ao estudar o sistema de tratamento de água da cidade de Dijon, na França, analisou o escoamento de fluidos através de filtros de areia. Darcy notou que a vazão do escoamento era proporcional à área da seção transversal do duto, à diferença de pressão ao longo do fluxo e a uma constante k - denominada permeabilidade do meio poroso - e inversamente proporcional ao comprimento do duto [7].

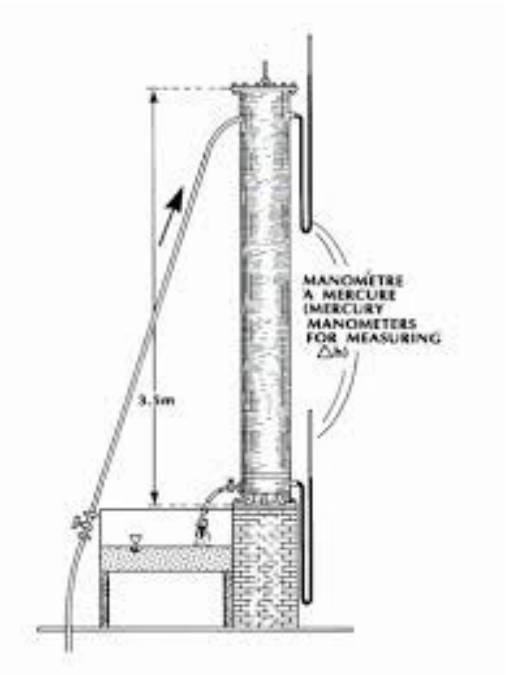


Figura 2.1: Fonte Pública da Cidade de Dijon usada em experimento de Darcy

Em forma vetorial, a Lei de Darcy é descrita como:

$$\nabla p = \mu k^{-1}\vec{u} + \vec{f}, \tag{2-1}$$

onde k é a permeabilidade do meio, u é o vetor velocidade, μ é a viscosidade do fluido, p é a pressão do fluido e f é uma força de corpo. Como dito anteriormente, o parâmetro k deve ser determinado e depende diretamente da estrutura e geometria do espaço poroso do meio.

2.3 - Equações de Navier-Stokes

As equações de Navier-Stokes são equações diferenciais que descrevem o comportamento de fluidos incompressíveis baseadas na equação de conservação de quantidade de movimento linear (2-2a) e a equação de conservação de massa (2-2b). Essas formam um sistema de equações para a velocidade u e pressão p.

$$\rho\left(\frac{\partial\vec{u}}{\partial t} + \vec{u}\cdot\nabla\vec{u}\right) = \nabla\cdot\sigma(\vec{u},p) + \vec{f}, \quad \forall\, x\,\epsilon\,\Omega, t > 0 \tag{2-2a}$$

$$\nabla\cdot\vec{u} = 0, \quad \forall\, x\,\epsilon\,\Omega, t > 0 \tag{2-2b}$$

sendo f uma determinada força aplicada ao fluido, ρ a densidade do fluido e $\sigma(\vec{u},p)$ o tensor de tensão, que para um fluido Newtoniano é ser definido por:

$$\sigma(\vec{u},p) = 2\eta\epsilon(\vec{u}) - pI \tag{2-3}$$

onde μ é a viscosidade do fluido, I é a matriz identidade e $\epsilon(\vec{u})$ é o tensor taxa de deformação dado por:

$$\epsilon(\vec{u}) = \frac{1}{2}(\nabla\vec{u} + (\nabla\vec{u})^T) \tag{2-4}$$

Normalmente não é possível encontrar uma solução analítica para as Equações de Navier-Stokes, o que torna necessário o uso de métodos numéricos para uma solução aproximada.

2.4 – Reologia dos Fluidos

A reologia é o estudo do escoamento e deformação da matéria; ou seja, é o ramo que analisa viscosidade, plasticidade, elasticidade e escoamento.

A viscosidade de um fluido pode ser definida como a resistência interna, ou fricção interna, a um escoamento de cisalhamento. Sendo assim, quanto maior sua viscosidade, maior potência de bombeio é necessária para seu escoamento.

Os fluidos independentes do tempo podem ser classificados em duas classes:

- *Newtonianos:* Fluidos que obedecem à Lei de Newton; ou seja, o gradiente de velocidade é linearmente proporcional a tensão de cisalhamento aplicada sobre ele. Sua viscosidade é constante e não varia com a taxa de cisalhamento.
- *Não-Newtonianos:* Fluidos que não seguem a Lei de Newton da viscosidade. A tensão não varia linearmente com a taxa de deformação. Isso equivale a dizer que a viscosidade varia com a taxa de cisalhamento.
    - *Pseudoplásticos:* A viscosidade tende a diminuir com o aumento da taxa de cisalhamento.
    - *Dilatantes:* Apresentam um aumento de viscosidade com a taxa de cisalhamento.

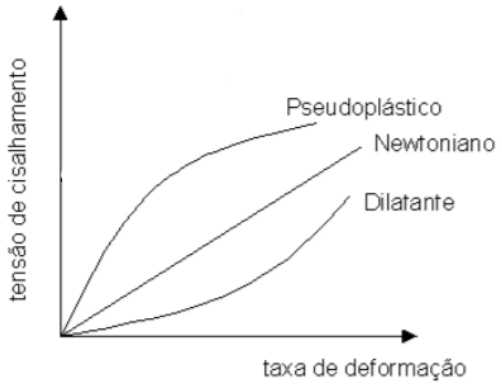


Figura 2.2: Curva de escoamento de fluidos independentes do tempo e sem tensão cisalhante inicial.

O comportamento da curva de viscosidade pode ser descrito por diferentes modelos matemáticos. O modelo de power-law é o mais simples e um dos mais utilizados para a determinação da viscosidade de fluidos e pode ser descrito pela seguinte equação:

$$\eta(\dot{\gamma}) = K|\dot{\gamma}|^{n-1} \tag{2-5}$$

sendo K o índice de consistência, $\dot{\gamma}$ a taxa de deformação e n o índice de power-law.

Diferentes valores de n representam diferentes comportamentos:

- $n = 1$: fluido Newtoniano com viscosidade K (Pa.s);
- $n > 1$: fluido dilatante; e
- $n < 1$: fluido pseudoplástico.

Entretanto, esse modelo não descreve o comportamento real dos fluidos nos limites das taxas de cisalhamento, pois quando as taxas de cisalhamento são muito altas, a viscosidade tenderá para zero (fluido pseudoplástico) ou infinito (fluido dilatante), e em taxas muito pequenas tenderá para valor nulo (fluido dilatante) ou infinito (fluido pseudoplástico).

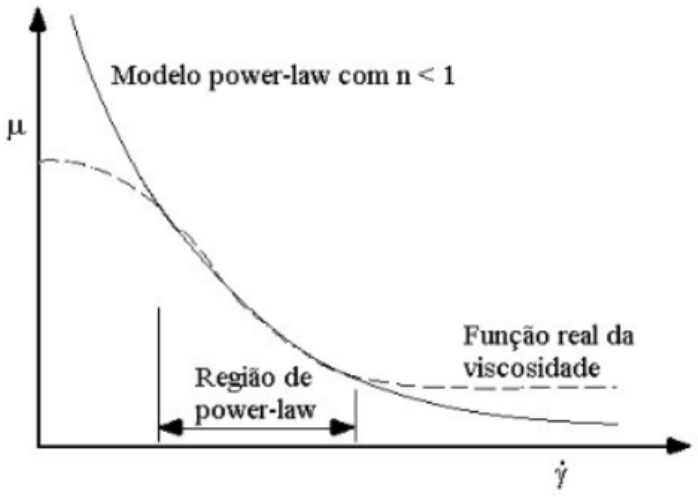


Figura 2.3: Curva viscosidade-taxa de deformação para o modelo de power-law com índice menor que 1.

2.4 – Elementos Finitos

O desenvolvimento do Método de Elementos Finitos (FEM – *Finite Element Method*) é datado de 1909, por Walter-Ritz, para a determinação das soluções de problemas mecânicos de sólidos deformáveis. Entretanto somente no final dos anos 1960 que o Método passou a ser utilizado em simulação de problemas não-estruturais, sendo estes problemas relacionados a fluidos, eletromagnetismo e eletromecânica. Nos dias de hoje, tem se mostrado um ferramenta poderosa para a solução computacional de diferentes problemas nas áreas de engenharia [6].

O procedimento numérico consiste na subdivisão do domínio do problema em regiões menores, chamadas de elementos finitos. Os campos desconhecidos são descritos como uma combinação linear de funções bases pré-escolhidas. As constantes da expansão representam as incógnitas do problema.

2.5 – Formulação Variacional

No Método de Elementos Finitos, a equação diferencial é escrita em sua forma fraca.

Como visto na seção 2.3 as equações de Navier-Stokes são descritas por:

$$\rho\left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla\vec{u}\right) = \nabla \cdot (2\eta\epsilon(\vec{u}) - pI) + \vec{f}, \quad \forall\, x \,\epsilon\, \Omega, t > 0 \tag{2-6a}$$

$$\nabla \cdot \vec{u} = 0, \quad \forall\, x \,\epsilon\, \Omega, t > 0 \tag{2-6b}$$

Primeiramente é necessário impor condições de contorno ao modelo matemático. Essas são definidas por:

$$\vec{u} \cdot \hat{n} = u_0, \quad \text{em } \Gamma_D \tag{2-7a}$$

$$p = p_{in} \qquad \text{em } \Gamma_{p_{in}} \tag{2-7b}$$

$$p = p_{out} \qquad \text{em } \Gamma_{p_{out}} \tag{2-7b}$$

Sendo $\Gamma_{p_{in}}$ e $\Gamma_{p_{out}}$ as fronteiras, onde p_{in} é a pressão de entrada especificada e p_{out} a pressão de saída especificada, Γ_D o contorno de Dirichlet e $\hat{n}$ o vetor normal ao contorno de Dirichlet.

Após definidas as condições de contorno, deve-se multiplicar os dois lados da equação referente a conservação de quantidade de movimento linear por uma função teste $\vec{v}$ e integrá-las no domínio Ω.

$$\int_{\Omega} \rho\left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla\vec{u}\right) \cdot \vec{v}\, d\vec{x} = \int_{\Omega} \left(\nabla \cdot (2\eta\epsilon(\vec{u}) - pI) + \vec{f}\right) \cdot \vec{v}\, d\vec{x} \tag{2-8}$$

Que pode ser reescrita como:

$$\int_{\Omega} \rho\left(\frac{\partial \vec{u}}{\partial t}\right) \cdot \vec{v}\, dx + \int_{\Omega} \rho(\vec{u} \cdot \nabla\vec{u}) \cdot \vec{v}\, d\vec{x} = \int_{\Omega} \left(\nabla \cdot (2\eta\epsilon(\vec{u}) - pI)\right) \cdot \vec{v}\, d\vec{x} + \int_{\Omega} \vec{f} \cdot \vec{v}\, d\vec{x} \tag{2-9}$$

No entanto, algumas modificações são necessárias no lado esquerdo da equação. É preciso fazer uma integração por partes no primeiro termo:

$$-\int_{\Omega} [(2\eta\epsilon(\vec{u}) - pI) \cdot \epsilon(\vec{v})] d\vec{x} + \oint_{\delta\Omega} [(2\eta\epsilon(\vec{u}) - pI) \cdot \vec{v}] \cdot \hat{n}\, ds \tag{2-10}$$

Após aplicar a integração, tem-se a eliminação do gradiente de pressão que leva ao estado natural de pressão nas condições de contorno, gerando o seguinte resultado:

$$-\int_{\Omega} [\sigma(\vec{u}, p) \cdot \epsilon(\vec{v})] d\vec{x} - \oint_{\delta\Omega} p_{out}(\vec{v} \cdot \hat{n}) ds - \oint_{\delta\Omega} p_{in}(\vec{v} \cdot \hat{n}) ds \tag{2-11}$$

Analogamente o mesmo deve ser feito na equação de conservação de massa (2-8b), porém multiplicada por uma função teste escalar q.

$$\int_{\Omega} q(\nabla \cdot \vec{u}) d\vec{x} = 0 \tag{2-12}$$

Com isso, a formulação Fraca das Equações de Navier-Stokes é obtida:

$$a_1(\vec{u}, \vec{v}) = L(\vec{v}) \tag{2-13a}$$

$$a_2(\vec{u}, q) = 0 \tag{2-13b}$$

Sendo os valores de a_1, L e a_2:

$$a_1(\vec{u}, \vec{v}) = \int_{\Omega} \rho\left(\frac{\partial \vec{u}}{\partial t}\right) \cdot \vec{v}\, d\vec{x} + \int_{\Omega} \rho(\vec{u} \cdot \nabla\vec{u}) \cdot \vec{v}\, d\vec{x} + \int_{\Omega} [\sigma(\vec{u}, p) \cdot \epsilon(\vec{v})] d\vec{x} \tag{2-14a}$$

$$L(\vec{v}) = -\oint_{\delta\Omega} p_{out}(\vec{v}\cdot\hat{n})ds \; - \oint_{\delta\Omega} p_{in}(\vec{v}\cdot\hat{n})ds \tag{2-14b}$$

$$a_2(\vec{u},q) \; = \int_{\Omega} q(\nabla\cdot\vec{u})d\vec{x} \tag{2-14c}$$

3 – Desenvolvimento da Ferramenta Computacional

3.1 – Projeto FEniCS

O Projeto FEniCS é um acervo de softwares livres que tem como principal objetivo resolver equações diferenciais parciais, permitindo que os usuários traduzam modelos científicos em códigos de elementos finitos.

A biblioteca DOLFIN, que fornece algoritmos e estruturas de dados que permitem a leitura e compilação do programa, é a principal interface de usuário do FEniCS. Os principais componentes da biblioteca são:

- UFL (*Unified Form Language*): Uma linguagem embutida em Python com o objetivo de discretizar de elementos finitos de equações diferenciais em relação a sua forma variacional;
- FIAT (*Finite Element Automatic Tabulator*): Um módulo dedicado a geração de funções de base de elementos finitos;
- UFC (*Unified Form-assembly Code*): Interface baseada na linguagem C++. Esse componente tem a finalidade de resolver as formas variacionais dos elementos finitos;
- FFC (*FEniCS Form Compiler*): Um compilador de formas variacionais que recebe o código UFL como entrada e gera uma saída UFC.

3.2 – Implementação da Ferramenta Computacional

Primeiramente deve-se importar as bibliotecas necessárias para o desenvolvimento do programa. Como dito anteriormente a biblioteca DOLFIN é a interface principal do pacote FEniCS e é a responsável pelas principais funções usadas para a solução das equações de Navier-Stokes por elementos finitos.

Python code

```
import matplotlib.pyplot as plt
```

```
import numpy as np
from dolfin import *
import os
```

Após importada as bibliotecas necessárias a implementação segue com o carregamento dos subdomínios referentes as paredes da malha, com a utilização da função *readDomains* previamente definida, e o uso da função *MeshFunction*, do pacote FEniCS, que é usada para marcação dos subdomínios e dos contornos da malha.

Python code

```
# Mesh Reading
# Load Subdomains
Subdomains = readDomains(meshPath,meshFile)
print(Subdomains)

# Option 2 - Gmsh generation and XML Conversion
meshObj = Mesh(meshPath+meshFile +'.xml')
# Initialize boundaries (inlet, outlet, etc...)
Boundaries = MeshFunction('size_t',meshObj,meshPath+meshFile +
"_facet_region.xml")
# Initialize subdomain (fluid)
markers = MeshFunction('size_t',meshObj,meshPath+meshFile +
'_physical_region.xml')
```

Um espaço de função é criado a partir dos elementos finitos referente a velocidade e pressão. Para uma melhor estabilização do método, um espaço misto de funções Lagrangeanas foi utilizado, esse consiste em um elemento linear para a solução de pressão (3 graus de liberdade) e um elemento quadrático para a solução do campo de velocidade (6 graus de liberdade), como mostrado na figura 3.1.

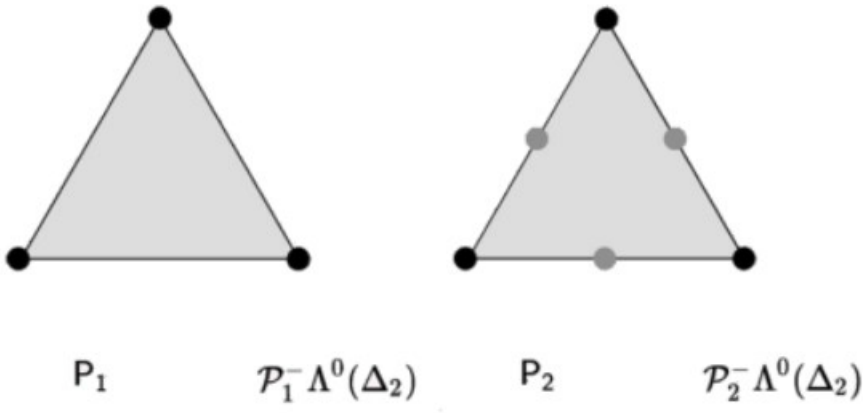


Figura 3.1: Elementos Finitos

Python code

```
elementShape = meshObj.ufl_cell()
# Set Mesh Elements
Uel = VectorElement(velocityElementfamily,elementShape, velocityElementOrder) #
Velocity vector field
Pel = FiniteElement(pressureElementfamily, elementShape, pressureElementOrder) #
Pressure field
Nel = FiniteElement(pressureElementfamily, elementShape, pressureElementOrder) #
Viscosity field
UPel = MixedElement([Uel,Pel])
# Function Spaces: Flow
W = FunctionSpace(meshObj,UPel)
C = FunctionSpace(meshObj,Nel)

W = Function(W)
c0 = Function(C)
```

Através dos espaços funcionais as funções base e funções testes podem ser criadas. Do espaço funcional misto de pressão e velocidade é definida uma função de elemento finito *w* que será dividida para obter separadamente velocidade e pressão.

Python code

```
## Trial and Test function(s)
dw = TrialFunction(W)
(v, q) = TestFunctions(W)
w = Function(W)
```

```
# Split into Velocity and Pressure
(u, p) = (as_vector((w[0], w[1])), w[2])
(U, P) = W.split()
```

As equações de Navier-Stokes são então transpostas para o problema variacional e acopladas para a formulação fraca completa.

Python code

```
# Bilinear Form
# Linear Momentum Conservation: F = a - L
          # Inertia Term                    # Viscous Forces Term
a1 = (rho*dot(dot(u,grad(u) ) ,v) + inner(TT(u,p,mu),DD(v)))*dx()

        # Inlet Pressure
L1 = - (Pin)*dot(n,v)*ds(Subdomains['Inlet'])
         # Outlet Pressure
(Pout)*dot(n,v)*ds(Subdomains['Outlet'])

# Mass Conservation(Continuity)
a2 = (q*div(u))*dx()
L2 = 0
# Complete Weak Form
F = (a1 + a2) - (L1 + L2)
# Jacobian Matrix
J = derivative(F,w,dw)
```

As condições de contorno de Dirichlet são definidas nas fronteiras do domínio superior, inferior e contornos essenciais internos. E, por fim, as equações são solucionadas.

Python code

```
# Apply Flow Boundary Conditions
bcU1 =
DirichletBC(W.sub(0),Constant((0.0,0.0)),boundaries,Subdomains['BottomWall
'])
bcU2 =
DirichletBC(W.sub(0),Constant((0.0,0.0)),boundaries,Subdomains['TopWall'])
bcU3 =
DirichletBC(W.sub(0),Constant((0.0,0.0)),boundaries,Subdomains['InnerWalls
```

```
'])
bcU = [bcU1,bcU2,bcU3]

##########   Numerical Solver Properties
# Problem and Solver definitions
problemU = NonlinearVariationalProblem(F,w,bcU,J)
solverU = NonlinearVariationalSolver(problemU)
# # Solver Parameters
prmU = solverU.parameters
prmU['nonlinear_solver'] = 'newton'
prmU['newton_solver']['linear_solver'] = linearSolver

# Solve Problem
(no_iterations,converged) = solverU.solve()
```

Para o caso de escoamento de fluidos não-Newtonianos, como a viscosidade não é constante, um procedimento iterativo deve ser utilizado. O chute inicial desse procedimento considera um campo de velocidade e pressão obtido da solução do problema para um fluido Newtoniano.

Após a solução do problema para a viscosidade constante, que será considerado o chute inicial do problema, a formulação variacional entrará junto com a solução do problema em loop de repetição com a finalidade de convergir o campo de velocidade. A repetição ocorrerá até o momento em que a diferença de velocidades do passo anterior com o passo atual seja menor que uma tolerância imposta.

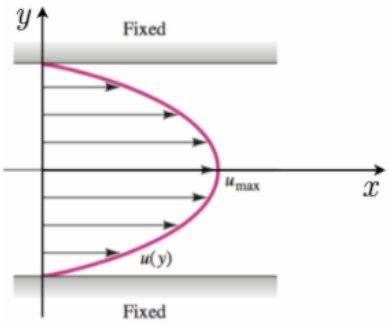
Python code

```
k=0
while err > eps:
    ETA = eta(u0)
    # Linear Momentum Conservation: F = a - L
              # Inertia Term                      # Viscous Forces Term
    a1 = (rho*dot(dot(u,grad(u)),v) + inner(TT(u,p,eta(u0)),DD(v)))*dx()

            # Inlet Pressure
    L1 = - (Pin)*dot(n,v)*ds(Subdomains['Inlet']) –
```

```
                # Outlet Pressure
    (Pout)*dot(n,v)*ds(Subdomains['Outlet'])

    # Mass Conservation (Continuity)
    a2 = (q*div(u))*dx()
    L2 = 0

    # Complete Weak Form
    F = (a1 + a2) - (L1 + L2)
    # Jacobian Matrix
    J = derivative(F,w,dw)

    ##########   Numerical Solver Properties
    # Problem and Solver definitions
    problemU = NonlinearVariationalProblem(F,w,bcs,J)
    solverU = NonlinearVariationalSolver(problemU)
    # # Solver Parameters
    prmU = solverU.parameters
    prmU['nonlinear_solver'] = 'newton'
    prmU['newton_solver']['linear_solver'] = linearSolver
    # Solve Problem
    (no_iterations,converged) = solverU.solve()
    w0.assign(w1); u0,p0 = w0.split()
    w1.assign(w); u1,p1 = w1.split()

    k+=1
    print(k)
    err = abs(norm(u0)-norm(u1))/norm(u0)
print(err)
```

4 – Resultados

Inicialmente, é interessante validar o programa através da comparação do perfil de velocidade da solução numérica com o perfil de velocidade analítico do escoamento de Poiseuille entre placas paralelas.



4.1: Perfil de Velocidade entre duas placas paralelas

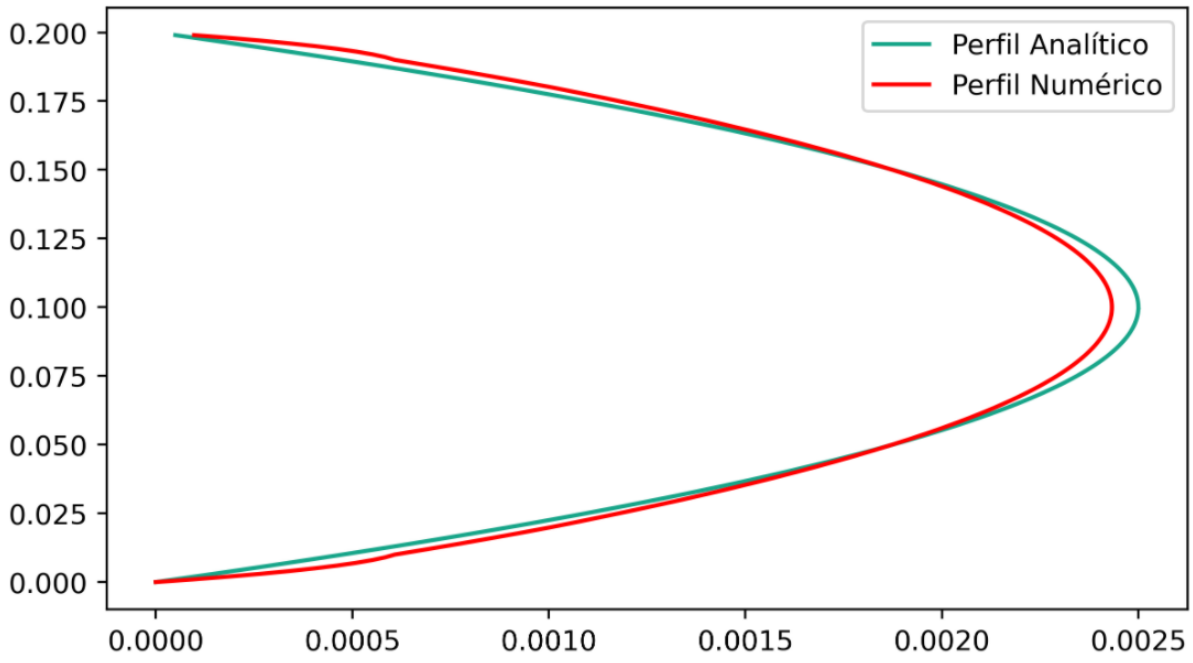


Figura 4.2: Comparação entre perfil de velocidade da solução analítica e solução numérica

Com a finalidade de melhor entender o escoamento no meio poros, o campo de pressão calculado no escoamento Newtoniano é apresentado na figura 4.3. Como esperado, o fluido escoa da esquerda para direita na figura, escoamento da região de alta pressão para baixa pressão.

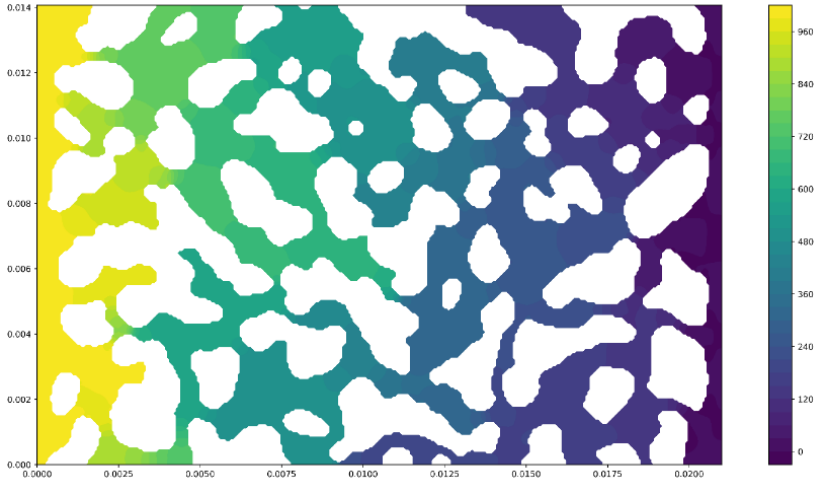


Figura 4.3: Distribuição de Pressão.

O próximo mapa é o campo de velocidade no meio. É possível observar o caminho obtido, onde a velocidade é maior do que no resto do domínio. Esse caminho preferencial corresponde ao caminho de menor resistência ao escoamento.

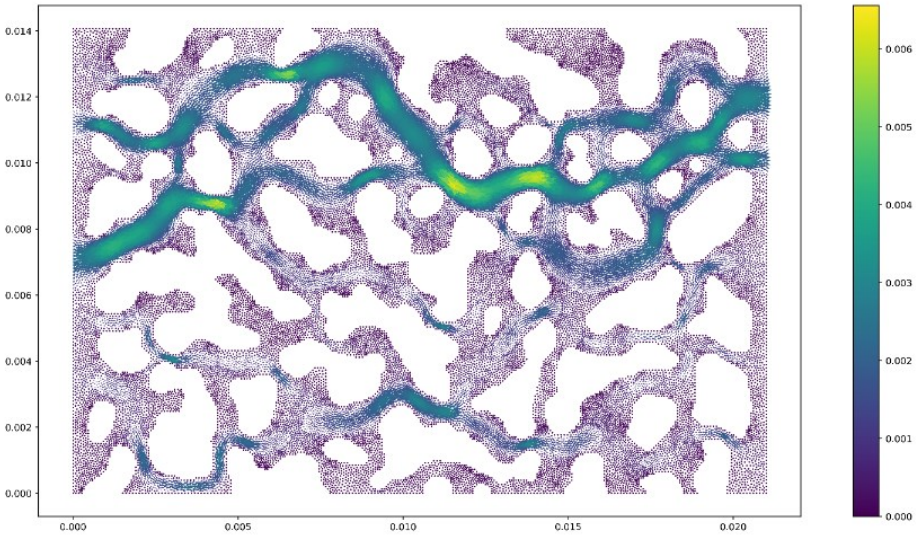


Figura 4.4: Campo de Velocidade

Para o Cálculo da Permeabilidade absoluta do meio poroso foram calculados valores da velocidade de Darcy para diferentes variações de Pressão. Sendo mantidos os dados de entrada do fluido, a viscosidade do fluido μ = 1,00 [Pa.s] e a densidade ρ = 1000 [kg/m³].

Variação de Pressão	Velocidade de Darcy ($x10^{-4}$)
1000	3,967
800	3,174
600	2,380
400	1,587
200	0,793
100	0,397
80	0,317
60	0,238
40	0,159
20	0,079
0	0,000

Tabela 4.1: Velocidade de Darcy em função da Variação de Pressão

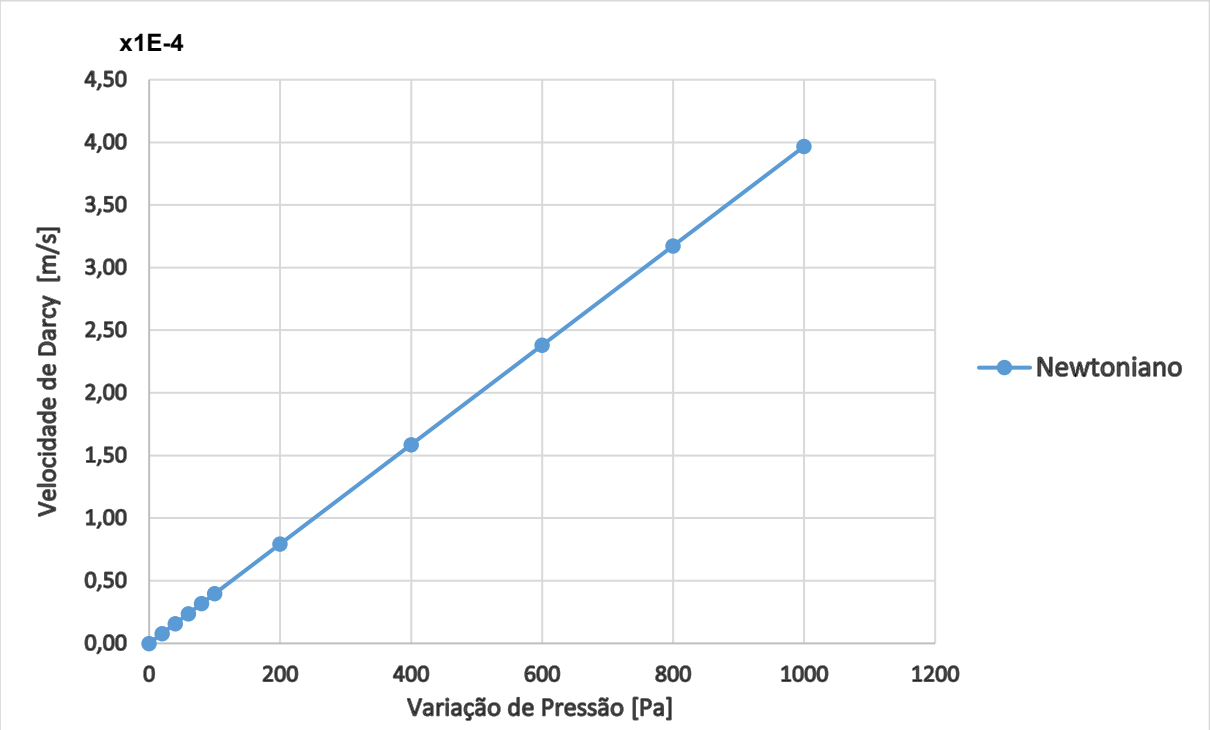


Figura 4.5: Gráfico da velocidade de Darcy em função da variação de Pressão

Pela Lei de Darcy sabe-se que o cálculo da velocidade de Darcy é Definido por:

$$v = \frac{k}{\mu L}\,\Delta p \tag{5-1}$$

Percebe-se que representa uma reta com coeficiente angular em função da permeabilidade k e o comprimento L do meio, e da viscosidade μ do fluido:

$$\alpha = \frac{k}{\mu L} \tag{5-2}$$

ou seja, calcula-se a permeabilidade absoluta do meio poroso através do coeficiente angular do gráfico da figura 4.5.

α	$5{,}55 \times 10^{-9}$	m/Pa.s
L	0,02	m
$\boldsymbol{k}$	**112,56**	**D**

Tabela 4.2: Cálculo da Permeabilidade absoluta

onde D representa Darcy e $1D \cong 9{,}869233 \times 10^{-13}\, m^2$.

Porém, o mesmo não pode ser feito para os fluidos não-Newtonianos, pois estes não apresentam viscosidade constante no escoamento. Isso ocorre porque, como visto anteriormente, a viscosidade dos fluidos não-Newtonianos depende da taxa de cisalhamento e estas variam dentro no meio poroso. A figura 4.6 deixa claro a variação da viscosidade do fluido dilatante dentro do meio poroso. Observa-se que a viscosidade é maior para a maior diferença de pressão, já que para um fluido dilatante, a viscosidade aumenta com a taxa de cisalhamento.

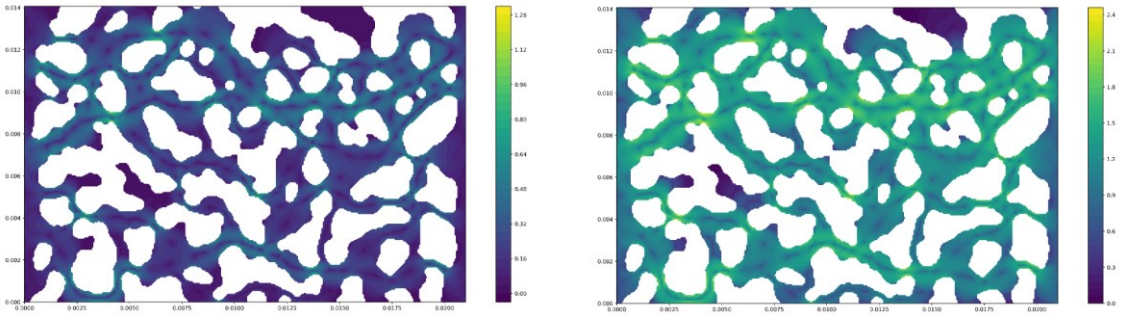


Figura 4.6: Mapa da viscosidade do fluido dilatante para variação de pressão de 20 Pa e 100 Pa, respectivamente.

A figura 4.7 permite ver que a variação da velocidade de Darcy não é constante em função da diferença de pressão para fluidos não-Newtonianos, isso é o mesmo que dizer que sua curva não é linear em função da variação de pressão, e, consequentemente, não é possível definir a permeabilidade absoluta por esses fluidos através da Lei de Darcy, pois estes não apresentam um coeficiente angular.

Esses resultados podem ser utilizados para determinar a viscosidade característica do fluido no escoamento.

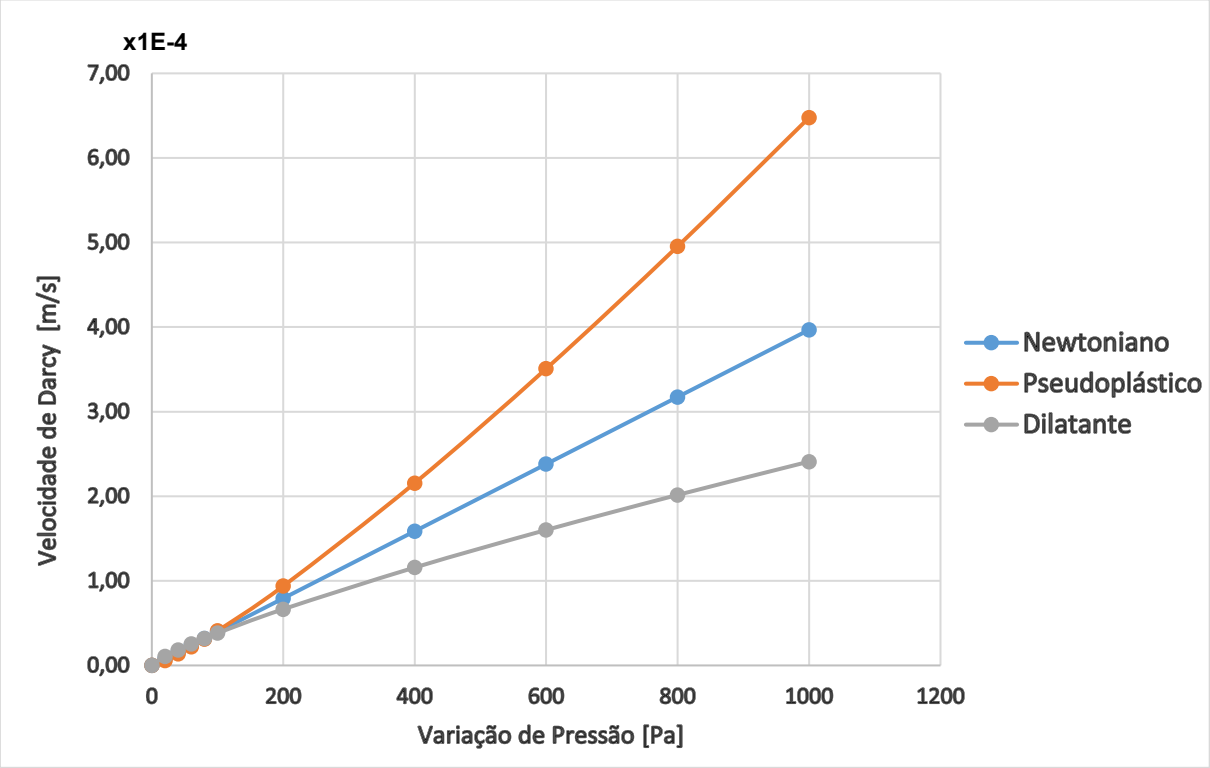


Figura 4.7: Comparação entre curvas de velocidade de Darcy para fluidos não-Newtoniano e Newtoniano

5 – Conclusão

Neste trabalho foi implementada uma ferramenta computacional com intuito de descrever um escoamento de um fluido Newtoniano ou não-Newtoniano em meios porosos. Com a finalidade de um estudo simplificado, o estudo foi realizado em um meio poroso modelo bidimensional.

No entanto, o programa foi capaz de aplicar uma metodologia de cálculo numérica para a solução das equações de Navier-Stokes que simula o escoamento generalizado de um fluido em diferentes meios, sendo estes estruturas complexas como uma malha porosa e escoamento simples entre placas paralelas, e permitir o cálculo da permeabilidade absoluta de um meio poroso aplicando-se lei de Darcy com os valores obtidos, para o fluido Newtoniano, de pressão e velocidade de Darcy.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Martin J. Blunt, Branko Bijeljic, Hu Dong, Oussama Gharbi, Stefan Iglauer, Peyman Mostaghimi, Adriana Paluszny, Christopher Pentland. *Advances in Water Resoources: Pore-Scale imaging and modelling.* ELSEVIER, 2012.

[2] Peyman Mostaghimi, Martin J. Blunt, Branko Bijeljic. *Computations of Absolute Permeability on Micro-CT Images.* Springer, 2012.

[3] Anders Logg, Kent-Andre Mardal, Garth N. Wells. *Automated Solution of Differential Equations by the Finite Element: The FEniCS Book.* Springer, 2012.

[4] VOLKER, JOHN. *Finite Element Methods for Incompressible Flow Problems.* Springer, 2016.

[5] Howard A. Barnes. *A Handbook of elementary Rheology*. Institute of Non-Newtonian Fluid Mechanics, University of Wales, 2000.

[6] CAMPOS, MARCO. *O Método de Elementos Finitos Aplicado à Simulação Numérica de Escoamentos de Fluidos*. IME-UFG, 2006.

[7] G. O. Brown. *Henry Darcy and the making of a law.* Department of Biosystems and Agricultural Engineering, Oklahoma State University, 2002.

[8] CARVALHO, PABLO. *Modelagem Computacional do escoamento de Stokes e estudo comparativo entre espaços de aproximação.* Faculdade de Engenharia Civil, Arquitetura e Urbanismo, Universidade Estadual de Campinas - UNICAMP, 2017.

A – Script

Python code

```
1   # Libraries Import
2   import matplotlib.pyplot as plt
3   import numpy as np
4   from dolfin import *
5   import os
6
7   # Helping self defined Functions
8   # Read subdomains from .msh file
9   def readDomains(inPath,inFile):
10      # Read .msh File
11      fid = open(inPath+inFile+'.msh', 'r')
12      # Initialize variables
13      found = 0
14      finished = 0
15      physicalNames = {}
16      # Loop througn .msh lines
17      for line in fid:
18          if '$EndPhysicalNames' in line:
19              finished == 1
20              break
21          elif '$PhysicalNames' in line:
22              found = 1
23          elif found==1 and finished == 0:
24              word=line.split()
25              if len(word)==3:
26                  physicalNames[word[2][1:len(word[2])-1]] =
27  int(word[1])
28      return physicalNames
29  # Convert .msh file to .xml files
30  def msh2xml(inPath,inFile,outPath,outFile):
31      cmd = 'dolfin-convert '+inPath+inFile+'.msh
32  '+outPath+outFile+'.xml'
33      print(cmd)
34      os.system(cmd)
```

```
35  # Deformation Tensor
36  def DD(u):
37      D = 0.5*(nabla_grad(u) + nabla_grad(u).T)
38      return D
39
40  # Stress Tensor
41  def TT(u, p, mu):
42      T = 2*mu*DD(u) - p*Identity(len(u))
43      return T
44
45  def gamma(u):
46      return pow(2*inner(DD(u),DD(u)),0.5)
47
48  def eta(u):
49      return kappa*pow(gamma(u),n_exp-1)
50
51  # Inputs
52  # Mesh File
53  meshPath = './'
54  meshFile = 'SimplerImageOKFlipped'
55
56  # Timestep
57  dt = 0.1
58
59  # Pressure Difference
60  Pin = 20
61  Pout = 0
62
63  # Fluid Properties
64  rho = 1000
65  mu = 1.0
66  alpha = 0.9
67  kappa = 1.0
68  n_exp = 1.5
69  # Mesh Elements
70  # Velocity
```

```
71  velocityElementfamily = 'Lagrange'
72  velocityElementOrder = 2
73  # Pressure
74  pressureElementfamily = 'Lagrange'
75  pressureElementOrder = 1
76  msh2xml(meshPath,meshFile,meshPath,meshFile)
78
79  # Solver Parameters
80  absTol = 1e-8
81  relTol = 1e-9
82  maxIter = 15
83  linearSolver = 'mumps'
84
85  # Mesh Reading
86  # Load Subdomains
87  Subdomains = readDomains(meshPath,meshFile)
88  print(Subdomains)
89
90  # Option 2 - Gmsh generation and XML Conversion
91  meshObj = Mesh(meshPath+meshFile +'.xml')
92  # Initialize boundaries (inlet, outlet, etc...)
93  boundaries = MeshFunction('size_t',meshObj,meshPath+meshFile +
94  "_facet_region.xml")
95  # Initialize subdomain (fluid)
96  markers = MeshFunction('size_t',meshObj,meshPath+meshFile +
97  '_physical_region.xml')
98  # Open new figure
99  plt.figure(figsize=(20, 10), dpi=400, facecolor='w', edgecolor='k')
100 # plot Mesh and save image
101 plot(meshObj)
102 plt.savefig('Mesh.png')
103
104 # Get Element Shape: Triangle, etc...
105 elementShape = meshObj.ufl_cell()
106
107
```

```
# Set Mesh Elements
Uel = VectorElement(velocityElementfamily, elementShape,
velocityElementOrder) # Velocity vector field
Pel = FiniteElement(pressureElementfamily, elementShape,
pressureElementOrder) # Pressure field
Nel = FiniteElement(pressureElementfamily, elementShape,
pressureElementOrder) # Viscosity field
UPel = MixedElement([Uel,Pel])

# Define any measure associated with domain and subdomains
dx = Measure('dx', domain=meshObj)
ds = Measure('ds', domain=meshObj, subdomain_data=boundaries)

# Vectors Normal to the Mesh
n = FacetNormal(meshObj) # Normal vector to mesh

# Function Spaces: Flow
# Mixed Function Space: Pressure and Velocity
W = FunctionSpace(meshObj,UPel)

C = FunctionSpace(meshObj,Nel)

w0 = Function(W)
c0 = Function(C)

##### Functions
## Trial and Test function(s)
dw = TrialFunction(W)
(v, q) = TestFunctions(W)
w = Function(W)

# Split into Velocity and Pressure
(u, p) = (as_vector((w[0], w[1])), w[2])

# Time step Constant
Dt = Constant(dt)
```

```
144
145 # Apply Flow Boundary Conditions
146 bcU1  =
147 DirichletBC(W.sub(0),Constant((0.0,0.0)),boundaries,Subdomains['Bott
148 omWall'])
149 bcU2 =
150 DirichletBC(W.sub(0),Constant((0.0,0.0)),boundaries,Subdomains['TopW
151 all'])
152 bcU3 =
153 DirichletBC(W.sub(0),Constant((0.0,0.0)),boundaries,Subdomains['Inne
154 rWalls'])
155 bcs = [bcU1,bcU2,bcU3]
156
157 w1 = Function(W)
158 w_mid = Function(W)
159 (u_mid, p_mid) = (as_vector((w_mid[0], w_mid[1])), w_mid[2])
160
161 eta0 = Constant(mu)
162 a01 = (rho*dot(dot(u,grad(u)),v) + inner(TT(u,p,eta0),DD(v)))*dx()
163
164         # Inlet Pressure
165 L01 = - (Pin)*dot(n,v)*ds(Subdomains['Inlet']) –
166           # Outlet Pressure
167 (Pout)*dot(n,v)*ds(Subdomains['Outlet'])
168
169 # Mass Conservation(Continuity)
170 a02 = (q*div(u))*dx()
171 L02 = 0
172
173 # Complete Weak Form
174 F0 = (a01 + a02) - (L01 + L02)
175        # Jacobian Matrix
176 J0 = derivative(F0,w,dw)
177
178
179
```

```
##########   Numerical Solver Properties
# Problem and Solver definitions
problemU0 = NonlinearVariationalProblem(F0,w,bcs,J0)
solverU0 = NonlinearVariationalSolver(problemU0)
# # Solver Parameters
prmU0 = solverU0.parameters
prmU0['nonlinear_solver'] = 'newton'
prmU0['newton_solver']['linear_solver'] = linearSolver

# Solve Problem
(no_iterations,converged) = solverU0.solve()

w1.assign(w)
(u1,p1) = w1.split()

err = 1.0
eps = 0.0001
norm(u1)
w0.assign(w1)
ETA = Function(C)
(u0,p0) = w0.split()

k=0
while err > eps:
    ETA = eta(u0)

    # Linear Momentum Conservation: F = a - L
               # Inertia Term
    a1 = (rho*dot(dot(u,grad(u)),v) +
      # Viscous Forces Term
inner(TT(u,p,eta(u0)),DD(v)))*dx()

             # Inlet Pressure
    L1 = - (Pin)*dot(n,v)*ds(Subdomains['Inlet']) –
            # Outlet Pressure
(Pout)*dot(n,v)*ds(Subdomains['Outlet'])
```

```
216
217     # Mass Conservation(Continuity)
218     a2 = (q*div(u))*dx()
219     L2 = 0
220
221
222     # Complete Weak Form
223     F = (a1 + a2) - (L1 + L2)
224     # Jacobian Matrix
225     J = derivative(F,w,dw)
226
227     ##########   Numerical Solver Properties
228     # Problem and Solver definitions
229     problemU = NonlinearVariationalProblem(F,w,bcs,J)
230     solverU = NonlinearVariationalSolver(problemU)
231     # # Solver Parameters
232     prmU = solverU.parameters
234     prmU['nonlinear_solver'] = 'newton'
235     prmU['newton_solver']['linear_solver'] = linearSolver
236
237     # Solve Problem
238     (no_iterations,converged) = solverU.solve()
239     w0.assign(w1); u0,p0 = w0.split()
240     w1.assign(w); u1,p1 = w1.split()
241
242     k+=1
243     print(k)
244     err = abs(norm(u0)-norm(u1))/norm(u0)
245
246 print(err)
247
248 # Open new figure
249 plt.figure(figsize=(20, 10), dpi=400, facecolor='w', edgecolor='k')
250 plot_p = plot(p)
251 plt.colorbar(plot_p)
252 plt.savefig('Pressure.png')
```

```
253
254 # Open new figure
255 plt.figure(figsize=(20, 10), dpi=400, facecolor='w', edgecolor='k')
256 plot_u = plot(u)
257 plt.colorbar(plot_u)
258 plt.savefig('Velocity.png')
259
260 w1.assign(w)
261 (u1,p1) = w1.split()
262 ux = []
263 j = []
264 for i in np.arange(0, 0.014, 0.0001):
265         j.append(i)
266         ux.append(u1(0.021, i)[0])
267 ux_tot = np.sum(ux)
268 Q = (ux_tot/len(j))*0.014
269 # plot new figure
270 plot_eta = plot(ETA)
271 plt.colorbar(plot_eta)
272 plt.savefig('Viscosity.png')
273
274 u_darcy = Q/0.014 # Darcy velocity
```