Um Estudo da Instabilidade de Saffman-Taylor com Fluido Magnético e, ou Anisotrópico

Ataias Pereira Reis, Yuri Dumaresq Sobral, Francisco Ricardo da Cunha
Universidade de Brasília
Departamento de Matemática
Brasília, Brasil
ataiasreis@gmail.com

***Abstract*—Criação de algoritmos para resolver equações diferenciais parciais. Uso de diferenças finitas para resolver as equações de Laplace e Poisson, usando método ímplicito e explícito, usando bibliotecas de álgebra linear de código aberto. Análise da diferença de tempo de resolução entre os métodos implícito e explícito. Uso do método iterativo e de resolução direta de sistema linear.**

***Index Terms*—Ferrofluidos, Navier Stokes, EDP, Poisson, Laplace, Diferenças finitas, Eigen**

I. Introdução

A instabilidade de Saffman-Taylor, também chamada de endedamento, ou *fingering*, é um fenômeno que ocorre na superfície de contato entre dois fluidos sob circunstâncias específicas. Esse fenômeno ocorre quando um fluido menos viscoso é injetado para deslocar um outro mais viscoso (na situação inversa, do fluido mais viscoso usado para movimentar o outro, a interface é estavel, não ocorrendo o endedamento). Também pode ocorrer movida pela gravidade, ao invés de injeção de um fluido em outro. Neste caso, a interface separando os fluidos de diferentes densidades está direcionada na horizontal, e o mais pesado está em cima do outro. Este tipo de fenômeno é um problema ocorrente em petrolíferas marítimas. Em tais petrolíferas, ocorre a injeção de água nos tubos de extração de petróleo, no objetivo do óleo subir. Na interface entre água e petróleo, o endedamento ocorre, originando bolhas de óleo dentro de água, que tem um efeito negativo na extração do petróleo, causando perca de óleo quando a água é jogada fora.

Para o estudo dessa instabilidade, que não é nada trivial, a solução se dá por meio de métodos numéricos auxiliados por computador. A equação de Navier Stokes deve ser discretizada e então resolvida numericamente. Ela é uma equação diferencial parcial altamente não-linear e de difícil resolução. No caso da instabilidade de Saffman-Taylor, no qual a fronteira está em movimento, que é a interface entre os dois fluidos, faz-se necessário algoritmos numéricos capazes de lidar com este movimento sem causar complicações extremas que impossibilitem a obtenção de soluções práticas.

Tais ferramentas numéricas e decisão de métodos/algoritmos a se utilizar são a primeira etapa neste projeto, para então, após se ter tais ferramentas, estudar a física do problema e propor soluções para o problema com a utilização de um ferrofluido. A apresentação no resto deste relatório mostrará até onde se alcançou na codificação dos algoritmos numéricos para resolução do problema proposto, que inicia-se com o estudo de equações diferenciais parciais e de diferenças finitas na forma contínua, e só então na forma discreta. Após isso, estuda-se em específico a equação de Navier Stokes, que é dividida em etapas, para facilitar a busca de resultados.

II. Metodologia

A metologia aqui proposta resume as tarefas realizadas, não seguindo um ordem cronológica, mas sim uma mais organizada, para estar bem divididas as seções.

A. Programação
B. Equação de Laplace
C. Equação de Poisson
D. Navier Stokes
   0.1. Discretização
   0.2. Malha Escalonada
      1. Desconsiderando a pressão
      2. Obtendo a pressão
      3. Avançando no tempo

A. Programação

O fato do problema numérico em questão ser difícil torna o tempo uma questão fundamental na escolha da linguagem de programação a ser utilizada. De início a ferramenta e linguagem de programação utilizada era o MATLAB®. O MATLAB inclui uma infinidade de algoritmos já prontos disponíveis para uso, ferramentas para plotagem de gráficos embutidas, ótimos sistemas de referências de funções, mas tem a desvantagem de ser muito caro, proprietário e consumir muita memória e processamento só de estar aberto, sem executar o próprio programa para resolver nossas equações em questão. Como o tempo é um fator crítico, como já mencionado, o custo de memória de um programa como o MATLAB, sendo executado por muitas horas ou dias, pode ser algo muito complicado, preferiu-se tomar outro rumo para o desenvolvimento dos nossos códigos.

Tais problemas levaram a uma escolha de uma linguagem de programação compilada e rápida, o C++. O fato do aluno ter experiência em C inicialmente, aliado ao fato de se haver encontrado uma biblioteca de álgebra linear chamada Eigen que evita a necessidade de criação de inúmeros algoritmos básicos, e que é feita em C++, influenciou na escolha de tal linguagem.

A Eigen[1] inclui algoritmos para resolução de sistemas lineares, criação de matrizes e alocação de memória, operações básicas de matrizes, resolução de sistemas de matrizes esparsas e vários outros algoritmos não explorados que estão à disposição, isso tudo de graça, pois é código aberto e gratuito.

Apesar de se ter escolhido o C++ e a Eigen, isto só não traz todas as ferramentas necessárias ao trabalho em questão. A plotagem de gráficos diretamente em C++ não é algo trivial, não foram encontradas bibliotecas de fácil uso que pudessem ser utilizadas de uma forma tão simples como o MATLAB. Após muita pesquisa, foi decidido utilizar o python, e suas bibliotecas de plotagem de gráficos. Os códigos em C++ passarem a ser compilados não em programas, mas em bibliotecas, e executados de dentro do ambiente Python, bem mais leve que o MATLAB, e então com o Python, que possui poderosos recursos para salvar arquivos, plotar gráficos e tudo de fácil estudo para uso, completou-se o arsenal de ferramentas de programação que são utilizadas no presente trabalho.

Além das ferramentas que lidam direto com código, é utilizado o CMake[2] para criação de Makefiles para compilar o código e o git[3] é utilizado como gerenciador de versões do projeto, de forma a sempre se ter as versões antigas dos códigos salvas, e bem organizadas.

O projeto tem sido desde certo ponto hospedado num servidor git gratuito, que pode ser obtido[4] facilmente online. Há um histórico de todas as versões dos programas do projeto, e os códigos C++, Python e instruções de como compilar e executar em sistema operacional Ubuntu.

B. Equação de Laplace

Primeiramente, teve-se a familiarização do aluno com a resolução de equações diferenciais parciais, realizando um curso na UnB, onde foram vistas principalmente a equação de Laplace, Poisson, Calor e Onda. O método de resolução aprendido foi por separação de variáveis, o método de Fourier. Após isso, o primeiro problema proposto foi resolver uma destas equações numericamente, a escolhida foi a equação de Laplace em duas dimensões, que segue abaixo:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{1}$$

A primeira etapa na resolução desta equação de maneira numérica é discretizá-la, e isto quer dizer limitar o domínio, escolher um número de pontos em cada dimensão, e representar a equação como operações mais simples que o computador possa entender, diferentes da derivada. O método escolhido para realizar a discretização foi diferenças finitas.

Para a discretização da equação 1, usa-se o método mencionado antes, que são as diferenças finitas. Este método basicamente faz a seguinte aproximação para a derivada:

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h} = \frac{d}{dx} f(x) \approx \frac{f_i - f_{i-1}}{\Delta x} \tag{2}$$

Os valores de f_i são valores discretizados da função $f(x)$, ou seja, $f_i = f(i\Delta x)$. Agora a distância entre dois pontos passa a ser de Δx, e daí a derivada se calcula como a diferença entre um ponto e outro, dividida pela distância entre eles.

Há vários tipos de diferenças finitas: progressivas, regressivas e centrais. Além disso, elas podem envolver mais de dois pontos, de forma a se obter diferentes ordens de erro. O erro também muda de acordo com quão grande é Δx, que está diretamente ligado ao número de pontos da malha. A malha é uma matriz bidimensional que contém pontos representando posições do espaço do problema. Para Laplace, utilizou-se diferenças centrais de segunda ordem, para vários tamanhos de malhas. Neste caso, o erro é da ordem de Δx^2.

$$f''(x) \approx \frac{f(x+\Delta x) - 2f(x) - f(x-\Delta x)}{\Delta x^2} \tag{3}$$

Utilizando essa equação para Laplace, chega-se na seguinte discretização, considerando $\Delta x = \Delta y$, ou seja, uma malha quadrada, e que a variável x varia de acordo com o índice i e a variável y com o índice j:

$$\frac{u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1} - 4u_{ij}}{\Delta x^2} = 0 \tag{4}$$

Reordenando essa equação:

$$u_{ij} = \frac{u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1}}{4} \tag{5}$$

A equação da fórmula 5 está na forma explícita, na qual um ponto central é obtido diretamente a partir dos pontos adjacentes. No entanto, quando a equação é aplicada ao ponto central, ele influencia no cálculo de outros, que faz com que esta fórmula deva ser aplicada muitas vezes, até se alcançar a convergência de todos os pontos do domínio. Entretanto, se apenas aplica-se essa equação a todos os pontos do domínio, já se vê um problema, pois ela sempre pede mais pontos para os lados. Por isso, além da equação, necessita-se das condições de contorno para o problema, onde-se são determinados os valores nas fronteiras do domínio, de forma direta ou indireta, dependendo se as condições são de Dirichlet, Neumann ou uma mistura. O domínio escolhido é um quadrado, e abaixo tem-se o exemplo de uma malha 6x6.

Cada cruzamento de linha indica um ponto no domínio, e a equação 5 será aplicada em quase todos estes postos, com exceção dos de fronteira. O valor dos pontos de fronteira deste primeiro problema foi escolhido fixo, utilizando condições de Dirichlet.

[1] Eigen: http://eigen.tuxfamily.org/
[2] Cmake: http://www.cmake.org/
[3] Git: http://git-scm.com/
[4] Github projeto: https://github.com/ataias/ff

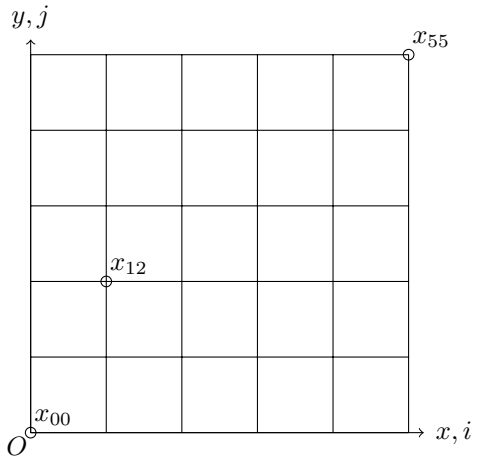

Fig. 1. Malha padrão para discretizar Laplace e Poisson

Pelo fato do método ser iterativo, com um *loop* sendo repetido muitas vezes, até a convergência, é natural escolher um método de parada. Isto poderia ser: (1) escolher um tempo limite de execução, (2) ver a diferença entre duas matrizes após uma iteração completa em seus pontos internos, para analisar se a diferença entre elas é desprezível e se indica convergência, ou ainda (3) pode-se calcular os valor da equação de Laplace em cada ponto interno, que seria o melhor a se fazer, e analisar se a equação está dentro de uma faixa de erro desejada. A segunda maneira de se fazer isto foi escolhida na primeira parte, e foi uma difícil decisão escolher qual utilizar, pois ainda não se compreendia como aplicar o terceiro método.

O problema inicialmente proposto tem condições de Dirichlet, que indica a função em cada fronteira do domínio. E os problemas todos são resolvidos num quadrado de lado igual a 1.

$$\begin{aligned} \nabla^2 u &= 0 \qquad (6)\\ (x,y) \in D\,; 0 \le x \le 1 \,\mathrm{e}\, 0 \le y \le 1 & \\ u(0,y) &= h_1(y)\\ u(1,y) &= h_2(y)\\ u(x,0) &= g_1(x)\\ u(x,1) &= g_2(x)\\ \Delta x &= \frac{1}{n-1} \end{aligned}$$

Agora, apresenta-se o algoritmo que se utilizou para resolver a equação

Na equação de Laplace, o Δx não aparece diretamente, tanto que não aparece no algoritmo, mas esta variável contém um valor importante que será usada nos problemas posteriores de Poisson e Navier Stokes.

O método de resolução apresentado é explícito, no qual o valor de cada ponto no método é obtido diretamente dos pontos adjacentes, e se faz necessária uma resolução iterativa. Isso tem um custo de processamento razoável, no qual o tempo aumenta grandemente conforme o número de pontos n escolhido. Tendo em vista uma resolução mais rápida e direta, pode-se utilizar o método implícito.

input : Matriz com as condições de contorno de tamanho $n \times n$
output: Matriz com cada ponto interno satisfazendo a equação de Laplace e a fronteira igual à entrada

```
STOP = 0;
err ← 10^-6;
while STOP ≠ 1 do
    u^old ← u;
    for i ← 1 to n − 2 do
        for j ← 1 to n − 2 do
            u_ij ← (u_{i+1j} + u_{i−1j} + u_{ij+1} + u_{ij−1})/4;
        end
    end
    ERROR ← Norma(u^old − u);
    if ERROR < err then
        STOP ← 1;
    end
end
```

Algorithm 1: Resolvendo Laplace Iterativamente

Anteriormente, a partir da equação 4, isolou-se u_{ij} e então resolveu-se iterativamente nosso problema. Outra forma de se resolver o mesmo problema é pegar essa mesma equação e trocar os índices (i, j) por números de fato, obtendo várias equações, e isolando na parte da direita os termos de fronteira. Assim, para um sistema $n \times n$, dos n^2 pontos da malha dos índices $(0,0) \to (n-1, n-1)$, $(n-1)^2$ pontos são internos, correspondendo aos índices de $(1,1) \to (n-2, n-2)$. A seguir temos as equações para um sistema 5×5:

$$\begin{aligned} u_{12} + u_{21} - 4u_{11} &= -u_{01} - u_{10}\\ u_{11} + u_{22} + u_{31} - 4u_{21} &= -u_{20}\\ u_{21} + u_{32} - 4u_{31} &= -u_{41} - u_{30}\\ u_{22} + u_{13} + u_{11} - 4u_{12} &= -u_{02}\\ u_{32} + u_{12} + u_{23} + u_{21} - 4u_{22} &= 0\\ u_{22} + u_{33} + u_{31} - 4u_{32} &= -u_{42}\\ u_{23} + u_{12} - 4u_{13} &= -u_{03} - u_{14}\\ u_{33} + u_{13} + u_{22} - 4u_{23} &= -u_{24}\\ u_{23} + u_{32} - 4u_{33} &= -u_{43} - u_{34} \end{aligned}$$

O conjunto de equações acima é um sistema linear, e é possível representá-lo na forma matricial $Ax = b$. O que torna este problema interessante é que a matriz A e b tem padrões muito bem definidos. A matriz A toma a seguinte

forma:

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}$$

Observando o padrão de formação da matriz A pentadiagonal e sabendo-se também obter o vetor de valores conhecidos b, tem-se um sistema linear para obter os pontos internos x. A solução é $x = A^{-1}b$.

Como se nota, a resolução é muito mais direta, mas o algoritmo para se resolver este sistema linear pode ser muito mais complexo. Não se preocupou em criar um algoritmo para isso, mas sim em utilizar um pronto da biblioteca Eigen. Mas há um problema para este método, que um sistema pode ficar extremamente grande com um n relativamente pequeno, tão grande, ao ponto de necessitar de mais memória do que o computador tem. Isso se resolve utilizando matrizes esparsas, que só salvam na memória elementos diferentes de zero. A matriz A é uma matriz esparsa, na qual a maioria de seus elementos é igual a 0. Também utilizaram-se métodos prontos para resolução de sistemas esparsos provenientes da Eigen.

C. Equação de Poisson

A equação de Poisson é muito similar a de Laplace, mas com uma diferença, que é a de haver uma função forçamento existente no lado direito da equação.

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \tag{7}$$

Este foi o segundo problema proposto, e o fato de ter essa função $f(x, y)$ complica o problema porque muita coisa que antes era desconsiderada com o 0 na equação de Laplace não é mais ignorado, mas sim levado em consideração. No caso explícito, tem-se a seguinte discretização obtida:

$$u_{ij} = \frac{1}{4}[(u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1}) - \Delta x^2 f_{ij}] \tag{8}$$

O algoritmo iterativo para este caso é o mesmo anterior, substituindo a equação 5 por 8 no algoritmo.

Para o caso implícito, o problema é muito similar, bastando mudar o vetor b, que passa a apresentar termos $\Delta x^2 f_{ij}$.

D. Navier Stokes

Após resolver-se as equações de Laplace e Poisson com condições de Dirichlet, pelos métodos explícito e implícito, tem-se já uma base para partir para a próxima etapa: Navier Stokes. Essa equação tem muitas complicações, e está levando ainda certo tempo para ter-se um programa que a resolva, e esteja de fato livre de bugs. A complexidade do programa é maior, tem muitas etapas, e tem ainda uma evolução no tempo que não foi lidada no problema de Poisson. A equação de Navier Stokes é a seguinte:

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \mu \nabla^2 \mathbf{v} + \mathbf{f} \tag{9}$$
$$\nabla \cdot \mathbf{v} = 0$$

A discretização não é simples e é um pouco diferente da que foi feita anteriormente no problema de Poisson. Primeiramente, tem-se de saber que a discretização de Navier Stokes que será feita aqui é a fórmulação em variáveis primitivas. Isso quer dizer que o problema será trabalhado com as variáveis originais de velocidade u, v (em x e y) e pressão p. Outra forma de resolver é usando uma formulação com a vorticidade, que não será trabalhada aqui.

No que se aprendeu lendo um material em pdf que ensina o processo de resolução em variáveis primitivas, sabe-se que utilizando a malha da Figura 1, ocorre modos espúrios de pressão que causa uma solução de velocidades que contém oscilações não esperadas no resultado do problema. O resultado desta forma não é satisfatório, e para resolver este problema usa-se uma malha escalonada.

A malha escalonada é uma malha na qual um ponto não salva a informação que é exatamente do local onde o ponto se situa, mas de um local próximo. No ponto $P = (i, j)$ são salvos cinco valores: força horizontal, força vertical, pressão, velocidade horizontal e vertical. Essas são representadas pelos símbolos f_x, f_y, p, u e v. Um elemento da malha escalonada é representado na figura a seguir, na qual as forças não estão mostradas, mas elas são localizadas nas mesmas posições que as velocidades.

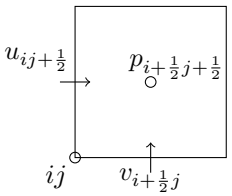

Fig. 2. Elemento de malha escalonada

Diferente dos problemas de Laplace e Poisson, no qual um elemento u_{ij} que estava salvo no computador na posição ij realmente é o elemento u_{ij}, agora o elemento u_{ij} salvo na memória do computador é fisicamente o elemento $u_{ij+\frac{1}{2}}$, enquanto o elemento v_{ij} se refere ao elemento $v_{i+\frac{1}{2}j}$. De maneira similar ocorre para as forças. Para a pressão, p_{ij} corresponde ao ponto $p_{i+\frac{1}{2}j+\frac{1}{2}}$.

1) Desconsiderando a pressão: A primeira etapa consiste literalmente de desconsiderar a pressão. Há um termo de pressão na equação 9 que é ∇p. A partir dessa simplificação, faz-se a discretização da equação. Alguns termos foram incluídos nas equações a seguir de forma a tornar algumas das equações menos extensas.

$$\mathbf{v}_{ij}^{s} = (u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1})\mathbf{i} + (v_{i+1j} + v_{i-1j} + v_{ij+1} + v_{ij-1})\mathbf{j} \quad (10)$$

$$\mathbf{v}_{ij}^{t} = 0.25(u_{ij} + u_{i+1j} + u_{i+1j-1} + u_{ij-1})\mathbf{i} + 0.25(v_{ij} + v_{i-1j} + v_{i-1j+1} + v_{ij+1})\mathbf{j} \quad (11)$$

$$u_{ij}^{*} = \left[\frac{\mu}{\rho}\left(\frac{u_{ij}^{s} - 4u_{ij}}{\Delta x^2}\right) + \frac{1}{\rho}\frac{f_{x,ij} + f_{x,i-1j}}{2}\right] + u_{ij} - \left[u_{ij}\frac{u_{i+1j} - u_{i-1j}}{2\Delta x} - v_{ij}^{t}\frac{u_{ij+1} - u_{ij-1}}{2\Delta x}\right]\Delta t \quad (12)$$

$$v_{ij}^{*} = \left[\frac{\mu}{\rho}\left(\frac{v_{ij}^{s} - 4v_{ij}}{\Delta x^2}\right) + \frac{1}{\rho}\frac{f_{y,ij} + f_{y,ij-1}}{2}\right] + v_{ij} - \left[u_{ij}^{t}\frac{v_{i+1j} - v_{i-1j}}{2\Delta x} - v_{ij}\frac{v_{ij+1} - v_{ij-1}}{2\Delta x}\right]\Delta t \quad (13)$$

III. Resultados

A. Laplace - Método Iterativo

Foram realizados vários testes utilizando as seguintes condições:

$$\nabla^2 u = 0 \quad (14)$$

$$u \quad (15)$$

IV. Conclusão

Agradecimentos

Primeiramente agradeço a Deus pela oportunidade de ter feito parte deste trabalho. Agradeço também ao professor Yuri Dumaresq pelas orientações e ânimo no ensino que eu vi nele e que também me animaram na resolução destes problemas numéricos. Agradeço também ao CNPq pela bolsa de incentivo à pesquisa.

References

[1] Getting Started with Ubuntu 10.10
[2] OpenKinect