

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

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

Resumo—Criação de algoritmos para resolver equações diferenciais parciais. Uso de diferenças finitas para resolver as equações de Laplace e Poisson, usando método implícito 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, ou *fingering*, que pode ser visto na Figura 1, 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 estável, não ocorrendo o endodamento). 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 *fingering* ocorre, originando bolhas de óleo dentro de água, que tem um efeito negativo na extração do petróleo, causando perda de óleo quando a água é jogada fora. Uma maneira de se controlar este fenômeno é usar fluidos magnéticos e a aplicação de campos magnéticos [1].

O objetivo deste estudo é estudar essa instabilidade 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 [2].

Tais ferramentas numéricas e decisão de métodos/algoritmos a se utilizar são a primeira etapa neste

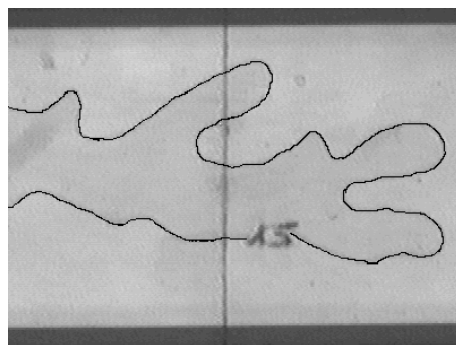


Figura 1. Demonstração de fingering

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. Este relatório mostrará até onde se alcançou na codificação dos algoritmos numéricos para resolução do problema proposto. Iniciou-se com o estudo de equações diferenciais parciais bastante conhecidas, como a equação do Calor, de Laplace, e métodos de solução numérica, especificamente o método de diferenças finitas. Após isso, estudou-se em específico a equação de Navier Stokes [3].

A metodologia aqui proposta resume as tarefas realizadas ao longo da execução do projeto:

- A. Técnicas de Computação Básica
- B. Solução da equação de Laplace
- C. Solução da equação de Poisson
- D. Solução da equação de Navier Stokes

Considerações gerais sobre a equação

1. Desconsiderando a pressão
2. Obtendo a pressão
3. Avançando no tempo

II. TÉCNICAS DE COMPUTAÇÃO BÁSICA

Pelo fato da solução numérica da equação diferencial parcial exigir um grande esforço computacional, o tempo de processamento é 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 consumir muita memória e processamento. O tempo é um fator crítico e o custo computacional de um programa como o MATLAB, sendo executado por muitas horas ou dias, pode ser algo excessivo e, portanto, preferiu-se tomar outro rumo para o desenvolvimento dos códigos.

Tais problemas levaram a uma escolha de uma linguagem de programação compilada e mais rápida, em comparação a uma linguagem interpretada como o MATLAB. A ferramenta escolhida foi o C++. O fato do aluno ter experiência em C inicialmente, aliado ao fato de se haver encontrado uma biblioteca de álgebra linear chamada Eigen¹ que evita a necessidade de criação de inúmeros algoritmos básicos, e que é feita em C++, influenciou na escolha de tal linguagem.

A Eigen 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++ passaram a ser compilados não em programas, mas em bibliotecas, e executados de dentro do ambiente Python, que possui poderosos recursos para salvar arquivos, plotar gráficos e com muita documentação disponível.

É utilizado o CMake² para criação de Makefiles. Makefiles são arquivos que tem o objetivo de compilar códigos, neles há informação das bibliotecas a qual um programa irá necessitar para ser compilado, localizações de códigos e logicamente o compilador que será utilizado. O CMake simplifica a criação de Makefiles e, por isso, é utilizado.

Outro programa que se faz uso neste projeto é o git³, que é um gerenciador de versões, 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 facilmente online⁴. Neste site do projeto há um histórico de quase todas as versões dos programas do projeto, e os códigos C++, Python e instruções de como compilar e executar em sistema operacional Ubuntu.

III. SOLUÇÃO DA EQUAÇÃO DE LAPLACE

Primeiramente, o aluno se familiarizou com a resolução de equações diferenciais parciais, estudando em detalhes

a solução analítica das equações de Laplace, Poisson, Calor e Onda. O método de resolução aprendido foi o método de Fourier. Tal método consiste na separação de variáveis. Tendo-se uma função incógnita $u(x, y)$, propõe-se uma solução do tipo $H(x)G(y)$ e este termo é então trabalhado na equação diferencial, de forma a obter-se equações diferenciais ordinárias que podem ser resolvidas mais facilmente. As soluções muitas vezes envolvem o uso da série de Fourier, onde partes das EDPs são representadas como uma soma de senos e cossenos, por necessidade pelo uso do método.

Após isso, o primeiro problema proposto foi resolver uma destas equações numericamente, a escolhida foi a equação de Laplace com condições de fronteira de Dirichlet em duas dimensões, num quadrado 1×1 . Ela foi escolhida porque é uma EDP muito simples. Verifica-se abaixo o problema:

$$\begin{aligned} \nabla^2 u &= \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \\ u(x, 0) &= f_1(x), u(x, 1) = f_2(x) \\ u(0, y) &= g_1(y), u(1, y) = g_2(y) \end{aligned} \quad (1)$$

A primeira etapa na resolução numérica desta equação de maneira é discretizá-la, e isto implica em tornar finito o número de pontos em cada dimensão, e representar a equação como operações mais simples e que o computador seja capaz de entender, operações diferentes da derivada, tais como soma, subtração e multiplicação. O método escolhido para realizar a discretização foi diferenças finitas. Este método basicamente faz a seguinte aproximação para a derivada:

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

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 . Este valor não está sendo utilizado agora é a distância Δx . Não é utilizada pelo fato dos valores de Δx e Δy serem iguais e também pelo fato do lado direito da equação de Laplace ser igual a zero. Caso essas condições fossem diferentes, o problema não se simplificaria. A fórmula para se calcular o espaçamento Δx é dada na Equação 3.

$$\Delta x = 1/(n-1) \quad (3)$$

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 quanto 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. Para resolver a equação de Laplace, utilizou-se diferenças centrais de segunda ordem, cujo erro é da ordem de Δx^2 . A diferença central de segunda ordem é apresentada na equação 4.

$$f''(x) \approx \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2} \quad (4)$$

¹Eigen: <http://eigen.tuxfamily.org/>

²Cmake: <http://www.cmake.org/>

³Git: <http://git-scm.com/>

⁴Github projeto: <https://github.com/ataias/ff>

Utilizando a discretização da segunda derivada mostrada na Equação 4 na equação de Laplace, obtém-se a Equação 5, que é a equação de Laplace discretizada. Note-se que $\Delta x = \Delta y$ e que a variável x varia de acordo com o índice i e a variável y de acordo com o índice j .

$$\frac{u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1} - 4u_{ij}}{\Delta x^2} = 0 \quad (5)$$

Reordenando essa equação algébrica para encontrar o valor de u_{ij} :

$$u_{ij} = \frac{u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1}}{4} \quad (6)$$

A Equação 6 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. A convergência é a aproximação do resultado, quando os valores da malha alcançam valores praticamente estacionários, mudando muito pouco, a solução é obtida.

Caso essa equação é aplicada a todos os pontos do domínio, se nota um problema, pois ela requiere pontos adjacentes em todo local no qual ela é aplicada. 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. No caso atual, as condições são de Dirichlet, que indica o valor de forma direta que a fronteira deve ter. O domínio escolhido é um quadrado, como já mencionado, e na Figura 2 tem-se o exemplo de uma malha 6x6.

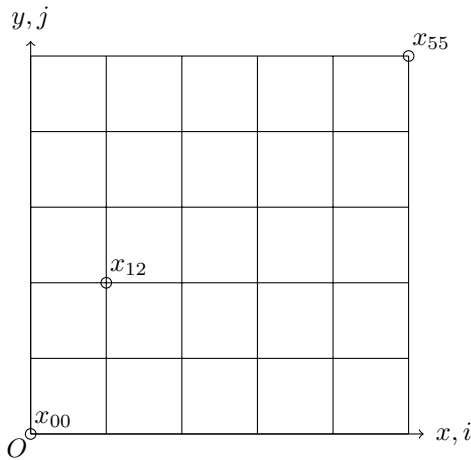


Figura 2. Malha padrão para discretizar Laplace e Poisson

Cada cruzamento de linha indica um ponto no domínio, e a Equação 6 será aplicada em quase todos estes pontos, com exceção dos de fronteira, cujo valor da função em tais é determinada pelas condições de Dirichlet. Pontos de fronteira são aqueles que estão sobre uma coluna ou linha dos extremos da matriz.

Pelo fato do método ser iterativo, com um *loop* sendo repetido muitas vezes, até a convergência ser obtida, 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. No Algoritmo 1, é apresentada a formulação inicial utilizada.

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

$STOP = 0;$

$err \leftarrow 10^{-6};$

while $STOP \neq 1$ **do**

$u^{old} \leftarrow u;$

for $i \leftarrow 1$ **to** $n - 2$ **do**

for $j \leftarrow 1$ **to** $n - 2$ **do**

$u_{ij} \leftarrow (u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1})/4;$

end

end

$ERROR \leftarrow Norma(u^{old} - u);$

if $ERROR < err$ **then**

$STOP \leftarrow 1;$

end

end

Algoritmo 1: Resolvendo a equação de Laplace Iterativamente

O método de resolução apresentado no Algoritmo 1 foi o primeiro a ser estudado por simplicidade. Este método é chamado de Gauss-Seidel. Note que o valor de cada ponto no método é obtido diretamente dos pontos adjacentes e, portanto, 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. Os resultados que mostram essa relação com o número de pontos são mostrados nas Figuras 3 e 4. Tendo em vista uma resolução mais rápida e direta, pode-se utilizar o método implícito.

Nas Figuras 3 e 4 tem-se uma visão geral do tempo que o programa criado consome para resolver a equação de Laplace com condições de Dirichlet, pelo método explícito iterativo. Os conjuntos de pontos obtidos nos dois últimos gráficos mencionados foram aproximados por polinômios cúbicos. A diferença entre os dois se observa no tempo de cálculo quando o número de pontos em cada dimensão, n , ultrapassa um valor em torno de 100. Pelo fato de serem polinômios cúbicos, ambos crescem muito rápido com o número de pontos. O cálculo com 4 threads foi mais rápido pelo fato de se estar utilizando mais poder de processamento do computador nos cálculos.

As Figuras 5, 7, 9 e 11 são um conjunto de resultados da equação de Laplace com condições de fronteira de Dirichlet, resolvidas pelo método iterativo explícito, com

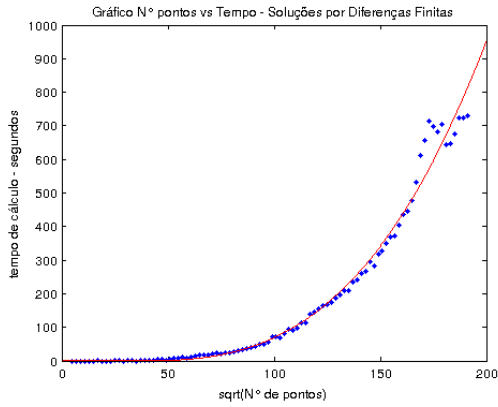


Figura 3. Pontos vs Tempo de Cálculo - 1 thread - Método Explícito

uso de 1 thread, enquanto as Figuras 6, 8, 10 e 12 são soluções do mesmo problema com o uso de 4 threads do computador. As condições de contorno em todas são as mesmas em todas essas figuras, como se pode notar pelo formato similar de cada uma das soluções. Os lados tem valores de condições de Dirichlet que são $u(x, 0) = 0$, $u(x, 1) = 50$, $u(0, y) = 75$ e $u(1, y) = 100$.

Note que que conforme se aumenta o número de pontos, o gasto computacional chega a ser excessivo, conforme se observa no tempo gasto para cada solução. O tempo, em segundos, sem nenhuma casa decimal depois da vírgula, está disponível para observação nas descrições das Figuras 5 a 12.

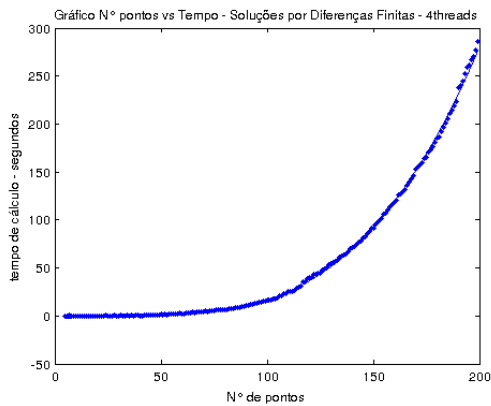


Figura 4. Pontos vs Tempo de Cálculo - 4 threads - Método Explícito

O uso de paralelismo ocorreu por causa do elevado custo computacional para malhas maiores e, sabendo-se que computadores atuais tem múltiplos threads, fez-se uso deste recurso para analisar a relação entre tempo e número de pontos⁵ com paralelismo em ação. Resultados similares foram obtidos no caso de múltiplos threads, mudando somente o tempo necessário para cada solução, e podem ser vistos nas Figuras 6, 8, 10 e 12.

⁵Pontos: o eixo de número de pontos nos gráficos do método explícito na verdade é a ordem da malha, que é a raiz quadrada do número de pontos.

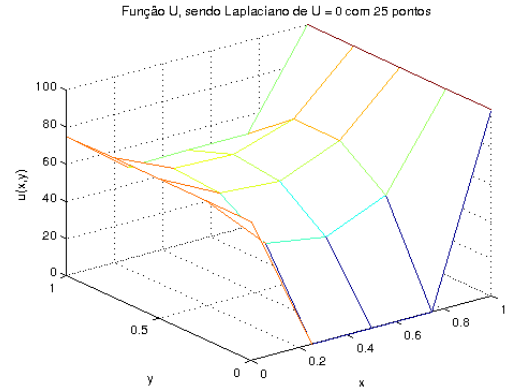


Figura 5. Malha 5x5 - 1 thread - Tempo: 0s - Método Explícito

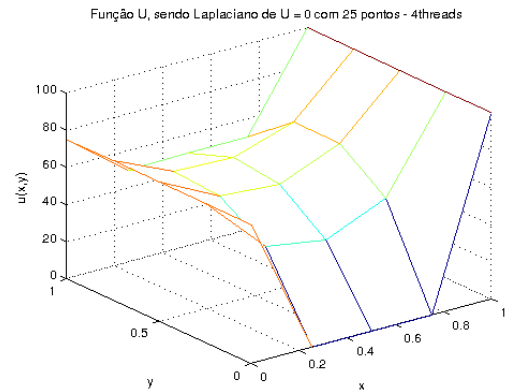


Figura 6. Malha 5x5 - 4 threads - Tempo: 0s - Método Explícito

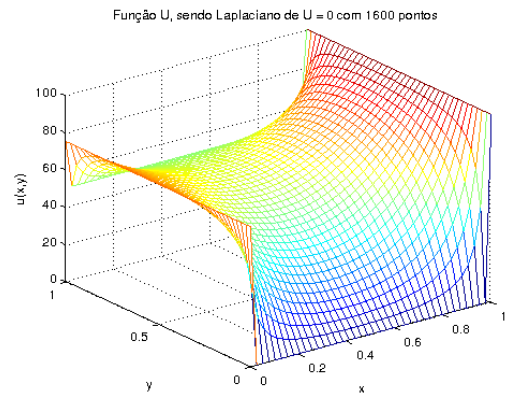


Figura 7. Malha 40x40 - 1 thread - Tempo: 2s - Método Explícito

No caso do método implícito, utilizando matrizes esparsas, os resultados foram espantosamente mais rápidos. Na maior malha que foi calculada no método explícito, foram gastas 10h⁶ no cálculo com 1 thread e 2h40min⁷ no cálculo com 4 threads. No método implícito com matrizes esparsas, o mesmo resultado foi obtido em menos de 1min. O método implícito não é utilizado em Navier Stokes devido ao problema da pressão nesta equação ter fronteiras de

⁶36083s

⁷9610s

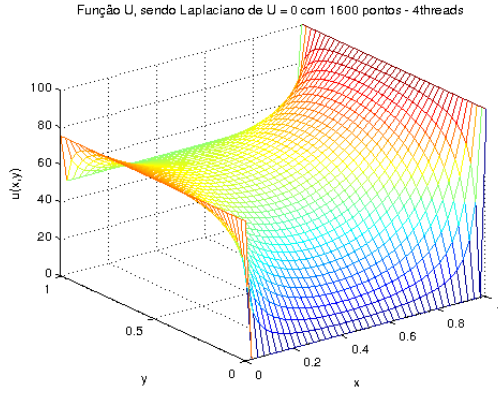


Figura 8. Malha 40x40 - 4 threads - Tempo: 0s - Método Explícito

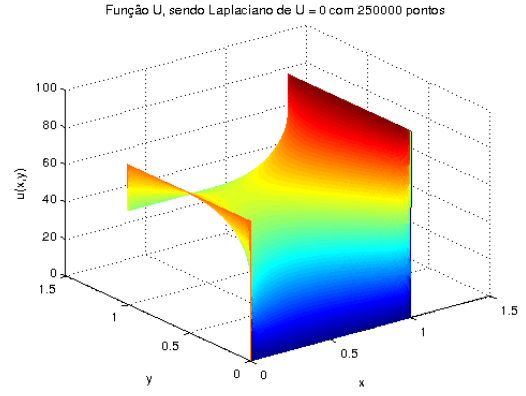


Figura 11. Malha 500x500 - 1 thread - Tempo: 36083s - Método Explícito

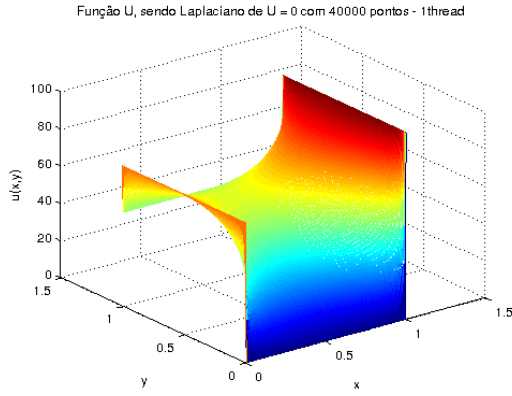


Figura 9. Malha 200x200 - 1 thread - Tempo: 1235s - Método Explícito

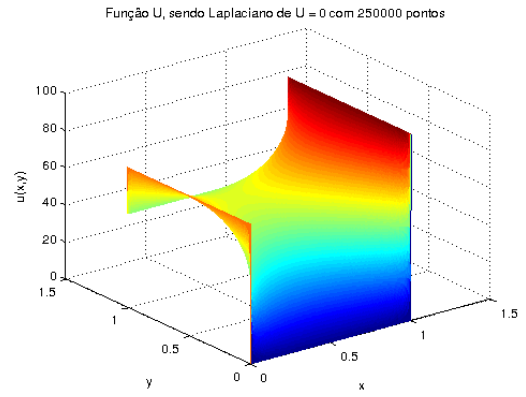


Figura 12. Malha 500x500 - 4 threads - Tempo: 9610s - Método Explícito

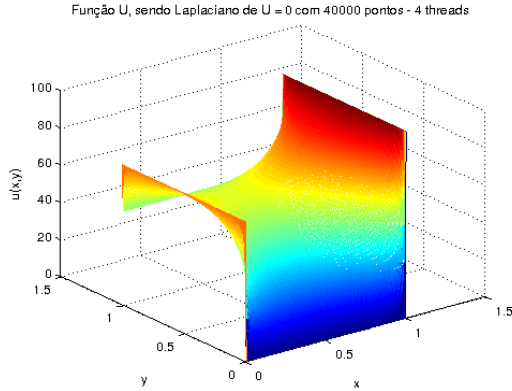


Figura 10. Malha 200x200 - 4 threads - Tempo: 274s - Método Explícito

Neumann, e o método implícito se torna muito complicado. Os resultados para Poisson são similares, só mudando o tempo no caso do método explícito.

Outra forma de se resolver o mesmo problema é substituir na Equação 5 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 $(0, 0) \rightarrow (n-1, n-1)$, $(n-1)^2$ pontos são internos, correspondendo aos índices

de $(1, 1) \rightarrow (n-2, n-2)$. As Equações 7 a 15 foram obtidas para um sistema 5×5 .

$$u_{12} + u_{21} - 4u_{11} = -u_{01} - u_{10} \quad (7)$$

$$u_{11} + u_{22} + u_{31} - 4u_{21} = -u_{20} \quad (8)$$

$$u_{21} + u_{32} - 4u_{31} = -u_{41} - u_{30} \quad (9)$$

$$u_{22} + u_{13} + u_{11} - 4u_{12} = -u_{02} \quad (10)$$

$$u_{32} + u_{12} + u_{23} + u_{21} - 4u_{22} = 0 \quad (11)$$

$$u_{22} + u_{33} + u_{31} - 4u_{32} = -u_{42} \quad (12)$$

$$u_{23} + u_{12} - 4u_{13} = -u_{03} - u_{14} \quad (13)$$

$$u_{33} + u_{13} + u_{22} - 4u_{23} = -u_{24} \quad (14)$$

$$u_{23} + u_{32} - 4u_{33} = -u_{43} - u_{34} \quad (15)$$

O conjunto das Equações 7 a 15 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. 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.

IV. 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.

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \quad (16)$$

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:

$$u_{ij} = \frac{1}{4}[(u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1}) - \Delta x^2 f_{ij}] \quad (17)$$

O algoritmo iterativo para este caso é o mesmo anterior, substituindo a equação 6 por 17 no algoritmo.

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

A equação de Poisson realmente complicou quando se preparou o algoritmo numa versão mais similar àquela que é utilizada em Navier Stokes. Esta versão resolve o problema de Poisson com as quatro fronteiras com condições de Neumann, isto é:

$$\frac{\partial u}{\partial n} = f \quad (18)$$

Isto resulta em maior processamento devido à esta equação gerar condições na fronteira que não são inicialmente fixas, mas que também são incógnitas. O problema foi resolvido e é apresentado na seção de resultados.

A Equação 19 foi resolvida. Note que se trata de uma equação de Poisson com condições de contorno de Neumann.

$$\nabla^2 u = 1 \quad (19)$$

$$\frac{\partial u}{\partial n} = 0 \text{ na fronteira, exceto} \quad (20)$$

$$\frac{\partial u(x, 0)}{\partial y} = 2x \quad (21)$$

As soluções da Equação 19 são apresentadas nas Figuras 13 e 14.

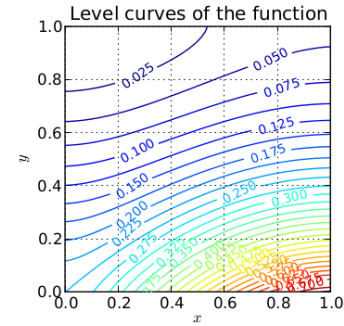


Figura 13. Poisson com 4 fronteiras de Neumann - Linhas de contorno

V. EQUAÇÃO DE NAVIER STOKES

Considerações gerais

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 muitas maior, tem várias etapas, e tem ainda uma evolução no tempo que não foi lidada nos problemas anteriores. A equação de Navier Stokes é a seguinte:

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \mu \nabla^2 \mathbf{v} + \mathbf{f} \quad (22)$$

$$\nabla \cdot \mathbf{v} = 0 \quad (23)$$

A discretização não é simples e é um pouco diferente das que foram feitas anteriormente. Em primeiro lugar, tem-se de saber que a discretização de Navier Stokes que será feita aqui é a formulação 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 sobre tal formulação do processo de resolução em variáveis primitivas, sabe-se que

$\nabla^2 u = 1$ Neumann conditions

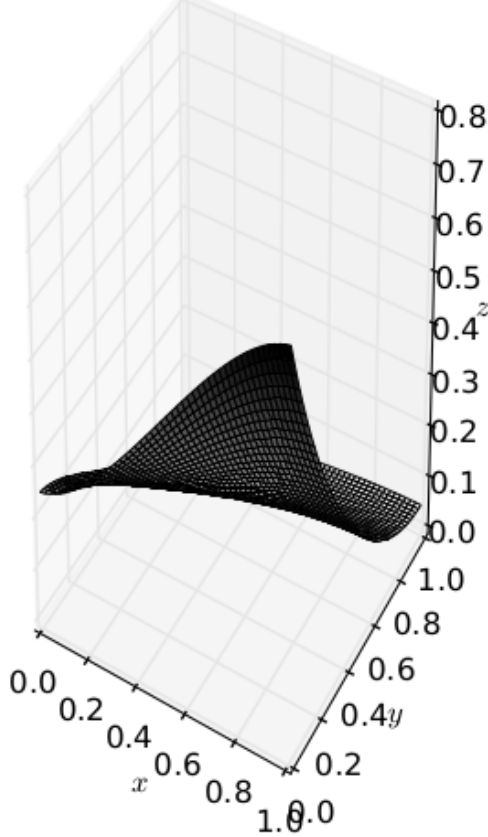


Figura 14. Poisson com 4 fronteiras de Neumann

utilizando uma malha como a da Figura 2, ocorrem modos espúrios de pressão que causam um perfil de velocidades que contém oscilações não esperadas no resultado do problema. O resultado, obtido com tal malha, desta forma, não é satisfatório, e para resolver este problema usa-se uma malha escalonada.

A malha escalonada é uma malha na qual valores guardados num ponto se referem à informação em um outro ponto, 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.

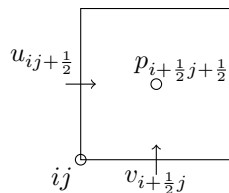


Figura 15. Elemento de malha escalonada

Diferente dos problemas de Laplace e Poisson, no qual

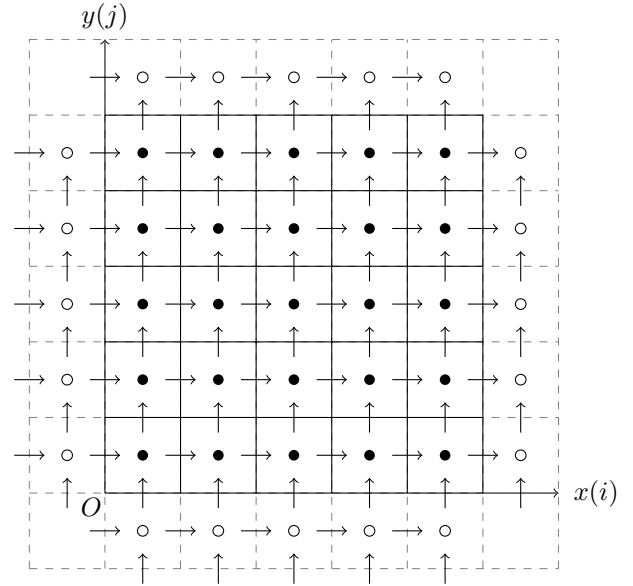


Figura 16. Malha escalonada. Círculos são pontos internos e de fronteira enquanto circunferências indicam fronteira imaginária adicionada

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 $p_{i+\frac{1}{2}j+\frac{1}{2}}$. Isso influencia diretamente na discretização, e na posterior criação dos gráficos.

Uma malha escalonada completa é apresentada na Figura a seguir:

Na malha escalonada da Figura 16, tem-se ainda uma nova adição: uma fronteira imaginária. Quando se discretiza a equação de Navier Stokes, faz-se ainda necessários outros pontos além do internos e fronteira. As equações apresentadas são discretizadas em domínios similares ao de cima, onde a única coisa que muda é o tamanho do domínio, que pode ser maior.

O problema de Navier Stokes é trabalhado em três etapas principais:

- 1) Desconsiderando a pressão
- 2) Obtendo a pressão
- 3) Avanço no tempo

Inicia-se agora a descrição da metodologia empregada em tais etapas.

A. Desconsiderando a pressão

A primeira etapa consiste literalmente de desconsiderar a pressão. Há um termo de pressão na Equação de Navier Stokes (eq. 22) que é o termo ∇p , chamado de gradiente de pressão. Elimina-se ele da equação e a partir dessa simplificação, faz-se a discretização da equação. Pelo fato da malha ser escalonada, termos de velocidade estão em pontos diferentes da malha, e há momentos nos quais é necessário elementos de velocidade que não estão salvos

na malha, fazendo-se necessária obter uma média de velocidades próximas para obter as velocidades necessárias na discretização.

$$\mathbf{v}_{ij}^s = (u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1})\mathbf{i} + (v_{i+1j} + v_{i-1j} + v_{ij+1} + v_{ij-1})\mathbf{j} \quad (24)$$

$$\mathbf{v}_{ij}^t = 0.25(u_{ij} + u_{i+1j} + u_{i-1j-1} + u_{ij-1})\mathbf{i} + 0.25(v_{ij} + v_{i-1j} + v_{i-1j+1} + v_{ij+1})\mathbf{j} \quad (25)$$

$$u_{ij}^* = \left[\frac{\mu}{\rho} \left(\frac{u_{ij}^s - 4u_{ij}}{\Delta x^2} \right) + \frac{1}{\rho} f_{x,ij} \right] + u_{ij} - \left[u_{ij}^t \frac{u_{i+1j} - u_{i-1j}}{2\Delta x} - v_{ij}^t \frac{u_{ij+1} - u_{ij-1}}{2\Delta x} \right] \Delta t \quad (26)$$

$$v_{ij}^* = \left[\frac{\mu}{\rho} \left(\frac{v_{ij}^s - 4v_{ij}}{\Delta x^2} \right) + \frac{1}{\rho} f_{y,ij} \right] + v_{ij} - \left[u_{ij}^t \frac{v_{i+1j} - v_{i-1j}}{2\Delta x} - v_{ij}^t \frac{v_{ij+1} - v_{ij-1}}{2\Delta x} \right] \Delta t \quad (27)$$

Com isto, tem-se:

$$\mathbf{v}^* = (u^*, v^*)$$

As equações para u_{ij}^* e v_{ij}^* são utilizadas para obter a solução de velocidades da equação de Navier Stokes sem a pressão, portanto, não é a solução real, mas está relacionada com a solução.

B. Obtendo a pressão

A forma de obter a pressão é utilizando as Equações 22 e 23. Primeiro é obtido o divergente da Eq. 22 e então, utilizando a Equação 23, se simplifica a equação obtida, obtendo-se o seguinte:

$$\nabla^2 p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{v}^* = \frac{\rho}{\Delta t} \left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y} \right) \quad (28)$$

O termo da direita na equação acima será chamado de *DIV*, e é discretizado na equação seguinte:

$$DIV_{ij} = \frac{\rho}{\Delta t} \left(\frac{u_{i+1j}^* - u_{ij}^*}{\Delta x} + \frac{v_{ij+1}^* - v_{ij}^*}{\Delta x} \right) \quad (29)$$

A equação que estamos tratando é a de Poisson, e resultará na pressão desejada para resolvermos o problema de Navier Stokes. A Equação 17 é usada para pontos internos da malha. Já para a fronteira, a situação muda em relação à anterior. A partir da equação de Navier Stokes podem ser obtidas as condições de contorno para a pressão. Veremos que tais condições são todas de Neumann, fazendo com que não haja uma única solução de pressão, mas como na equação de Navier Stokes só aparece gradiente de pressão, qualquer solução de pressão pode ser usada para então obter esse gradiente. Destrinchando a Equação 22 em duas coordenadas, tem-se:

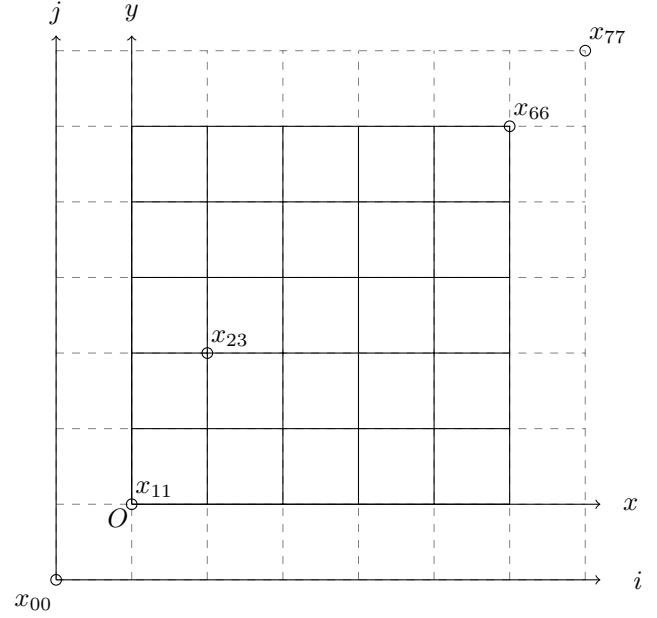


Figura 17. Mostra dos índices dos pontos do domínio e pontos imaginários

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = -\frac{1}{\rho} \frac{\partial p}{\partial x} + \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f_x \quad (30)$$

$$\frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} = -\frac{1}{\rho} \frac{\partial p}{\partial y} + \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) + f_y \quad (31)$$

Se fizermos uma análise dos pontos na parede da esquerda ou direita, e aplicarmos as condições de impenetrabilidade e não-deslizamento, que dizem que $u = 0$ e $v = 0$ na parede, temos a simplificação da equação, pois vários termos se tornam zero. As equações para as duas paredes verticais é a seguinte:

$$\left. \frac{\partial p}{\partial x} \right|_{\text{parede}} = \rho \nu \left. \frac{\partial^2 u}{\partial x^2} \right|_{\text{parede}} + \rho f_x \quad (32)$$

Para as paredes que ficam na horizontal tem-se:

$$\left. \frac{\partial p}{\partial y} \right|_{\text{parede}} = \rho \nu \left. \frac{\partial^2 v}{\partial y^2} \right|_{\text{parede}} + \rho f_y \quad (33)$$

As discretizações das Equações 32 e 33 devem ser feitas considerando-se a malha escalonada, da forma como os dados são salvos no computador. O seguinte esquema mostra a origem O do nosso domínio e o ponto no computador que representa a origem imaginária x_{00} :

Esta malha é mostrada como na figura acima porque em C++ não é possível ter elementos de matrizes com índices negativos, como em Fortran. Discretizando as equações e isolando os termos da fronteira imaginária, que se deseja

obter, tem-se o seguinte:

$$p_{i-1j}|_{i=1} = p_{ij} - \frac{\mu}{\Delta x} (2u_{ij} - 5u_{i+1j} + 4u_{i+2j} - u_{i+3j}) - \rho \Delta x f_{ij} \quad (34)$$

$$p_{i+1j}|_{i=n} = p_{ij} + \frac{\mu}{\Delta x} (2u_{ij} - 5u_{i-1j} + 4u_{i-2j} - u_{i-3j}) + \rho \Delta x f_{ij} \quad (35)$$

$$p_{ij-1}|_{j=1} = p_{ij} - \frac{\mu}{\Delta x} (2v_{ij} - 5v_{ij+1} + 4u_{ij+2} - u_{ij+3}) - \rho \Delta x f_{ij} \quad (36)$$

$$p_{ij+1}|_{j=n} = p_{ij} + \frac{\mu}{\Delta x} (2v_{ij} - 5v_{ij-1} + 4u_{ij-2} - u_{ij-3}) + \rho \Delta x f_{ij} \quad (37)$$

Tais equações devem então ser utilizadas no algoritmo para resolver a equação de Poisson, lembrando que essas condições são de Neumann, e então a fronteira não é fixa, mas irá variar até o momento que o algoritmo iterativo utilizado convergir.

C. Avanço no tempo

Agora faz-se a relação entre a velocidade obtida ignorando-se a pressão com o gradiente da pressão obtida no item anterior para então obter-se o próximo passo de tempo da velocidade. Finalmente, tem-se:

$$\frac{\mathbf{v}^{n+1} - \mathbf{v}^*}{\Delta t} = -\frac{1}{\rho} \nabla p \quad (38)$$

Expandindo para as equações escalares, tem-se:

$$u^{n+1} = u^* - \frac{\Delta t}{\rho} \frac{\partial p}{\partial x} \quad (39)$$

$$v^{n+1} = v^* - \frac{\Delta t}{\rho} \frac{\partial p}{\partial y} \quad (40)$$

Basta discretizar os termos das derivadas parciais agora:

$$p_{x,ij} = \frac{p_{ij} - p_{i-1j}}{\Delta x} \quad (41)$$

$$p_{y,ij} = \frac{p_{ij} - p_{ij-1}}{\Delta x} \quad (42)$$

E finalmente tem-se a solução para o tempo $n + 1$ a partir do tempo passado n :

$$u^{n+1} = u^* - \frac{\Delta t}{\rho} p_x \quad (43)$$

$$v^{n+1} = v^* - \frac{\Delta t}{\rho} p_y \quad (44)$$

Cada uma das variáveis acima é uma matriz. Com isso, tem-se as fórmulas necessárias para resolver a equação de Navier Stokes. No entanto, mais processos são necessários computacionalmente, é necessário escolher uma boa condição de parada para as etapas que consideram convergência, trabalhar com arquivos para salvar os resultados, entre outras coisas, que realmente complicam a solução do problema.

Não há resultados para a equação de Navier Stokes, pois está muito complicado descobrir os bugs ainda existentes no programa. Muita coisa já foi corrigida, mas não há resultado nenhum.

VI. CONCLUSÃO

O problema proposto inicialmente ainda não foi alcançado, que é trabalhar com a instabilidade de Saffman-Taylor com fluido magnético e, ou Anisotrópico. Antes de trabalhar com tal problema, teve-se uma etapa inicial que demandou um certo trabalho. Essa primeira etapa deste projeto consistiu então do estudo da resolução de equações diferenciais parciais, por meio de sua discretização e codificação em um programa de computador.

Foi bem sucedida a resolução das equações de Laplace e Poisson numa malha quadrada, pelos métodos implícito e explícito, usando diferenças finitas. No caso de Poisson, ainda teve-se a resolução bem sucedida com condições de fronteira de Neumann, que seria essencial para trabalhar com a pressão na equação de Navier Stokes.

Chegando na equação de Navier Stokes, houve muitas equações a serem discretizadas, e a malha escalonada foi algo difícil de ser plenamente entendido, tendo muitas vezes ocorrido erros no caminho por causa disso.

O trabalho foi um aprendizado razoavelmente complexo, e que teve muitas consequências positivas. As habilidades de programação de programas numéricos pelo aluno aumentaram bastante, na realidade, de qualquer programa, pelo fato de se ter treinado bastante o uso do git e da ferramenta de compilação cmake. Apesar da programação ainda ser um problema, pela complexidade do programa, o código⁸ está muito mais organizado do que se esperava, e pra isso ele foi refeito algumas vezes.

O programa de Navier Stokes ainda está em fase de desenvolvimento, e se espera ter resultados em breve, com mais um tempo de dedicação e, futuramente, continuar trabalhando com esse programa para um problema físico com ferrofluido.

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.

REFERÊNCIAS

- [1] Rosensweig, R.E. 1987. Magnetic Fluids. Annu. Rev. Fluid Mech. 19:437-461.
- [2] Mittal R, Iaccarino G. 2005. Immersed Boundary Methods. Annu. Rev. Fluid Mech. 37:239-61.
- [3] Hinch, E.J. Lecture notes: Part I - A first problem.

⁸Código está acessível online em <https://github.com/ataias/ff>