PRECONDICIONADOR MULTIESCALA APLICADO À GEOMECÂNICA

Mateus Oliveira de Figueiredo

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Nelson Maculan Filho
Luiz Mariano Paes de Carvalho Filho

Rio de Janeiro
Abril de 2019

PRECONDICIONADOR MULTIESCALA APLICADO À GEOMECÂNICA

Mateus Oliveira de Figueiredo

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Nelson Maculan Filho, D.Sc.

Prof. Luiz Mariano Paes de Carvalho Filho, D.Sc.

Dr. José Roberto Pereira Rodrigues, D.Sc.

Prof. Paulo Goldfeld, D.Sc.

Dr. Rafael Jesus de Moraes, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2019

Figueiredo, Mateus Oliveira de

Precondicionador Multiescala aplicado à Geomecânica/Mateus Oliveira de Figueiredo. – Rio de Janeiro: UFRJ/COPPE, 2019.

XIII, 79 p.: il.; 29, 7cm.

Orientadores: Nelson Maculan Filho

Luiz Mariano Paes de Carvalho Filho

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2019.

Referências Bibliográficas: p. 71 – 73.

1. Multiescala. 2. Sistemas Lineares. 3. Geomecânica. 4. Precondicionador. I. Maculan Filho, Nelson *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha esposa Érika.

Agradecimentos

Gostaria primeiro de agradecer a Deus por sempre ter me dado todas as condições e oportunidades para que eu pudesse obter mais conhecimento.

À minha esposa Érika, que faz a minha vida mais leve e feliz. Me deu todo o apoio durante o mestrado para que eu o fizesse da melhor maneira possível, especialmente na reta final onde o desespero começou a aparecer com mais frequência.

À minha família que sempre me incentivou a seguir em frente, mesmo eu estando distante já há vários anos. Aos meus pais, Marcelo e Flora, por terem me educado, me dado todo o amor e feito de tudo para que eu pudesse chegar até aqui.

Ao meu orientador Nelson Maculan por ter me apresentado a UFRJ tornando possível o meu ingresso no mestrado. E ao meu orientador Luiz Mariano, por todas as reuniões e por todos os ensinamentos passados durante esses dois anos nas reuniões semanais.

Aos meus companheiros de trabalho José Roberto e Felipe por terem me incentivado a fazer o mestrado em tempo parcial pela Petrobras. Ao Rafael, por ter me ajudado com seu conhecimento em métodos multiescala. E a própria Petrobras por investir na minha formação profissional me dando a liberação das horas para que eu concluísse esse trabalho.

Aos meus professores de olimpíada de matemática Marcelo, Cicero e Israel que, por todas as aulas que me deram na adolescência, tornaram meu caminho menos tortuoso na vida acadêmica.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PRECONDICIONADOR MULTIESCALA APLICADO À GEOMECÂNICA

Mateus Oliveira de Figueiredo

Abril/2019

Orientadores: Nelson Maculan Filho
Luiz Mariano Paes de Carvalho Filho

Programa: Engenharia de Sistemas e Computação

Apresenta-se, nesta dissertação, uma aplicação do método mutiescala para o problema Geomecânico de Elasticidade linear. A aplicação consiste em utilizar um precondicionador multiescala junto com um precondicionador no espaço fino para reduzir o número de iterações de solvers lineares como o Gradiente Conjugado e Bicgstab. Os resultados mostram comparações entre a utilização de precondicionadores acoplados de maneira aditiva e de forma multiplicativa mostrando que, em casos que o engrossamento é suficientemente alto, a redução da complexidade do aditivo pode trazer ganho nos tempos de solução. Além disso, é mostrada uma comparação com o solver multigrid sendo ambos os solvers utilizados como precondicionadores para o Gradiente Conjugado.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

MULTISCALE PRECONDITIONER APPLIED TO GEOMECHANICS

Mateus Oliveira de Figueiredo

April/2019

Advisors: Nelson Maculan Filho
Luiz Mariano Paes de Carvalho Filho

Department: Systems Engineering and Computer Science

In this work, we present an application of the multiscale method for the Geomechanical problem of linear elasticity. The application consists of using a multiscale preconditioner with a fine space preconditioner to reduce the number of iterations of linear solvers such as Conjugate Gradient and Bicgstab. The results show comparisons between the use of additive and multiplicative coupled preconditioners showing that, in cases where the coarsening is sufficiently high, the lower complexity of the additive can speed up the solution process. In addition, a comparison with a multigrid solver is shown, both being used as preconditioners for the Conjugate Gradient.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Geomecânica é o estudo das deformações e tensões em solos e rochas. Esse estudo se torna de extrema importância para a explotação de campos de petróleo de maneira segura e mais eficiente. De acordo com ZOBACK [2], os estudos de tensões são importantes, pois:

- Falhas em poços ocorrem por conta de tensões concentradas ao redor da circunferência dos poços.
- Falhas geológicas vão deslizar dependendo das tensões e da sua resistência à fricção.
- Depleções do reservatório causam mudanças nas tensões *in-situ* que podem ser maléficas ou benéficas para a produção.

Além disso, deformações na rocha devido à produção causam subsidência do leito marinho que podem tanto causar problemas ambientais como danificar equipamentos de sub-superfície. Injeção demasiada de água pode reativar falhas ou provocar fraturas na rocha capeadora fazendo com que aconteçam exsudações de óleo na superfície. Assim, fica claro que os estudos de geomecânica são importantes para uma melhor produção do campo, redução de custos de explotação e também para uma operação mais segura. E para a realização desses estudos é importante a utilização de simulações geomecânicas capazes de representar a realidade a fim de evitar problemas no campo.

Os modelos geomecânicos, diferentemente dos modelos de fluxo de reservatório, estão interessados em fenômenos que ocorrem em regiões que são não reservatório como os exemplos de subsidência e tensões ao longo da trajetória do poço do parágrafo anterior. Assim, os modelos necessitam de um domínio de simulação maior que o das simulações de fluxo, contendo regiões de rochas adjacentes ao reservatório. A Figura 1.1 mostra uma seção de um modelo geomecânico em três dimensões. Os elementos pintados de azul são relacionados ao reservatório. Pode-se

ver que este corresponde a uma pequena parte do modelo, isso faz com que o número de elementos dos modelos seja grande, tornando necessárias técnicas numéricas e de computação de alto desempenho com boa performance, para que as simulações possam ser realizadas em tempo viável.

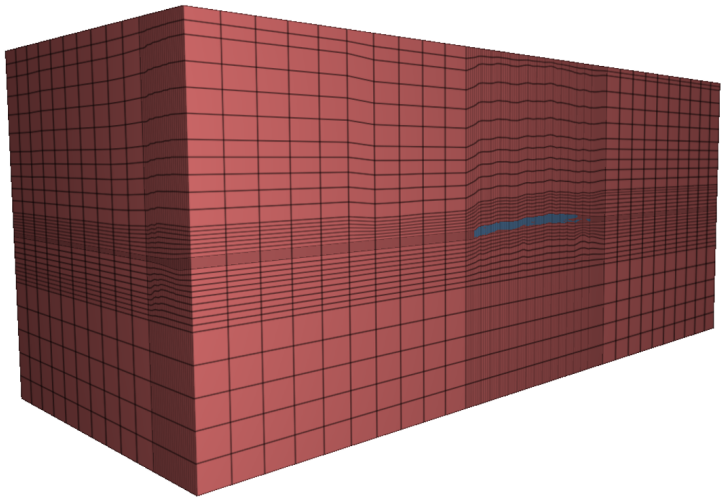


Figura 1.1: Corte vertical de modelo geomecânico 3D. Células em azul representam o reservatório.

Em geral, os simuladores gastam a maior parte do tempo resolvendo sistemas lineares e, portanto, melhorias realizadas nos solvers são bastante impactantes nos tempos das simulações. Tendo isso em vista, essa dissertação tem como intuito avaliar um precondicionador multiescala para reduzir o número de iterações de solver lineares de Krylov, em particular, para o Gradiente Conjugado e o Bicgstab. Ela é organizada da seguinte forma: no Capítulo 2 são apresentadas as equações que regem os fenômenos geomecânicos; no Capítulo 3 é apresentada a discretização realizada nas equações do capítulo anterior através do método dos elementos finitos, chegando a um sistema linear; no Capítulo 4 são apresentadas estruturas de dados para matrizes esparsas e métodos para solução de sistemas lineares, no Capítulo 5 é apresentado o método multiescala e como ele pode ser utilizado como precondicionador para acelerar métodos iterativos de solução dos sistemas lineares; no Capítulo 6 são apresentados detalhes da implementação realizada nesse trabalho e; no Capítulo 7 são apresentados os experimentos numéricos, onde se destacam a comparação entre precondicionador aditivo e multiplicativo e comparação com o método multigrid; no Capítulo 8 são apresentadas as conclusões da dissertação e; o Apêndice A apresentam os grids dos reservatórios utilizados nas execuções e; finalmente, o Apêndice B apresentam os parâmetros utilizados no PyAmg utilizados para a comparação.

Capítulo 2

Modelagem Matemática

Nesse capítulo, serão apresentados as equações que regem os fenômenos geomecânicos. O primeiro trabalho famoso na modelagem de tensões no solo é Terzaghi que desenvolveu a teoria para uma dimensão. Mais tarde, Biot generalizou essa teoria para três dimensões e essas são as equações que serão apresentadas. Detalhes das duas teorias podem ser encontradas em VERRUIJT [1]. Como o intuito do trabalho é avaliar o método multiescala aplicado para matrizes dessa natureza, o caso aqui considerado será o de elasticidade linear onde a rocha não é tensionada até que apresente falhas.

2.1 Tensor de Tensões

Para representar a tensão total em um ponto da rocha, é utilizado um tensor de tensões de segunda ordem como mostra (2.1).

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix} \tag{2.1}$$

O primeiro subscrito do tensor representa a face que a tensão está sendo aplicada, enquanto o segundo representa a direção da tensão. A Figura 2.1 mostra as componentes com o com primeiro subscrito x.

Ao aplicar a condição de equilíbrio do momento, chega-se a conclusão que $\sigma_{xy} = \sigma_{yx}$, $\sigma_{xz} = \sigma_{zx}$ e $\sigma_{yz} = \sigma_{zy}$. Dessa maneira, para a representação desse tensor, são necessários guardar apenas seis valores. Pode-se considerar então a tensão como o vetor apresentado (2.2) essa notação é chamada de notação de Voigt e é a bastante utilizada nas implementações de elementos finitos, por exemplo, nas formulações apresentadas por HUGHES [4] e FISH e BELYTSCHKO [5].

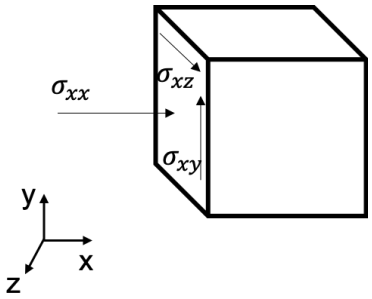


Figura 2.1: Tensões $\sigma_{x.}$ representadas graficamente.

$$\boldsymbol{\sigma}^T = \begin{bmatrix} \sigma_{xx} & \sigma_{yy} & \sigma_{zz} & \sigma_{xy} & \sigma_{xz} & \sigma_{yz} \end{bmatrix} \tag{2.2}$$

2.2 Teoria da Consolidação

Para um certo elemento de volume $\Delta x \Delta y \Delta z$ pode-se escrever as equações de equilíbrio para cada uma dessas direções. A Figura 2.2 apresenta todas as tensões relativas à direção x.

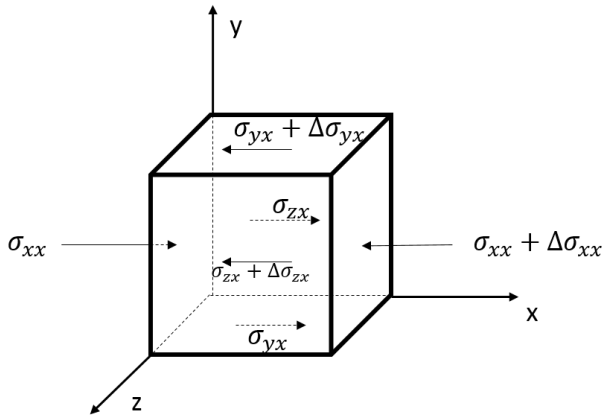


Figura 2.2: Tensões na direção x ($\sigma_{.x}$) representadas graficamente. Adaptado de [1].

As forças atuantes no cubo são calculadas a partir da tensão multiplicada pela área de atuação. Sobre o valor f_x, ele, juntamente com os termos f_y e f_z são termos gravitacionais e representam a força peso do volume $\Delta x \Delta y \Delta z$. Como apresentado em GAI [6], $\mathbf{f} = [\rho_s(1-\phi) + \phi(\rho_o S_o + \rho_w S_w + \rho_g S_g)]\mathbf{g}$ onde ϕ é a porosidade, S_o, S_w e S_g as saturações de óleo, água e gás e ρ_o, ρ_w, ρ_g, ρ_s as densidades do óleo, água, gás e rocha e $\mathbf{g}$ o vetor gravidade. O termo gravitacional será considerado constante ao longo da simulação, fazendo com que as modificações no equilíbrio sejam apenas relativas à mudança de pressão.

O equilíbrio de forças na direção x pode ser escrito como apresentado em (2.3).

$$(\sigma_{xx} - \sigma_{xx} - \Delta\sigma_{xx})\Delta y\Delta z + (\sigma_{yx} - \sigma_{yx} - \Delta\sigma_{yx})\Delta x\Delta z + \\ + (\sigma_{zx} - \sigma_{zx} - \Delta\sigma_{zx})\Delta x\Delta y + f_x\Delta x\Delta y\Delta z = 0 \quad (2.3)$$

$$\Delta\sigma_{xx}\Delta y\Delta z + \Delta\sigma_{yx}\Delta x\Delta z + \Delta\sigma_{zx}\Delta x\Delta y - f_x\Delta x\Delta y\Delta z = 0 \quad (2.4)$$

Dividindo todos os termos por $\Delta x\Delta y\Delta z$.

$$\frac{\Delta\sigma_{xx}}{\Delta x} + \frac{\Delta\sigma_{yx}}{\Delta y} + \frac{\Delta\sigma_{zx}}{\Delta z} - f_x = 0 \quad (2.5)$$

Fazendo $\Delta x \to 0$, $\Delta y \to 0$ e $\Delta z \to 0$

$$\frac{\partial\sigma_{xx}}{\partial x} + \frac{\partial\sigma_{yx}}{\partial y} + \frac{\partial\sigma_{zx}}{\partial z} - f_x = 0 \quad (2.6)$$

Analogamente, para as direções y e z pode-se encontrar a equação de equilíbrio também, montando o sistema (2.7).

$$\begin{cases} \frac{\partial\sigma_{xx}}{\partial x} + \frac{\partial\sigma_{yx}}{\partial y} + \frac{\partial\sigma_{zx}}{\partial z} - f_x &= 0 \\ \frac{\partial\sigma_{xy}}{\partial x} + \frac{\partial\sigma_{yy}}{\partial y} + \frac{\partial\sigma_{zy}}{\partial z} - f_y &= 0 \\ \frac{\partial\sigma_{xz}}{\partial x} + \frac{\partial\sigma_{yz}}{\partial y} + \frac{\partial\sigma_{zz}}{\partial z} - f_z &= 0 \end{cases} \quad (2.7)$$

As tensões apresentadas em (2.7) são as tensões totais que atuam no cubo infinitesimal. Acontece que, como mostra a Figura 2.3, os reservatórios de petróleo possuem fluído no volume poroso da rocha (óleo, água e gás) e, portanto, parte da tensão total será suportada pelo fluído e parte será suportada pelos grãos da rocha. Como o fluído não oferece resistência ao cisalhamento, ele suporta apenas a parte das tensões σ_{xx}, σ_{yy} e σ_{zz}.

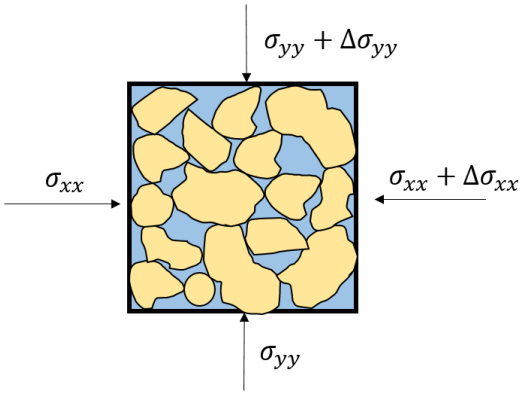


Figura 2.3: Representação de meio poroso. Grãos amarelos representando a matriz da rocha e fluído representado pela cor azul.

Conforme mostrado pela teoria da consolidação de poroelasticidade, desenvolvida

por Biot para três dimensões, a tensão efetiva na rocha ($\boldsymbol{\sigma}''$) e a tensão total são relacionadas por (2.8), conforme mostrado em ZOBACK [2].

$$\begin{cases} \sigma''_{xx} = \sigma_{xx} - \alpha P_p \\ \sigma''_{yy} = \sigma_{yy} - \alpha P_p \\ \sigma''_{zz} = \sigma_{zz} - \alpha P_p \end{cases} \tag{2.8}$$

Onde σ''_{xx}, σ''_{yy} e σ''_{zz} são as tensões efetivas, α é o coeficiente de Biot e P_p a pressão de poros. O coeficiente de Biot representa o quanto a pressão de poros do fluído suporta a tensão total na rocha e está no intervalo $\alpha \in [0, 1]$.

As equações (2.7) podem ser reescritas como (2.9) substituindo as tensões totais ($\boldsymbol{\sigma}$) pela tensões efetivas na rocha ($\boldsymbol{\sigma}''$).

$$\begin{cases} \frac{\partial \sigma''_{xx}}{\partial x} + \frac{\partial \sigma''_{yx}}{\partial y} + \frac{\partial \sigma''_{zx}}{\partial z} - \frac{\partial \alpha P_p}{\partial x} - f_x = 0 \\ \frac{\partial \sigma''_{xy}}{\partial x} + \frac{\partial \sigma''_{yy}}{\partial y} + \frac{\partial \sigma''_{zy}}{\partial z} - \frac{\partial \alpha P_p}{\partial y} - f_y = 0 \\ \frac{\partial \sigma''_{xz}}{\partial x} + \frac{\partial \sigma''_{yz}}{\partial y} + \frac{\partial \sigma''_{zz}}{\partial z} - \frac{\partial \alpha P_p}{\partial z} - f_z = 0 \end{cases} \tag{2.9}$$

Ou ainda, escrevendo (2.9) utilizando os operadores divergente, gradiente e $\mathbf{f}^T = \begin{bmatrix} f_x & f_y & f_z \end{bmatrix}$ pode-se chegar em (2.10).

$$\nabla \cdot \boldsymbol{\sigma}'' - \nabla \cdot \alpha \mathbf{I} P_p - \mathbf{f} = 0 \tag{2.10}$$

Por motivos de implementação mais eficiente, menor uso de memória e utilização de operações mais simples (multiplicação de matrizes por vetores), a Equação (2.10) pode ser escrita na notação de Voigt considerando as definições conforme (2.12).

$$\nabla_S^T \boldsymbol{\sigma}'' - \nabla_S^T \alpha \mathbf{m} P_p - \mathbf{f} = 0 \tag{2.11}$$

Onde,

$$\boldsymbol{\sigma}'' = \begin{bmatrix} \sigma''_{xx} \\ \sigma''_{yy} \\ \sigma''_{zz} \\ \sigma''_{xy} \\ \sigma''_{xz} \\ \sigma''_{yz} \end{bmatrix} \quad ; \quad \mathbf{f} = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \quad ; \quad \mathbf{m} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad ; \quad \nabla_S = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \end{bmatrix} \tag{2.12}$$

2.2.1 Relações Constitutivas

A Equação (2.11) apresenta o equilíbrio em função das tensões. Porém, pelo grande número de variáveis dessa equação, é necessário colocar em função das variáveis de deslocamento para que seja possível resolve-la. O campo de deslocamentos será representado pela variável $\mathbf{u} = [u_x \quad u_y \quad u_z]^T$. A deformação ($\boldsymbol{\epsilon}$) se relaciona com o

deslocamento de acordo com (2.13).

$$\boldsymbol{\epsilon} = \nabla_S \mathbf{u} \tag{2.13}$$

Já a relação entre a deformação e as tensões é definida como relação constitutiva de acordo com ZOBACK [2]. Diferentes tipos de relações constitutivas podem ser utilizadas para representar essa relação entre tensão e deformação. A Figura 2.4 mostra dados de um teste típico de tensão-deformação em uma rocha bem cimentada. Nesse caso, é importante notar que existe um comportamento linear dominante na parte central do gráfico antes da falha. Essa região onde as deformações e tensões se relacionam linearmente é chamada de elástica, e será a região de estudo desse trabalho.

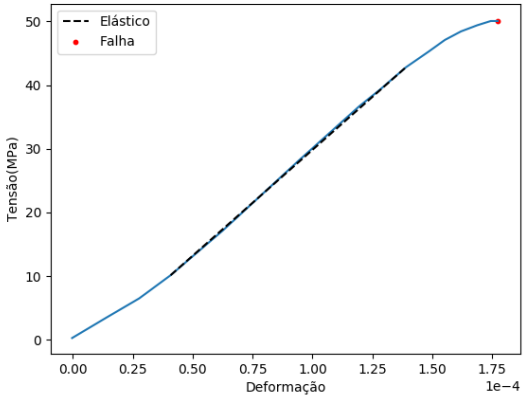


Figura 2.4: Teste de laboratório tensão-deformação para uma rocha bem cimentada. Adaptada de ZOBACK [2].

Para o caso de elasticidade linear, a relação entre tensões e deformação é dada por uma forma simples que é a Lei de Hooke Generalizada apresentada em (2.14). Essa equação é a mesma que apresentada a para estudos de mecânica dos sólidos clássicos.

$$\boldsymbol{\sigma}'' = \mathbf{D}\boldsymbol{\epsilon} \tag{2.14}$$

$$\mathbf{D} = \frac{E}{(1+\upsilon)(1-2\upsilon)} \begin{bmatrix} 1-\upsilon & \upsilon & \upsilon & 0 & 0 & 0 \\ \upsilon & 1-\upsilon & \upsilon & 0 & 0 & 0 \\ \upsilon & \upsilon & 1-\upsilon & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\upsilon}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\upsilon}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\upsilon}{2} \end{bmatrix} \tag{2.15}$$

onde, E é o módulo de Young e υ o módulo de Poisson da rocha.

A EDP da equação (2.11) pode então ser escrita em função dos deslocamentos, inserindo (2.13) e (2.14) em (2.11).

$$\nabla_S^T \mathbf{D} \nabla_S \mathbf{u} - \nabla_S^T \alpha \mathbf{m} P_p - \mathbf{f} = 0 \tag{2.16}$$

Essa forma será a utilizada junto dos métodos do elementos finitos para construção de um simulador para regime permanente de geomecânica em duas dimensões. Ao se passar (2.16) de três dimensões para duas existem duas abordagens possíveis: tensão plana ou deformação plana. As matrizes de elasticidade são apresentadas respectivamente em (2.17) e (2.18).

$$\mathbf{D}_{stress} = \frac{E}{1-\upsilon^2} \begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \frac{1-\upsilon}{2} \end{bmatrix} \tag{2.17}$$

$$\mathbf{D}_{strain} = \frac{E}{(1+\upsilon)(1-2\upsilon)} \begin{bmatrix} 1-\upsilon & \upsilon & 0 \\ \upsilon & 1-\upsilon & 0 \\ 0 & 0 & \frac{1-2\upsilon}{2} \end{bmatrix} \tag{2.18}$$

De acordo com FISH e BELYTSCHKO [5], a condição de deformação plana é melhor aplicada quando o elemento é grosso em relação ao plano xy , que é o caso dos reservatórios de petróleo com os mesmos eixos definidos na Figura 2.2. Dessa forma, artigos como [3], [7] e [8] utilizam a hipótese de deformação plana que é a mesma utilizada nessa dissertação. Os operadores e vetores podem ser redefinidos para os mostrados em (2.19).

$$\boldsymbol{\sigma}'' = \begin{bmatrix} \sigma''_{xx} \\ \sigma''_{yy} \\ \sigma''_{xy} \end{bmatrix} \quad ; \quad \mathbf{f} = \begin{bmatrix} f_x \\ f_y \end{bmatrix} \quad ; \quad \mathbf{m} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad ; \quad \nabla_S = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \quad ; \quad \mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{2.19}$$

Além de (2.16), são necessárias as condições de contorno para que o problema fique totalmente definido. Existem dois principais tipos de condição de contorno a de Dirichlet, onde o deslocamento da fronteira é prescrito e condição de contorno de Neumann, onde a tensão normal a fronteira é prescrita. Considerando Ω o domínio em que está sendo resolvido o problema e $\Gamma = \partial\Omega$ a fronteira do domínio, pode-se dividi-la em duas partes Γ_u e Γ_σ, onde $\Gamma_u \cup \Gamma_\sigma = \Gamma$, $\Gamma_u \cap \Gamma_\sigma = \emptyset$, que possuem condição de Dirichlet e Neumann respectivamente. O problema fica então totalmente definido em (2.20).

$$\begin{aligned} \nabla_S^T \mathbf{D} \nabla_S \mathbf{u} - \nabla_S^T \alpha \mathbf{m} P_p - \mathbf{f} = 0, \text{ em } \Omega \\ \mathbf{u} = \bar{\mathbf{u}}, \text{ em } \Gamma_u \\ \begin{bmatrix} \sigma''_{xx} & \sigma''_{xy} \end{bmatrix} \mathbf{n} = \bar{t}_x \quad \begin{bmatrix} \sigma''_{xy} & \sigma''_{yy} \end{bmatrix} \mathbf{n} = \bar{t}_y, \text{ em } \Gamma_\sigma \end{aligned} \tag{2.20}$$

onde, $\mathbf{n}$ representa o vetor normal a Γ, $\bar{\mathbf{u}} \in \mathbb{R}^2$ e $\bar{\mathbf{t}} = [\bar{t}_x \quad \bar{t}_y]$. O problema (2.20) pode ser descrito mais genericamente com regiões da fronteira onde uma condição de contorno de Dirichlet é aplicada apenas ao campo u_x ou u_y enquanto o outro campo tem condição de Neumann. Esse exemplo mais geral pode ser encontrado em HUGHES [4].

Capítulo 3

Discretização

3.1 Modelagem pelo Método dos Elementos Finitos

Em posse de (2.20), é necessário um método numérico para encontrar sua solução. O método utilizado nesse trabalho é o dos Elementos Finitos que irá transformar (2.20) em um sistema linear. De acordo com [5], o método dos Elementos finitos é bastante difundido, ele é utilizado, por exemplo, em análise de tensões, transferência de calor, escoamento de fluido e eletromagnetismo. Em particular, a parte de análise de tensões e escoamento de fluído são importantes para a Engenharia de Reservatórios.

3.1.1 Formulação Fraca

O primeiro passo para utilizar o método dos elementos finitos é chegar na formulação fraca de (2.20). As referências [4] e [5] mostram a dedução para chegar em (3.1) e provam também a equivalência dela para a forma original (chamada de forma forte) com exceção do termo relacionado à pressão de poros, pois tratam do problema clássico de elasticidade linear. A forma fraca com esse termo pode ser encontrado em LEWIS e SCHREFLER [9]. O termo relativo ao peso foi removido, pois as análises dessa dissertação serem relativas as mudanças no equilíbrio por conta da variação da pressão de poros. A notação utilizada aqui é baseada em [3], para manter a coerência com o Capítulo 5.

$$\begin{aligned} &\text{Encontrar } \mathbf{u} \in \mathcal{S}(\Omega) \text{ tal que} \\ &\int_{\Omega} (\nabla_S \mathbf{w})^T \mathbf{D} \nabla_S \mathbf{u} \, d\Omega - \int_{\Gamma_\sigma} \mathbf{w}^T \bar{\mathbf{t}} d\Gamma = \int_{\Omega} (\nabla_S \mathbf{w})^T \mathbf{m} P_p \, d\Omega \quad \forall \mathbf{w} \in \mathcal{V}(\Omega) \end{aligned} \tag{3.1}$$

Com conjunto de teste $\mathcal{V}(\Omega) = \{\mathbf{w}|\mathbf{w} \in H^1(\Omega)^2, \mathbf{w} = 0 \text{ em } \Gamma_u\}$ e conjunto de avaliação igual a $\mathcal{S}(\Omega) = \{\mathbf{u}|\mathbf{u} \in H^1(\Omega)^2, \mathbf{u} = \bar{\mathbf{u}} \text{ em } \Gamma_u\}$. Onde $H^1(\Omega)$ representa o espaço de Sobolev de grau um.

3.1.2 Divisão do domínio

O domínio do problema será dividido em uma quantidade finita de elementos, o conjunto desses elementos será chamado de τ^h. A partição do domínio será realizada em elementos quadriláteros e seus vértices são denominados nós. A Figura 3.1 mostra um exemplo dessa divisão para um determinado domínio.

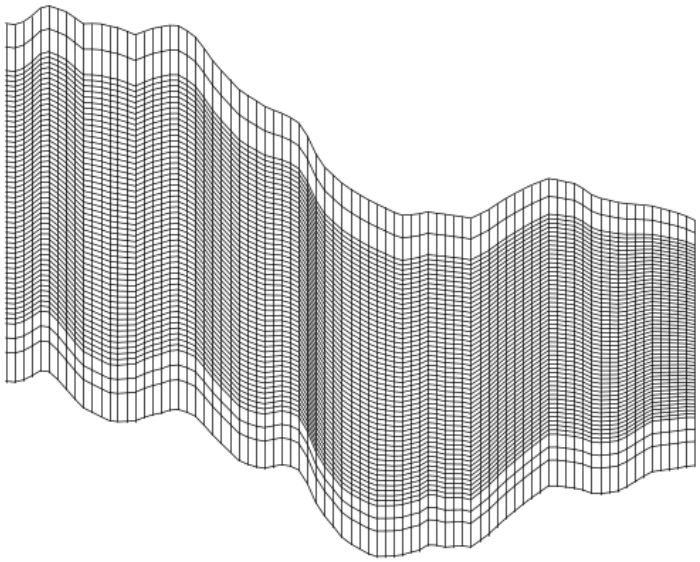


Figura 3.1: Exemplo de domínio/reservatório dividido em quadriláteros.

Dadas a forma fraca definida em (3.1) e funções de forma $\mathbf{N}_i^h$ para $i = 1, 2, 3, ..., n_u^h + n_{\bar{u}}^h$, onde n_u^h representa a quantidade de graus de liberdade e $n_{\bar{u}}^h$ está relacionado com os graus de liberdade que caíram em condições de contorno de Dirichlet, pode-se encontrar uma solução aproximada pelo método de elementos finitos de Galerkin. O método consiste em procurar solução não mais para $\mathbf{w} \in \mathcal{V}(\Omega)$ e $\mathbf{u} \in \mathcal{S}(\Omega)$, mas em encontrar uma solução aproximada onde $\mathbf{w^h} \in \mathcal{V}^h = \text{span}\{\mathbf{N}_i^h | i = 1, 2, ..., n_u^h\}$ e $\mathbf{u}$ é da forma aproximada mostrada em (3.2).

$$\mathbf{u^h}(\mathbf{x}) = \sum_{i=1}^{n_u^h} \mathbf{N}_i^h d_i^h + \sum_{i=1}^{n_{\bar{u}}^h} \mathbf{N}_{i+n_u^h}^h \bar{d}_i^h \tag{3.2}$$

Onde cada d_i^h representa um grau de liberdade associado a um nó que, a princípio, tem dois graus de liberdade, um para a direção x e outro para a direção y, a menos dos

nós que estão em fronteiras com condição de contorno de Dirichlet. Nesse caso, os nós terão associados valores $\bar{d}_i^h$ iguais ao valor especificado pela condição para garantir que elas sejam satisfeitas. É importante perceber que, com essas aproximações, a busca da solução está sendo realizada em espaços de dimensão finita, diferentemente da forma fraca original onde os espaços $\mathcal{V}(\Omega)$ e $\mathcal{S}(\Omega)$ são infinitos.

Com isso pode-se chegar na forma fraca aproximada (3.3). Caso a solução do problema original pertença a $\mathcal{V}^h$ a solução exata ainda será encontrada pelo método.

$$\int_\Omega (\nabla_S \mathbf{w}^h)^T \mathbf{D} \nabla_S \mathbf{u}^h \, d\Omega = \int_{\Gamma_\sigma} \mathbf{w}^{h^T} \bar{\mathbf{t}} d\Gamma + \int_\Omega (\nabla_S \mathbf{w}^h)^T \mathbf{m} P_p \, d\Omega \quad \forall \mathbf{w}^{h^T} \in \mathcal{V}^h \tag{3.3}$$

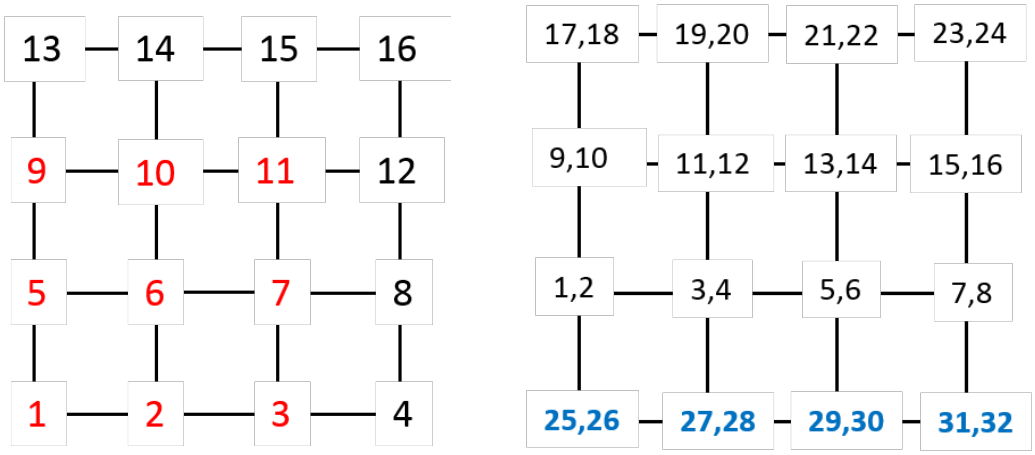
A Equação (3.2) também pode ser reescrita utilizando vetores,

$$\mathbf{u^h}(\mathbf{x}) = \mathbf{N^h}\mathbf{d}^h + \bar{\mathbf{N}}^\mathbf{h}\bar{\mathbf{d}}^h \tag{3.4}$$

onde

$$\begin{aligned} \mathbf{N^h} &= [\mathbf{N}_1^h, \mathbf{N}_2^h, ..., \mathbf{N}_{n_u^h}^h] \\ \bar{\mathbf{N}}^\mathbf{h} &= [\mathbf{N}_{n_u^h+1}^h, \mathbf{N}_{n_u^h+2}^h, ..., \mathbf{N}_{n_u^h+n_{\bar{u}}^h}^h] \\ \mathbf{d} &= [d_1^h, d_2^h, ..., d_{n_u^h}^h]^T \\ \bar{\mathbf{d}} &= [\bar{d}_1^h, \bar{d}_2^h, ..., \bar{d}_{n_{\bar{u}}^h}^h]^T \end{aligned}$$

As funções $\mathbf{N}_i^h : \Omega \to \mathbb{R}^2$, em uma abordagem clássica de elementos finitos, são da forma $[N_{i_x}^h \quad 0]^T$ para um grau de liberdade x e $[0 \quad N_{i_y}^h]^T$ para um grau de liberdade y, portanto, um grau de liberdade x não é utilizado para interpolar o valor do campo $u_y(x, y)$ e vice-versa. Além disso, para um nó com graus de liberdade d_i^h para x e d_j^h para y, tem-se $N_{i_x}^h = N_{j_y}^h$ que, de acordo com [5] é a forma mais usual. As funções $N_{i_x}^h$ e $N_{j_y}^h$ são tais que assumem valor um no seu nó correspondente e possuem valor zero em todos os outros nós, mais especificamente, o suporte dessas funções são apenas os elementos que possuem o nó correspondente como vértice e são bilineares em cada um desses elementos (bilineares por partes). A Figura 3.2(a) mostra a numeração dos nós em uma malha 3×3 enquanto a Figura 3.2(b) apresenta a numeração das funções $\mathbf{N}_i^h$ considerando condições de Dirichlet na borda inferior. Exemplos das funções são apresentadas na Figura 3.3. Um detalhe importante é que as funções de forma que serão utilizadas no método multiescala, apresentadas no Capítulo 5, não possuem a propriedade de uma de suas componentes ser nula.



(a) Numeração global dos nós em malha 3×3. Marcados de vermelho o nó 6 e seus adjacentes.

(b) Numeração das funções de forma N_i^h. Azul as funções de forma relativa à condição de contorno de Dirichlet

Figura 3.2: Numeração dos nós e das funções de base. Para numeração das funções de base foi considerada a borda inferior com condição de Dirichlet nas direções x e y.

3.1.3 Construção do sistema linear

É importante perceber que as variáveis a serem descobertas são as do vetor $\mathbf{d}^h$, dado que as entradas de $\bar{\mathbf{d}}^h$ já estão definidas pelos valores da fronteira. Os valores de $\mathbf{d}^h$ são encontrados a partir da solução de um sistema linear.

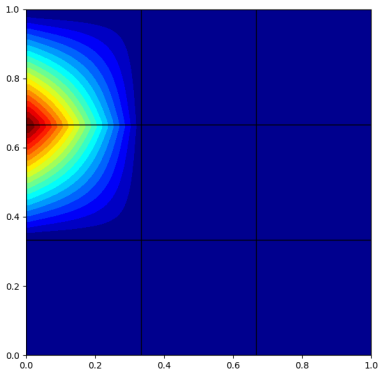
Para obter um sistema linear, pode-se substituir $\mathbf{w}^h$ por cada uma das funções $\mathbf{N}_i^h$ para $i = 1, 2, 3, ..., n_u^h$ em (3.3). Seguem abaixo os cálculos substituindo $\mathbf{w}^h$ por $\mathbf{N}_i^h$ e $\mathbf{u}^h$ por (3.4).

$$\int_\Omega (\nabla_S \mathbf{N}_i^h)^T \mathbf{D} \nabla_S (\mathbf{N^h d}^h + \bar{\mathbf{N}}^\mathbf{h} \bar{\mathbf{d}}^h)\, d\Omega = \int_{\Gamma_\sigma} \mathbf{N}_i^h \bar{\mathbf{t}} d\Gamma + \int_\Omega (\nabla_S \mathbf{N}_i^h)^T \mathbf{m} P_p \, d\Omega \tag{3.5}$$

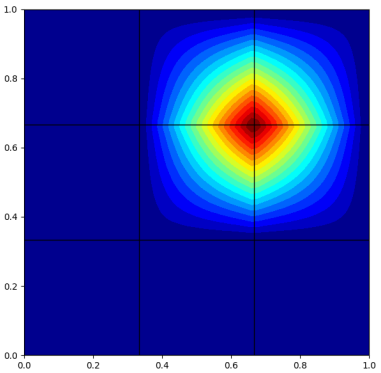
$$\int_\Omega (\nabla_S \mathbf{N}_i^h)^T \mathbf{D} \nabla_S \mathbf{N^h d}^h \, d\Omega = \int_{\Gamma_\sigma} \mathbf{N}_i^h \bar{\mathbf{t}} d\Gamma + \int_\Omega (\nabla_S \mathbf{N}_i^h)^T \mathbf{m} P_p \, d\Omega - \int_\Omega (\nabla_S \mathbf{N}_i^h)^T \mathbf{D} \nabla_S \bar{\mathbf{N}}^\mathbf{h} \bar{\mathbf{d}}^h \, d\Omega$$

$$\int_\Omega (\nabla_S \mathbf{N}_i^h)^T \mathbf{D} \nabla_S \mathbf{N^h}\, d\Omega \mathbf{d}^h = \int_{\Gamma_\sigma} \mathbf{N}_i^h \bar{\mathbf{t}} d\Gamma + \int_\Omega (\nabla_S \mathbf{N}_i^h)^T \mathbf{m} P_p \, d\Omega - \int_\Omega (\nabla_S \mathbf{N}_i^h)^T \mathbf{D} \nabla_S \bar{\mathbf{N}}^\mathbf{h}\, d\Omega \bar{\mathbf{d}}^h \tag{3.6}$$

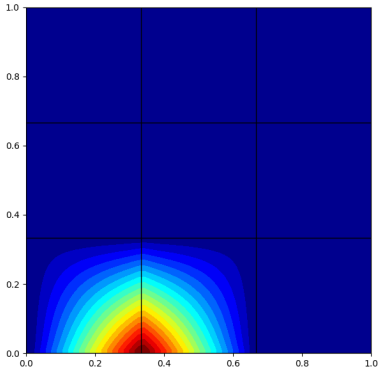
Definindo,



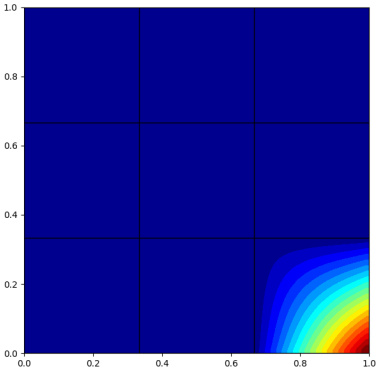
(a) $\mathbf{N}^h_{9x}$ ou $\mathbf{N}^h_{10y}$



(b) $\mathbf{N}^h_{13x}$ ou $\mathbf{N}^h_{14y}$



(c) $\mathbf{N}^h_{27x}$ ou $\mathbf{N}^h_{28y}$



(d) $\mathbf{N}^h_{31x}$ ou $\mathbf{N}^h_{32y}$

Figura 3.3: Gráficos das funções de forma para um grid 3x3. Foi considerada a mesma numeração apresentada na Figura 3.2(b).

$$[\mathbf{K}^h]_{i,j} = \int_\Omega (\nabla_S \mathbf{N}^h_i)^T \mathbf{D} \nabla_S \mathbf{N}^h_j \, d\Omega \tag{3.7}$$

e

$$[\mathbf{f}^h]_i = \int_{\Gamma_\sigma} \mathbf{N}^h_i \bar{\mathbf{t}} d\Gamma + \int_\Omega (\nabla_S \mathbf{N}^h_i)^T \mathbf{m} P_p \, d\Omega - \int_\Omega (\nabla_S \mathbf{N}^h_i)^T D \nabla_S \bar{\mathbf{N}}^\mathbf{h} \, d\Omega \bar{\mathbf{d}}^h \tag{3.8}$$

A parcela da integral do lado esquerdo pode ser reescrita como:

$$\int_\Omega (\nabla_S \mathbf{N}^h_i)^T \mathbf{D} \nabla_S \mathbf{N}^\mathbf{h} \, d\Omega = [[\mathbf{K}^h]_{i,1} \quad [\mathbf{K}^h]_{i,2} \quad ... \quad [\mathbf{K}^h]_{i,n^h_u}] = [\mathbf{K}^h]_{i,:} \tag{3.9}$$

e então chegar em (3.10).

$$[\mathbf{K}^h]_{i,:} \mathbf{d}^h = [\mathbf{f}^h]_i \quad \forall \quad i = 1, ..., n^h_u \tag{3.10}$$

Finalmente, substituindo $i = 1, 2, ..., n_u^h$ pode-se encontrar o sistema linear (3.11).

$$\mathbf{K}^h \mathbf{d}^h = \mathbf{f}^h \tag{3.11}$$

onde as entradas de $\mathbf{K}^h$ são definidas por (3.7). A solução do sistema tem como resultado os valores de cada um dos graus de liberdade d_i^h e, portanto, é possível encontrar a aproximação dos deslocamentos com (3.2). Sobre a matriz $\mathbf{K}^h$, ela satisfaz as seguintes propriedades:

- A matriz é simétrica, pois, como $\mathbf{D}$ é simétrica:

$$\begin{aligned}
[\mathbf{K}^h]_{i,j} &= ([\mathbf{K}^h]_{i,j})^T && (3.12)\\
&= \int_\Omega ((\nabla_S \mathbf{N}_i^h)^T D \nabla_S \mathbf{N}_j^h)^T \, d\Omega && (3.13)\\
&= \int_\Omega (\nabla_S \mathbf{N}_j^h)^T D^T ((\nabla_S \mathbf{N}_i^h)^T)^T \, d\Omega && (3.14)\\
&= \int_\Omega (\nabla_S \mathbf{N}_j^h)^T D (\nabla_S \mathbf{N}_i^h) \, d\Omega && (3.15)\\
&= [\mathbf{K}^h]_{j,i} && (3.16)
\end{aligned}$$

- A matriz é esparsa. Pois cada uma das funções de forma é não nula apenas nos elementos em que aquele nó é vértice e, portanto, a intersecção dos suportes de $\mathbf{N}_i^h$ e $\mathbf{N}_j^h$ é não vazia somente se estiverem associadas a vértices que estão em um mesmo elemento. No caso de uma malha de quadriláteros, cada função de forma é não nula em quatro elementos apenas.

- A matriz é positiva definida. A prova desse fato é encontrada em [4], que mostra que esse propriedade depende da relação constitutiva ser também positiva definida.

Ainda sobre a esparsidade da matriz de rigidez em um grid com quadriláteros, cada função de base de um nó (a menos de nós de fronteira) tem conexão com nove nós, contando com ele mesmo. Como mostra a Figura 3.2(a) o nó 6 tem conexões com os nós [1,2,3,5,6,7,9,10,11]. Considerando a simplificação que cada nó possui os dois graus de liberdade isso implica em duas linhas correspondentes na matriz de rigidez, e cada linha possui $2 \times 9 = 18$ valores não nulos, portanto, cada nó tem $2 \times 18 = 36$ não zeros associados na matriz, o que leva a uma aproximação para o número de não nulos da matriz de $\text{nnz}_{\mathbf{K}^h} = 36 n_{\text{nós}}$. Como será mostrado no Capítulo 4, a matriz terá que utilizar alguma estrutura de matriz esparsa, pois

caso fosse alocada densa a quantidade de valores alocados seria aproximadamente $(2 \times n_{\text{nós}})^2$ que tem ordem quadrática com o número de nós e limitaria rapidamente o tamanho dos modelos simulados.

Apesar de ter sido mostrado explicitamente quanto vale cada entrada em (3.7) a montagem da matriz não é feita dessa forma. As integrais que aparecem no domínio Ω serão divididas em cada um dos elementos passando da forma (3.17) para (3.18).

$$\mathbf{K}^h = \int_{\Omega} \begin{bmatrix} (\nabla_S \mathbf{N}_1^h)^T \mathbf{D} \nabla_S \mathbf{N}_1^h & (\nabla_S \mathbf{N}_1^h)^T \mathbf{D} \nabla_S \mathbf{N}_2^h & \dots & (\nabla_S \mathbf{N}_1^h)^T \mathbf{D} \nabla_S \mathbf{N}_{n_u^h}^h \\ (\nabla_S \mathbf{N}_2^h)^T \mathbf{D} \nabla_S \mathbf{N}_1^h & (\nabla_S \mathbf{N}_2^h)^T \mathbf{D} \nabla_S \mathbf{N}_2^h & \dots & (\nabla_S \mathbf{N}_2^h)^T \mathbf{D} \nabla_S \mathbf{N}_{n_u^h}^h \\ \vdots & & \ddots & \vdots \\ (\nabla_S \mathbf{N}_{n_u^h}^h)^T \mathbf{D} \nabla_S \mathbf{N}_1^h & (\nabla_S \mathbf{N}_{n_u^h}^h)^T \mathbf{D} \nabla_S \mathbf{N}_2^h & \dots & (\nabla_S \mathbf{N}_{n_u^h}^h)^T \mathbf{D} \nabla_S \mathbf{N}_{n_u^h}^h \end{bmatrix} d\Omega \tag{3.17}$$

$$\mathbf{K}^h = \sum_{e \in \tau^h} \int_{\Omega^e} \begin{bmatrix} (\nabla_S \mathbf{N}_1^h)^T \mathbf{D} \nabla_S \mathbf{N}_1^h & (\nabla_S \mathbf{N}_1^h)^T \mathbf{D} \nabla_S \mathbf{N}_2^h & \dots & (\nabla_S \mathbf{N}_1^h)^T \mathbf{D} \nabla_S \mathbf{N}_{n_u^h}^h \\ (\nabla_S \mathbf{N}_2^h)^T \mathbf{D} \nabla_S \mathbf{N}_1^h & (\nabla_S \mathbf{N}_2^h)^T \mathbf{D} \nabla_S \mathbf{N}_2^h & \dots & (\nabla_S \mathbf{N}_2^h)^T \mathbf{D} \nabla_S \mathbf{N}_{n_u^h}^h \\ \vdots & & \ddots & \vdots \\ (\nabla_S \mathbf{N}_{n_u^h}^h)^T \mathbf{D} \nabla_S \mathbf{N}_1^h & (\nabla_S \mathbf{N}_{n_u^h}^h)^T \mathbf{D} \nabla_S \mathbf{N}_2^h & \dots & (\nabla_S \mathbf{N}_{n_u^h}^h)^T \mathbf{D} \nabla_S \mathbf{N}_{n_u^h}^h \end{bmatrix} d\Omega^e \tag{3.18}$$

Cada parcela da somatório dos elementos em (3.18) é uma integral no domínio Ω^e de algum elemento e, portanto, apenas 8 funções de forma são não zero em Ω^e. Então, apenas $8 \times 8 = 64$ valores são diferentes de zero em cada uma das matrizes do somatório. Assim, esse valores podem ser condensados em uma matriz menor 8×8 com uma numeração local do elemento conforme mostrado em (3.19).

$$\mathbf{K}^e = \int_{\Omega^e} \begin{bmatrix} (\nabla_S \mathbf{N}_1^e)^T \mathbf{D}^e \nabla_S \mathbf{N}_1^e & (\nabla_S \mathbf{N}_1^e)^T \mathbf{D}^e \nabla_S \mathbf{N}_2^e & \dots & (\nabla_S \mathbf{N}_1^e)^T \mathbf{D}^e \nabla_S \mathbf{N}_8^e \\ (\nabla_S \mathbf{N}_2^e)^T \mathbf{D}^e \nabla_S \mathbf{N}_1^e & (\nabla_S \mathbf{N}_2^e)^T \mathbf{D}^e \nabla_S \mathbf{N}_2^e & \dots & (\nabla_S \mathbf{N}_2^e)^T \mathbf{D}^e \nabla_S \mathbf{N}_8^e \\ \vdots & & \ddots & \vdots \\ (\nabla_S \mathbf{N}_8^e)^T \mathbf{D}^e \nabla_S \mathbf{N}_1^e & (\nabla_S \mathbf{N}_8^e)^T \mathbf{D}^e \nabla_S \mathbf{N}_2^e & \dots & (\nabla_S \mathbf{N}_8^e)^T \mathbf{D}^e \nabla_S \mathbf{N}_8^e \end{bmatrix} d\Omega^e \tag{3.19}$$

Onde $\mathbf{N}_1^e, \dots, \mathbf{N}_8^e$ representam uma numeração local do elemento para as funções $\mathbf{N}_i^h$ que são não nulas nesse determinado elemento e $\mathbf{D}^e$ a matriz de elasticidade calculadas com as propriedades do elemento. E pode ser transformada em (3.20).

$$\mathbf{K}^e = \int_{\Omega^e} (\nabla_S [\mathbf{N}_1^e \mathbf{N}_2^e \dots \mathbf{N}_8^e])^T \mathbf{D} (\nabla_S [\mathbf{N}_1^e \mathbf{N}_2^e \dots \mathbf{N}_8^e])\, d\Omega^e \tag{3.20}$$

Definindo,

$$\mathbf{B}^e = \nabla_S[\mathbf{N}_1^e \mathbf{N}_2^e \dots \mathbf{N}_8^e] \tag{3.21}$$

A matriz de rigidez do elemento pode ser escrita como:

$$\mathbf{K}^e = \int_{\Omega^e} (\mathbf{B}^e)^T \mathbf{D}^e \mathbf{B}^e \, d\Omega^e \tag{3.22}$$

A montagem da matriz de rigidez pode ser feita calculando a matriz do elemento (3.22) e acumulando esses valores em suas respectivas posições globais. Abaixo, é apresentado o algoritmo para montagem da matriz de rigidez global.

Algoritmo 1: MontagemMatrizRigidez

```
início
    K^h ← 0
    para elemento e ∈ τ^h faça
        Calcule a matriz de rigidez do elemento K^e
        para Grau de liberdade i em e faça
            para Grau de liberdade j em e faça
                Acumule em K^h o valor K^e[i, j] na posição global
                correspondente
            fim
        fim
    fim
fim
```

Algo que ainda precisa ser feito é mostrar como o cálculo da matriz $\mathbf{K}^e$ é realizado. O primeiro passo é modificar a integral para um domínio onde ela pode ser mais facilmente calculada, para isso, cada domínio Ω^e será associado a um elemento padrão $\Omega^\xi = [-1, 1] \times [-1, 1]$ que é um elemento padrão em coordenadas ξ e η. A Figura 3.4 mostra a associação entre os dois elementos.

No elemento padrão Ω^ξ as funções de forma bilineares associadas aos nós são facilmente definidas como apresentado em (3.23).

$$\begin{aligned} \phi_1(\xi, \eta) &= \tfrac{1}{4}(1 - \xi)(1 - \eta) \\ \phi_2(\xi, \eta) &= \tfrac{1}{4}(1 + \xi)(1 - \eta) \\ \phi_3(\xi, \eta) &= \tfrac{1}{4}(1 + \xi)(1 + \eta) \\ \phi_4(\xi, \eta) &= \tfrac{1}{4}(1 - \xi)(1 + \eta) \end{aligned} \tag{3.23}$$

Um ponto de Ω^e é associado a um ponto em Ω^ξ através de uma associação

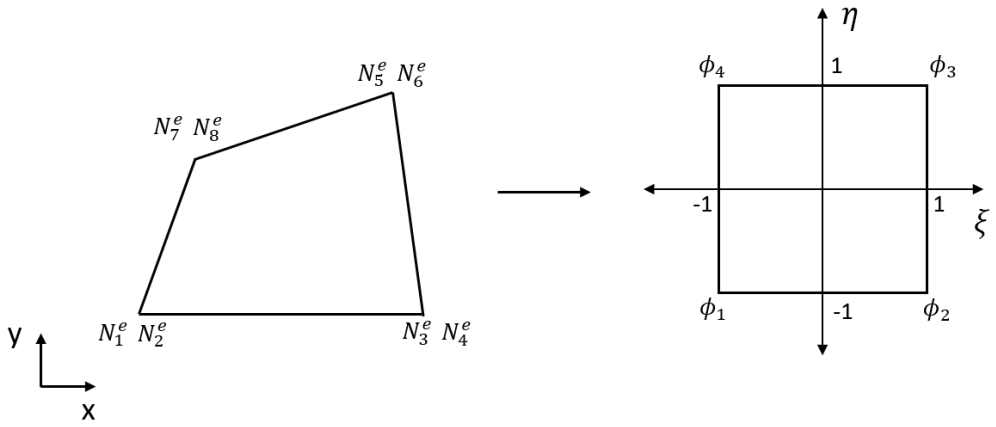


Figura 3.4: Bijeção entre elemento Ω^e e Ω^ξ

isoparamétrica de acordo com (3.24).

$$\begin{aligned} x(\xi,\eta) &= \textstyle\sum_{A=1}^{4} \phi_A(\xi,\eta) x_A^e \\ y(\xi,\eta) &= \textstyle\sum_{A=1}^{4} \phi_A(\xi,\eta) y_A^e \end{aligned} \tag{3.24}$$

Onde x_A^e e y_A^e para $A = 1, 2, 3, 4$ são as coordenadas dos vértices do elemento.

Com isso, as funções ϕ em (3.23) estão definidas implicitamente em função de x e y. As funções de base $\mathbf{N}_i^h$ assumem exatamente os valores dessas funções em um dado elemento. Portanto, pode-se reescrever a matriz $\mathbf{B}^e$ da seguinte forma:

$$\mathbf{B}^e = \nabla_S \begin{bmatrix} \phi_1 & 0 & \phi_2 & 0 & \phi_3 & 0 & \phi_4 & 0 \\ 0 & \phi_1 & 0 & \phi_2 & 0 & \phi_3 & 0 & \phi_4 \end{bmatrix} \tag{3.25}$$

Assim é possível realizar uma substituição de variáveis na integral apresentada em (3.20).

$$\begin{aligned} \mathbf{K}^e &= \int_{\Omega^e} (\mathbf{B}^e)^T \mathbf{D}^e \mathbf{B}^e \, d\Omega^e \\ &= \int_{-1}^{1} \int_{-1}^{1} (\mathbf{B}^e)^T \mathbf{D}^e \mathbf{B}^e |\mathbf{J}| d\xi d\eta \end{aligned} \tag{3.26}$$

Onde $\mathbf{J}$ representa o jacobiano

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \tag{3.27}$$

A integral no domínio $[-1, 1] \times [-1, 1]$ pode ser calculada através da Quadratura de Gauss que pode ser encontrada em [5]. O método consiste em trocar a integral por um somatório ponderado por pesos w_i em alguns pontos determinados da função, chamados de pontos de integração. A quantidade de pontos de integração determina

quão boa é a aproximação para o caso. Com n_p pontos de integração é possível integrar exatamente um polinômio de tamanho $2n_p - 1$.

Com isso a integral pode ser aproximada pelo somatório (3.28).

$$\sum_{i=1}^{n_p} (\nabla_S [\mathbf{N}_1^e \mathbf{N}_2^e \dots \mathbf{N}_8^e])^T \mathbf{D} (\nabla_S [\mathbf{N}_1^e \mathbf{N}_2^e \dots \mathbf{N}_8^e]) |\mathbf{J}||_{x=p_i} w_i \tag{3.28}$$

O cálculo do jacobiano em um determinado ponto pode ser calculado derivando (3.24)

$$\begin{array}{ll} \frac{\partial x}{\partial \xi} = \sum_{A=1}^{4} \frac{\partial \phi_A}{\partial \xi} x_A^e & \frac{\partial x}{\partial \eta} = \sum_{A=1}^{4} \frac{\partial \phi_A}{\partial \eta} x_A^e \\ \frac{\partial y}{\partial \xi} = \sum_{A=1}^{4} \frac{\partial \phi_A}{\partial \xi} y_A^e & \frac{\partial y}{\partial \eta} = \sum_{A=1}^{4} \frac{\partial \phi_A}{\partial \eta} y_A^e \end{array} \tag{3.29}$$

Além disso, para o cálculo de $\mathbf{B}^e$ é necessário calcular as derivadas de ϕ_A em relação as coordenadas x e y. Isso implica na inversão da matriz do Jacobiano, conforme mostrado abaixo.

$$\mathbf{J} \begin{bmatrix} \frac{\partial \phi^1}{\partial x} & \frac{\partial \phi^2}{\partial x} & \frac{\partial \phi^3}{\partial x} & \frac{\partial \phi^4}{\partial x} \\ \frac{\partial \phi^1}{\partial y} & \frac{\partial \phi^2}{\partial y} & \frac{\partial \phi^3}{\partial y} & \frac{\partial \phi^4}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial \phi^1}{\partial \xi} & \frac{\partial \phi^2}{\partial \xi} & \frac{\partial \phi^3}{\partial \xi} & \frac{\partial \phi^4}{\partial \xi} \\ \frac{\partial \phi^1}{\partial \eta} & \frac{\partial \phi^2}{\partial \eta} & \frac{\partial \phi^3}{\partial \eta} & \frac{\partial \phi^4}{\partial \eta} \end{bmatrix} \rightarrow \tag{3.30}$$

$$\begin{bmatrix} \frac{\partial \phi^1}{\partial x} & \frac{\partial \phi^2}{\partial x} & \frac{\partial \phi^3}{\partial x} & \frac{\partial \phi^4}{\partial x} \\ \frac{\partial \phi^1}{\partial y} & \frac{\partial \phi^2}{\partial y} & \frac{\partial \phi^3}{\partial y} & \frac{\partial \phi^4}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \frac{\partial \phi^1}{\partial \xi} & \frac{\partial \phi^2}{\partial \xi} & \frac{\partial \phi^3}{\partial \xi} & \frac{\partial \phi^4}{\partial \xi} \\ \frac{\partial \phi^1}{\partial \eta} & \frac{\partial \phi^2}{\partial \eta} & \frac{\partial \phi^3}{\partial \eta} & \frac{\partial \phi^4}{\partial \eta} \end{bmatrix} \tag{3.31}$$

Com (3.31) é possível calcular as derivadas das funções de forma em função das coordenadas x e y. Feito isso, todos os termos da matriz $\mathbf{B}^e$ podem ser obtidos finalizando assim o cálculo da matriz de rigidez do elemento.

3.1.4 Cálculo da Carga por Diferença de pressão

A Equação (3.8), mostra cada valor do vetor de carga $\mathbf{f}$. Três termos aparecem nesse caso: o primeiro relativo às condições de contorno de Neumann, o segundo relativo à pressão de poros e o terceiro relativo às condições de contorno de Dirichlet. Nesse trabalho, o segundo termo será o de maior importância, pois serão calculados os deslocamentos no reservatório e rochas adjacentes em função de uma diferença de pressão causada pela produção do petróleo e/ou injeção de água. Devido a essas duas ações, a pressão de poros do reservatório irá variar gerando deformações e variação nas tensões da rocha.

Assim, o cálculo do lado direito será realizado através do somatório da contribuição de cada elemento de forma análoga ao cálculo da matriz de rigidez. Abaixo, no cálculo da contribuição do elemento, será considerado que o campo de pressões será constante por elemento. Essa consideração foi realizada, pois esses valores serão recebidos de simuladores de fluxo que, geralmente, utilizam o método dos volumes

finitos e, portanto, calculam as pressões nos elementos e não nos nós.

$$\mathbf{f^e} = \int_{\Omega^e} \begin{bmatrix} (\nabla_S \mathbf{N}_1^e)^T \mathbf{m} \alpha^e P_p^e \\ (\nabla_S \mathbf{N}_2^e)^T \mathbf{m} \alpha^e P_p^e \\ (\nabla_S \mathbf{N}_3^e)^T \mathbf{m} \alpha^e P_p^e \\ (\nabla_S \mathbf{N}_4^e)^T \mathbf{m} \alpha^e P_p^e \\ (\nabla_S \mathbf{N}_5^e)^T \mathbf{m} \alpha^e P_p^e \\ (\nabla_S \mathbf{N}_6^e)^T \mathbf{m} \alpha^e P_p^e \\ (\nabla_S \mathbf{N}_7^e)^T \mathbf{m} \alpha^e P_p^e \\ (\nabla_S \mathbf{N}_8^e)^T \mathbf{m} \alpha^e P_p^e \end{bmatrix} d\Omega^e, \tag{3.32}$$

onde α^e e P_p^e são os valores do coeficiente de Biot e de pressão do elemento e em questão. Assim, pode-se continuar o calculo de $\mathbf{f^e}$.

$$\mathbf{f^e} = \int_{\Omega^e} \begin{bmatrix} (\nabla_S \mathbf{N}_1^e)^T \\ (\nabla_S \mathbf{N}_2^e)^T \\ (\nabla_S \mathbf{N}_3^e)^T \\ (\nabla_S \mathbf{N}_4^e)^T \\ (\nabla_S \mathbf{N}_5^e)^T \\ (\nabla_S \mathbf{N}_6^e)^T \\ (\nabla_S \mathbf{N}_7^e)^T \\ (\nabla_S \mathbf{N}_8^e)^T \end{bmatrix} \mathbf{m} \alpha^e P_p^e \, d\Omega^e = \int_{\Omega^e} (\nabla_S \begin{bmatrix} \mathbf{N}_1^e & \mathbf{N}_2^e & ... & \mathbf{N}_8^e \end{bmatrix})^T \mathbf{m} \, d\Omega^e \alpha^e P_p^e \tag{3.33}$$

O primeiro termo da equação pode ser substituída por (3.25).

$$\mathbf{f^e} = \int_{\Omega^e} (\nabla_S \begin{bmatrix} \phi_1 & 0 & \phi_2 & 0 & \phi_3 & 0 & \phi_4 & 0 \\ 0 & \phi_1 & 0 & \phi_2 & 0 & \phi_3 & 0 & \phi_4 \end{bmatrix})^T \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \alpha^e P_p^e \, d\Omega^e \tag{3.34}$$

$$\mathbf{f^e} = \int_{\Omega^e} \begin{bmatrix} \frac{\partial \phi_1}{\partial x} & 0 & \frac{\partial \phi_1}{\partial y} \\ 0 & \frac{\partial \phi_1}{\partial y} & \frac{\partial \phi_1}{\partial x} \\ \frac{\partial \phi_2}{\partial x} & 0 & \frac{\partial \phi_2}{\partial y} \\ 0 & \frac{\partial \phi_2}{\partial y} & \frac{\partial \phi_2}{\partial x} \\ \vdots & \vdots & \vdots \\ \frac{\partial \phi_8}{\partial x} & 0 & \frac{\partial \phi_8}{\partial y} \\ 0 & \frac{\partial \phi_8}{\partial y} & \frac{\partial \phi_8}{\partial x} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} d\Omega^e \alpha^e P_p^e = \int_{\Omega^e} \begin{bmatrix} \frac{\partial \phi_1}{\partial x} \\ \frac{\partial \phi_1}{\partial y} \\ \frac{\partial \phi_2}{\partial x} \\ \frac{\partial \phi_2}{\partial y} \\ \vdots \\ \frac{\partial \phi_8}{\partial x} \\ \frac{\partial \phi_8}{\partial y} \end{bmatrix} d\Omega^e \alpha^e P_p^e \tag{3.35}$$

A fórmula com a carga final utilizada é apresentada (3.35). Procedimento similar ao feito para a matriz de rigidez tem que ser feito para o cálculo dos valores: Mudança de variáveis para elemento padrão Ω^ξ, substituição da integral por um somatório

através da Quadratura de Gauss e cálculo da derivada das funções de forma no sistema de coordenadas x, y.

3.1.5 Condições de Contorno

As condições de contorno do problema já foram incorporadas no problema através do lado direito do sistema em (3.8), porém existem alternativas que, dependendo do caso, podem ser melhores para incorporar as condições de contorno ao problema. São apresentados aqui outras duas maneiras. As duas consistem de, ao invés de remover os graus de liberdade referentes as condições de Dirichlet, adicioná-los na matriz de rigidez e forçar junto do lado direito que tenham o valor imposto pela condição de contorno. Com isso, a matriz de rigidez terá sempre o mesmo tamanho igual a $2 \times n_{\text{nós}}$. A (3.36) apresenta como a condição de contorno pode ser adicionada a matriz, onde o valor B representa um valor muito maior que os outros valores da linha.

$$\begin{array}{c} \\ \bar{d}_1^{\,h} \to \\ \vdots \\ d_i^h \to \end{array} \begin{array}{c} \begin{array}{ccccc} \bar{d}_1^h & & & d_i^h & \\ \downarrow & & & \downarrow & \end{array} \\ \begin{pmatrix} B & \dots & (\nabla_S \mathbf{N}^h_{1+n_u^h})^T \mathbf{D} \nabla_S \mathbf{N}^h_i & \dots \\ \vdots & & \vdots & \\ (\nabla_S \mathbf{N}^h_i)^T \mathbf{D} \nabla_S \mathbf{N}^h_{n_u^h+1} & & (\nabla_S \mathbf{N}^h_i)^T \mathbf{D} \nabla_S \mathbf{N}^h_i & \dots \\ \vdots & & \vdots & \end{pmatrix} \end{array} \begin{pmatrix} \bar{d}_1^{\,h} \\ \vdots \\ d_i^h \\ \vdots \end{pmatrix} = \begin{pmatrix} B\bar{u}_1 \\ \vdots \\ f_i' \\ \vdots \end{pmatrix} \tag{3.36}$$

onde $\bar{u}_1$ representa o valor imposto pela condição de Dirichlet e f_i' representa o termo de (3.8) sem o termo da condição de Dirichlet presente.

Dessa forma, a primeira equação quando resolvida mostra $B\bar{d}_1^h = B\bar{u}_1 \to \bar{d}_1^h = \bar{u}_1$, já a equação relativa ao grau de liberdade d_i^h terá o lado direito f_i' complementada com o valor do lado esquerdo $\bar{d}_1^h(\nabla_S \mathbf{N}_i^h)^T \mathbf{D} \nabla_S \mathbf{N}^h_{1+n_u^h}$ (marcado em vermelho) ficando equivalente a (3.8) novamente.

Uma variante dessa versão é zerar toda a linha relativa à $\bar{d}_1^h$ com exceção da diagonal principal que terá valor um, conforme mostrado em (3.37). Esse caso tem a desvantagem de remover a simetria da matriz. Os dois métodos foram implementados nessa dissertação.

$$
\begin{array}{c}
\\
\bar{d_1}^h \to \\
\vdots \\
d_i^h \to
\end{array}
\begin{array}{c}
\begin{array}{c} d_1^h \\ \downarrow \end{array} \\
\begin{pmatrix}
1 & \dots \\
\vdots & \\
(\nabla_S \mathbf{N}_i^h)^T \mathbf{D} \nabla_S \mathbf{N}^h_{n_u^h+1} & \\
\vdots &
\end{pmatrix}
\end{array}
\begin{array}{c}
\begin{array}{c} d_i^h \\ \downarrow \end{array} \\
\begin{pmatrix}
0 & \dots \\
\vdots & \\
(\nabla_S \mathbf{N}_i^h)^T \mathbf{D} \nabla_S \mathbf{N}_i^h & \dots \\
\vdots &
\end{pmatrix}
\end{array}
\begin{pmatrix}
\bar{d_1}^h \\ \vdots \\ d_i^h \\ \vdots
\end{pmatrix}
=
\begin{pmatrix}
\bar{u}_1 \\ \vdots \\ f_i' \\ \vdots
\end{pmatrix}
\tag{3.37}
$$

Capítulo 4

Solução de Sistemas Lineares

Nesse capítulo, serão mostrados as técnicas utilizadas para a solução dos sistemas lineares gerados a partir da discretização apresentada no Capítulo 3. Nas simulações reais de geomecânica, a quantidade de células pode chegar a centenas de milhões de elementos de forma que métodos efetivos de estruturas de dados e algoritmos são necessários para que a resolução seja possível. A Seção 4.1 apresenta uma breve discussão sobre as estruturas de dados para armazenamento das matrizes, a Seção 4.2 apresenta alguns métodos de solução de sistemas esparsos, enquanto que a Seção 4.3 apresenta a fatoração LU.

4.1 Estruturas de Dados para Matrizes Esparsas

Conforme mostrado no Capítulo 3, a quantidade de não zeros da matriz (nnz) é $O(n_{\text{nós}})$, enquanto a quantidade total de entradas da matriz é da ordem de $O(n_{\text{nós}}^2)$. Dessa forma, é necessário utilizar uma estrutura de dados que seja capaz de armazenar apenas os não zeros para tornar a implementação mais eficiente. Uma ideia simples para guardar esse tipo de matriz consiste em guardar para cada elemento não zero seu valor, sua linha e sua coluna. Esse formato é chamado de *Coordinate Format* (COO) mostrado em SAAD [10]. Três vetores são necessários para guardar os valores:

- JR: vetor que guarda a linha de cada entrada (Tamanho nnz)
- JC: vetor que guarda a coluna de cada entrada (Tamanho nnz)
- AA: vetor que guarda os valores das entrada (Tamanho nnz)

Por exemplo a matriz,

$$\begin{bmatrix} 1 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 \\ 0 & 6 & 7 & 0 \\ 0 & 0 & 8 & 9 \end{bmatrix} \tag{4.1}$$

Tem como vetores associados na sua representação COO os mostrados abaixo:

- JR = 1 1 2 2 2 3 3 4 4
- JC = 1 3 1 2 4 2 3 3 4
- AA = 1 2 3 4 5 6 7 8 9

É importante perceber que nesse formato, as entradas da matriz podem ser escritas em qualquer ordem. Porém, no exemplo acima, elas estão ordenadas por linha e pode-se notar que o vetor JR apresenta uma série de valores repetidos. Para tirar proveito dessa repetição, existe outro formato chamado *Compressed Sparse Row Format* (CSR) onde o vetor associado as linhas JR é substituído por um vetor IA que é um ponteiro para o vetor de colunas com o início de cada uma das linhas.

Para a matriz acima, tem-se:

- A linha 1 é a primeira e, portanto, IA(1) = 1
- A linha 2 começa a partir do elemento 3 do vetor AA e, portanto, IA(2) = 3
- A linha 3 começa a partir do elemento 5 do vetor AA e, portanto, IA(3) = 6
- A linha 4 começa a partir do elemento 7 do vetor AA e, portanto, IA(4) = 8

É adicionado ainda um valor a mais ao vetor IA que guarda a quantidade de nnz+1. No exemplo acima, esse valor é IA(5) = 10. Com isso é possível saber as entradas de determinada linha i olhando para as posições IA(i) a IA(i+1)-1 do vetor AA e as respectivas colunas em JC. Os valores dos vetores CSR da matriz então ficam:

- IA = 1 3 6 8 10
- JC = 1 3 1 2 4 2 3 3 4
- AA = 1 2 3 4 5 6 7 8 9

De acordo com SAAD [10], o formato CSR é provavelmente o mais comum para guardar matrizes esparsas e costuma ser a porta de entrada de estrutura de dados para matrizes esparsas. Nesse trabalho, a implementação é feita utilizando esse tipo de estrutura de dados conforme será mostrado no Capítulo 6.

Uma operação particularmente importante com matrizes é a multiplicação matriz vetor $\mathbf{y} = \mathbf{Ax}$. No contexto dessa dissertação, ela é importante para aplicar o operadores de prolongamento e restrição apresentados no Capítulo 5 e também nos métodos iterativos de solução de sistemas lineares: Gradiente Conjugado (CG) e Bicgstab. O Algoritmo 2 apresenta a multiplicação matriz vetor utilizando uma matriz CSR representada pelos vetores IA, JC, AA pelo vetor x.

Algoritmo 2: y = MultMatrizVetor(IA(n+1), JC(nnz), AA(nnz), x(n))

```
início
    y ← 0
    para row ∈ {1, 2, 3, ..., n} faça
        para j ∈ {IA(row), ..., IA(row+1)-1} faça
            col ← JC(j)
            y(row) ← y(row) + AA(j) × x(col)
        fim
    fim
fim
```

Nesse algoritmo, é importante perceber que a variável j irá assumir valores de 1 a IA(n+1)-1 e , como IA(n+1)=nnz+1, a multiplicação tem complexidade $O(\text{nnz})$. O CSR então, além de necessitar de menos memória para armazenar a matriz, também tem a vantagem de fazer operações de multiplicação mais eficientemente visto que a complexidade seria $O(n^2)$ caso a matriz fosse armazenada densa.

4.2 Solução de Sistemas Esparsos

Nessa seção serão apresentados alguns métodos para solução de sistemas lineares, em especial o Gradiente Conjugado que será utilizado para as comparações entre o método multiescala e multigrid apresentados no Capítulo 7. Será considerado o sistema linear apresentado em (4.2).

$$\mathbf{Ax} = \mathbf{b} \tag{4.2}$$

4.2.1 Precondicionadores

Define-se como número de condicionamento da matriz a expressão apresentada em (4.3), quanto maior esse valor mais difícil a convergência do sistema com métodos de Krylov. As matrizes advindas de sistemas gerados por (2.20) costumam ter números

de condicionamentos elevados fazendo com que os métodos de solução tenham dificuldade de convergir.

$$\text{cond}(\mathbf{A}) = ||\mathbf{A}|| \cdot ||\mathbf{A}^{-1}|| \tag{4.3}$$

Uma estratégia importante para a solução desses sistemas lineares é a utilização de precondicionadores que consistem em tentar resolver um sistema equivalente a (4.2) mas com um número de condicionamento menor. Há dois principais métodos de precondicionamento, pela esquerda ou pela direita. Chamando de $\mathbf{M}$ a matriz de precondicionamento, (4.4) e (4.5) apresentam o precondicionamento pela esquerda e pela direita respectivamente. Também é possível fazer o precondicionamento utilizando ambos os lados. A matriz de precondicionamento geralmente aproxima a matriz original $\mathbf{A}$, de forma que $\mathbf{M}^{-1}\mathbf{A}$ e $\mathbf{A}\mathbf{M}^{-1}$ tenham número de condicionamento ordens de grandeza menores que o da matriz original. Um fato importante é que, apesar de aparecerem os produtos $\mathbf{M}^{-1}\mathbf{A}$ e $\mathbf{A}\mathbf{M}^{-1}$, eles não são efetivamente realizados, pois, muitas vezes se é conhecido apenas a matriz $\mathbf{M}$ e, mesmo que $\mathbf{M}^{-1}$ fosse conhecido, esses produtos podem ser densos.

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{4.4}$$

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b} \tag{4.5}$$

$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{y} \tag{4.6}$$

Precondicionadores populares são os baseados em fatorações incompletas (ILU) apresentados em MEIJERINK e VAN DER VORST [11]. Esses precondicionadores calculam aproximações para a fatoração LU de uma matriz e são apresentadas na seção 4.3.

4.2.2 Iteração de Richardson

Um método simples de solução de sistemas lineares é a Iteração de Richardson, essa solução é utilizada em solvers multiescala apresentados em MANEA *et al.* [12]. No caso, dada a matriz uma matriz $\mathbf{A}$ e um precondicionador $\mathbf{M}$ uma iteração do método é definida conforme apresentado em (4.7).

$$\mathbf{I}_m = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}) \tag{4.7}$$

$$\mathbf{x}_{i+1} = \mathbf{I}_m\mathbf{x}_i + \mathbf{M}^{-1}\mathbf{b} \tag{4.8}$$

Onde, $\mathbf{I}$ representa a matriz identidade. A partir de (4.7) é possível encontrar a equação do erro ao longo das iterações $\mathbf{e}_{i+1} = \mathbf{I}_m \mathbf{e}_i$, isso faz com que a convergência seja garantida apenas se $\rho(\mathbf{I}_m) < 1$, onde ρ representa o raio espectral da matriz. Essa condição é necessária para que cada componente do erro relativa a cada um dos autovetores reduza a cada iteração.

4.2.3 Método do Gradiente Conjugado

O método do Gradiente Conjugado pode ser utilizado na solução de sistemas lineares da forma (4.2) onde $\mathbf{A}$ é uma matriz simétrica positiva definida (SPD) conforme mostrado em SHEWCHUK [13]. Nesse caso, pode-se encontrar um problema de minimização equivalente através da definição da forma quadrática apresentada em (4.9).

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c, \quad f : \mathbb{R}^n \to \mathbb{R} \tag{4.9}$$

Para encontrar os pontos extremos da função (4.9) é necessário calcular os pontos em que o gradiente de f é o vetor nulo. Portanto, é necessário calcular a derivada parcial de f para cada uma das coordenadas. Para facilitar esse cálculo a equação (4.9) é apresentada em notação indicial em (4.10).

$$f(\mathbf{x}) = \frac{1}{2}x_i A_{ij} x_j - b_i x_i + c \tag{4.10}$$

Aplicando a derivada parcial com relação a uma coordenada x_k.

$$\begin{aligned}
\frac{\partial f(\mathbf{x})}{\partial x_k} &= \frac{1}{2}\frac{\partial x_i A_{ij} x_j}{\partial x_k} - \frac{\partial}{\partial x_k} b_i x_i + \frac{\partial}{\partial x_k} c && (4.11)\\
&= \frac{1}{2}\frac{\partial x_i A_{ij} x_j}{\partial x_k} - b_k && (4.12)\\
&= \frac{1}{2}\left(\frac{\partial x_i}{\partial x_k} A_{ij} x_j + x_i A_{ij}\frac{\partial x_j}{\partial x_k}\right) - b_k && (4.13)\\
&= \frac{1}{2}(A_{kj} x_j + x_i A_{ik}) - b_k && (4.14)
\end{aligned}$$

Retornando para a notação matricial

$$\nabla f = \frac{1}{2}(\mathbf{A}\mathbf{x} + \mathbf{A}^T\mathbf{x}) - \mathbf{b} \tag{4.15}$$

$$\nabla f = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T)\mathbf{x} - \mathbf{b} \tag{4.16}$$

Como $\mathbf{A}$ é simétrica ($\mathbf{A}^T = \mathbf{A}$)

$$\nabla f = \mathbf{A}\mathbf{x} - \mathbf{b} \tag{4.17}$$

Em um ponto extremo $\mathbf{x}_e$ de $f(\mathbf{x})$ tem-se $(\nabla f)|_{\mathbf{x}=\mathbf{x}_e} = 0$

$$\mathbf{A}\mathbf{x}_e - \mathbf{b} = 0 \tag{4.18}$$

$$\mathbf{A}\mathbf{x}_e = \mathbf{b} \tag{4.19}$$

Portanto, $f(\mathbf{x})$ possui um único ponto extremo $\mathbf{x}_e = \mathbf{A}^{-1}\mathbf{b}$ que é justamente a solução do sistema linear (4.2). Resta mostrar que $\mathbf{x}_e$ é um ponto de mínimo. A demonstração a seguir pode ser encontrada em SHEWCHUK [13]. Considerando $\mathbf{x} = \mathbf{x}_e + \mathbf{e}$ com $\mathbf{e} \neq 0$

$$\begin{aligned}
f(\mathbf{x}_e + \mathbf{e}) &= \frac{1}{2}(\mathbf{x}_e + \mathbf{e})^T\mathbf{A}(\mathbf{x}_e + \mathbf{e}) - \mathbf{b}^T(\mathbf{x}_e + \mathbf{e}) + c \\
&= \frac{1}{2}(\mathbf{x}_e^T\mathbf{A}\mathbf{x}_e + \mathbf{x}_e^T\mathbf{A}\mathbf{e} + \mathbf{e}^T\mathbf{A}\mathbf{x}_e + \mathbf{e}^T\mathbf{A}\mathbf{e}) - \mathbf{b}^T(\mathbf{x}_e + \mathbf{e}) + c \\
&= \frac{1}{2}\mathbf{x}_e^T\mathbf{A}\mathbf{x}_e - \mathbf{b}^T\mathbf{x}_e + c + \frac{1}{2}(\mathbf{x}_e^T\mathbf{A}\mathbf{e} + \mathbf{e}^T\mathbf{A}\mathbf{x}_e + \mathbf{e}^T\mathbf{A}\mathbf{e}) - \mathbf{b}^T\mathbf{e} \\
&= f(\mathbf{x}_e) + \frac{1}{2}(\mathbf{x}_e^T\mathbf{A}\mathbf{e} + \mathbf{e}^T\mathbf{A}\mathbf{x}_e + \mathbf{e}^T\mathbf{A}\mathbf{e}) - \mathbf{b}^T\mathbf{e} \\
&= f(\mathbf{x}_e) + \frac{1}{2}((\mathbf{A}^{-1}\mathbf{b})^T\mathbf{A}\mathbf{e} + \mathbf{e}^T\mathbf{A}\mathbf{A}^{-1}\mathbf{b} + \mathbf{e}^T\mathbf{A}\mathbf{e}) - \mathbf{b}^T\mathbf{e} \\
&= f(\mathbf{x}_e) + \frac{1}{2}(\mathbf{b}^T\mathbf{A}^{-1}\mathbf{A}\mathbf{e} + \mathbf{e}^T\mathbf{b} + \mathbf{e}^T\mathbf{A}\mathbf{e}) - \mathbf{b}^T\mathbf{e} \\
&= f(\mathbf{x}_e) + \frac{1}{2}(\mathbf{b}^T\mathbf{e} + \mathbf{e}^T\mathbf{b} + \mathbf{e}^T\mathbf{A}\mathbf{e}) - \mathbf{b}^T\mathbf{e} \\
&= f(\mathbf{x}_e) + \frac{1}{2}\mathbf{e}^T\mathbf{A}\mathbf{e} + \mathbf{b}^T\mathbf{e} - \mathbf{b}^T\mathbf{e} \\
&= f(\mathbf{x}_e) + \frac{1}{2}\mathbf{e}^T\mathbf{A}\mathbf{e}
\end{aligned}$$

Como $\mathbf{A}$ é positiva definida, por definição $\mathbf{e}^T\mathbf{A}\mathbf{e} > 0$, quando $\mathbf{e} \neq \mathbf{0}$, logo

$$f(\mathbf{x}_e + \mathbf{e}) = f(\mathbf{x}_e) + \frac{1}{2}\mathbf{e}^T\mathbf{A}\mathbf{e} > f(\mathbf{x}_e) \rightarrow f(\mathbf{x}_e + \mathbf{e}) > f(\mathbf{x}_e),\ \forall \mathbf{e} \neq \mathbf{0} \tag{4.20}$$

Assim, $\mathbf{x}_e$ é ponto de mínimo global de $f(\mathbf{x})$ e o problema de encontrar $\mathbf{x}$ tal que $\mathbf{A}\mathbf{x} = \mathbf{b}$ é equivalente a *minimizar* $f(\mathbf{x})$. Para encontrar o mínimo de f , é possível utilizar o método de otimização do Gradiente Conjugado. O método consiste em partir de um ponto $\mathbf{x}_0$ e caminhar em direções conjugadas até que o mínimo local seja encontrado. Entende-se por direções conjugadas $\mathbf{d}_i$ e $\mathbf{d}_j$ tais que $\mathbf{d}_i^T\mathbf{A}\mathbf{d}_j = 0$.

Em matemática exata, o método do Gradiente Conjugado encontra a solução

exata do sistema linear em no máximo n iterações, onde n é a ordem da matriz, porém, este é utilizado como solver iterativo aproximado, onde outro critério de parada é utilizado, como por exemplo, quando o resíduo da solução se torna menor que um determinado valor. O Algoritmo 3 apresenta o método do Gradiente Conjugado.

Algoritmo 3: GradienteConjugado($\mathbf{A}$, $\mathbf{x}$, $\mathbf{b}$, i_{max}, ϵ)

início
- $i \leftarrow 0$
- $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{Ax}$
- $\mathbf{d} \leftarrow \mathbf{r}$
- $\delta_{new} \leftarrow \mathbf{r}^T\mathbf{r}$
- $\delta_0 \leftarrow \delta_{new}$
- **enquanto** $i < i_{max}$ *e* $\delta_{new} > \epsilon^2\delta_0$ **faça**
  - $\mathbf{q} \leftarrow \mathbf{Ad}$
  - $\alpha \leftarrow \delta_{new}/\mathbf{d}^T\mathbf{q}$
  - $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{d}$
  - $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{q}$
  - $\delta_{old} \leftarrow \delta_{new}$
  - $\delta_{new} \leftarrow \mathbf{r}^T\mathbf{r}$
  - $\beta \leftarrow \delta_{new}/\delta_{old}$
  - $\mathbf{d} \leftarrow \mathbf{r} + \beta\mathbf{d}$
  - $i \leftarrow i + 1$
- **fim**

fim

Gradiente Conjugado Precondicionado

Uma dificuldade em se utilizar um precondicionador com o Gradiente Conjugado é que a garantia de convergência tem como condição suficiente que a matriz seja simétrica positiva definida (SPD). Porém, se aplicarmos um precondicionador $\mathbf{M}^{-1}$ pela esquerda ou pela direita perde-se a simetria da matriz. Para solucionar essa questão, o precondicionador é dividido no produto $\mathbf{EE}^T = \mathbf{M}$ e é aplicado simultaneamente pela esquerda e pela direita conforme mostrado em (4.21), que também preserva a simetria da matriz.

$$(\mathbf{E}^{-1})\mathbf{A}(\mathbf{E}^{-T})\mathbf{y} = \mathbf{E}^{-1}\mathbf{b} \tag{4.21}$$

$$\mathbf{x} = \mathbf{E}^{-T}\mathbf{y} \tag{4.22}$$

É importante notar que para que tal decomposição exista é necessário que a matriz seja SPD, visto que, $\mathbf{M}^T = (\mathbf{E}\mathbf{E}^T)^T = (\mathbf{E}^T)^T\mathbf{E}^T = \mathbf{E}\mathbf{E}^T = \mathbf{M}$ e que $\mathbf{x}^T\mathbf{M}\mathbf{x} = \mathbf{x}^T\mathbf{E}\mathbf{E}^T\mathbf{x} = ||\mathbf{E}^T\mathbf{x}||^2$. Em SHEWCHUK [13] é mostrado que apesar dessa decomposição, o método não precisa calcular explicitamente a matriz $\mathbf{E}$ conforme apresentado no Algoritmo 4.

Algoritmo 4: GradienteConjugadoPrecon($\mathbf{A}$, $\mathbf{x}$, $\mathbf{b}$, $\mathbf{M}^{-1}$, i_{max}, ϵ)

início
- $i \leftarrow 0$
- $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$
- $\mathbf{d} \leftarrow \mathbf{M}^{-1}\mathbf{r}$
- $\delta_{new} \leftarrow \mathbf{r}^T\mathbf{r}$
- $\delta_0 \leftarrow \delta_{new}$
- **enquanto** $i < i_{max}$ *e* $\delta_{new} > \epsilon^2\delta_0$ **faça**
  - $\mathbf{q} \leftarrow \mathbf{A}\mathbf{d}$
  - $\alpha \leftarrow \delta_{new}/\mathbf{d}^T\mathbf{q}$
  - $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{d}$
  - $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{q}$
  - $s \leftarrow \mathbf{M}^{-1}\mathbf{r}$
  - $\delta_{old} \leftarrow \delta_{new}$
  - $\delta_{new} \leftarrow \mathbf{r}^T\mathbf{s}$
  - $\beta \leftarrow \delta_{new}/\delta_{old}$
  - $\mathbf{d} \leftarrow \mathbf{r} + \beta\mathbf{d}$
  - $i \leftarrow i + 1$
- **fim**

fim

4.2.4 Método Multigrid

O método multigrid tem como ideia principal a resolução do solver linear utilizando um conjunto de operadores que representam versões mais grosseiras do operador inicial, onde os níveis mais finos são responsáveis a reduzir os erros de alta frequência enquanto os níveis mais grosseiros reduzem as frequências baixas do erro, conforme

mostrado em BRIGGS *et al.* [14]. O nível mais grosso geralmente é resolvido com solver direto, pois a quantidade de variáveis é pequena.

Apesar de inicialmente ter sido concebido com ideias geométricas onde o operador é calculado com base no grid, poucas fontes mostram implementações dessa versão, e o método ficou mais reconhecido por sua versão algébrica onde os operadores são montados apenas a partir das entradas da matriz. Em especial, nesse trabalho será realizada a comparação com a biblioteca PyAMG (OLSON e SCHRODER [15]).

Entre os níveis, são necessários operadores que levam os vetores de uma escala mais fina para uma escala mais grossa e vice-versa. O operador que leva um vetor de um grid fino para o grid grosso é chamado de restrição, enquanto um que leva de um nível mais grosso para um nível mais fino é chamado de prolongamento.

Iniciando com um exemplo de solver multigrid apenas com dois níveis, chamemos de $\mathbf{A^1}$ como sendo o operador do nível fino e $\mathbf{A}^2$ o operador do grid grosso. Nesse caso, só existe um operador de prolongamento ($\mathbf{P}_2^1$) e um operador de restrição ($\mathbf{R}_1^2$). Uma iteração de um solver multigrid com dois níveis é apresentada no Algoritmo 5. A cada ciclo multigrid, υ_1 relaxações são aplicadas no grid fino antes de descer para o nível mais grosso e υ_2 são aplicadas após o retorno do grid grosso para o grid fino. As relaxações tem como intuito fazer reduzir os erros de alta frequência nos níveis mais finos e os de baixa frequência nos níveis mais grossos. Esse ciclo é semelhante ao apresentado para o método multiescala no Capítulo 5 com a diferença que apenas uma relaxação no grid fino é aplicada no método multiescala por iteração.

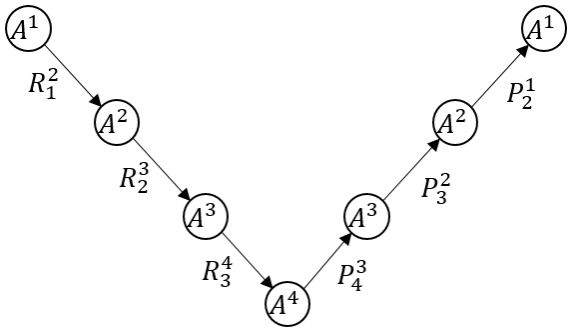
Quando mais que dois níveis são utilizados com o Multigrid, existem mais opções de ciclos possíveis. O ciclo mais simples é o ciclo V apresentado na Figura 4.1(a), onde se desce até o nível mais grosso e se retorna para o nível mais fino formando uma figura de V. Uma variação é o ciclo W apresentado na Figura 4.1(b). Esses dois ciclos podem ser descritos de acordo com o ciclo μ descrito no Algoritmo 6, onde para o ciclo V tem-se $\mu = 1$, enquanto para o ciclo W $\mu = 2$. É importante perceber que o ciclo W possui mais relaxações e mais aplicações de operadores de restrição e prologamento e, portanto, possui um maior custo computacional.

Algoritmo 5: Ciclo-V(A_1, A_2, x_0, b)

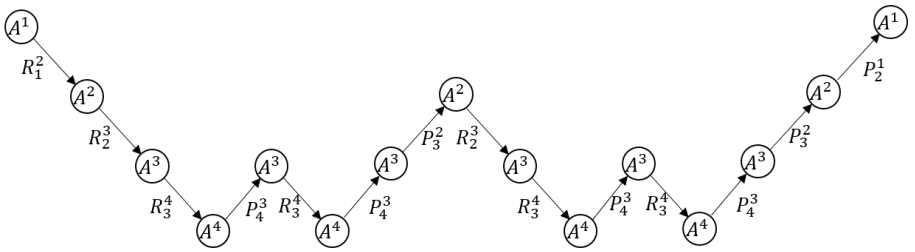
início

$x_0 \leftarrow$ relaxação(A_1, x_0, b), υ_1 vezes
$r \leftarrow b - A_1 x_0$
$r \leftarrow R_1^2 r$
$\delta_2 \leftarrow A_2^{-1} r$
$\delta_1 \leftarrow P_2^1 \delta_2$
$x_0 \leftarrow x_0 + \delta_1$
$x_0 \leftarrow$ relaxação(A_1, x_0, b), υ_2 vezes

fim



(a) Ciclo V



(b) Ciclo W

Figura 4.1: Aplicação de um ciclo V e W com quatro níveis. Os círculos A_1, A_2, A_3 representam relaxações com esses operadores, enquanto o círculo relacionado a A_4 representa a solução direta de um sistema linear.

Algoritmo 6: Ciclo-$\mu(i,\ \mu,\ x,\ b,\ A_1,\ A_2,\ ...)$ (Adaptado de [14])

início
  relaxação(A_i, x), υ_1 vezes
  $b_{i+1} \leftarrow R_i^{i+1}(b - A_i x)$
  se *$i + 1$ é o nível mais grosso* **então**
    $x_{i+1} \leftarrow A_{i+1}^{-1} b_{i+1}$
  fim
  senão
    $x_{i+1} \leftarrow \mathbf{0}$
    $x_{i+1} \leftarrow$ Ciclo-$\mu(i+1,\ \mu,\ x_i,\ b_i,\ A_1,\ A_2,\ ...)$, μ vezes
  fim
  $x \leftarrow x + P_{i+1}^i x_{i+1}$
  relaxação(A_i, x), υ_2 vezes
fim

Relaxações

Em relação às relaxações utilizadas pelo método multigrid, duas bastante utilizadas são a Jacobi e o Gauss-Seidel. Elas tem como intuito remover os erros de alta frequência em cada um dos níveis multigrid. Em particular, a relaxação de Gauss-

Seidel simétrica foi utilizada na comparação do Capítulo 7 e é mostrada no Algoritmo 7 que foi retirado do código do PyAMG. É importante notar que, nesse caso, a quantidade de operações feitas é proporcional a duas vezes o número de não zeros da matriz , pois, a relaxação é realizada começando de i=1 e depois voltando de i=n. Uma outra maneira de implementar o Gauss-Seidel é a apresentada em [10] onde as iterações são definidas pela recorrência: $x^{k+1/2} = x^k + (L+D)^{-1}r^k$ e $x^{k+1} = x^{k+1/2} + (D+U)^{-1}r^{k+1/2}$.

Algoritmo 7: Gauss-Seidel-Simétrico(A, x, b)

```
início
    n ← Tamanho de A
    para i ∈ 1, 2, 3, ··· , n faça
        x(i) ← b(i)
        para j ∈ 1, 2, 3, ··· , n faça
            se j ≠ i então
                x(i) ← x(i) − A(i, j) × x(j)
            fim
        fim
        x(i) ← x(i)/A(i, i)
    fim
    para i ∈ n, n − 1, n − 2, ··· , 1 faça
        x(i) ← b(i)
        para j ∈ 1, 2, 3, ··· , n faça
            se j ≠ i então
                x(i) ← x(i) − A(i, j) × x(j)
            fim
        fim
        x(i) ← x(i)/A(i, i)
    fim
fim
```

Complexidade do Grid

O PyAMG possui um método para os seus solvers para cálculo da Complexidade do Grid, esse valor tenta representar o quão mais custoso é a utilização de um determinado solver multigrid em relação ao operador original. A definição é mostrada em (4.23). Como pode-se ver, ele conta quantos não zeros o somatório de todos os níveis possui a mais que o nível fino. Assim, por exemplo, considerando $\upsilon_1 = 1$ e

$\upsilon_2 = 1$ cada nível irá realizar duas relaxações, com exceção do nível mais grosso, tornando o custo de um ciclo aproximadamente $2 \times$ Complexidade Grid.

$$\text{Complexidade Grid} = \frac{\sum_{i=1}^{n} \text{nnz}(A_i)}{\text{nnz}(A_1)} \tag{4.23}$$

Para saber o custo de um ciclo W se torna mais complicado devido a recursão associada a esse tipo de ciclo. No caso do ciclo apresentado na Figura 4.1(b), são feitas duas relaxações com A_1 e quatro com A_2 (o vértice A_2 do meio em que chega e sai uma seta acontecem duas relaxações) e oito relaxações com A_3. O PyAMG também tem um método que calcula a complexidade do ciclo através de recursão e foi utilizado para as comparações apresentadas no Capítulo 6.

4.3 Fatoração LU

Outra parte importante para esse trabalho é a solução direta de sistemas, em particular por conta dos sistemas que serão resolvidos nos espaços grossos e também para cálculo das funções de base que serão apresentados no Capítulo 5. Esses sistemas tem um número reduzido de variáveis em comparação com os sistemas associados com o grid fino e, portanto, pode-se pensar na utilização de solvers diretos.

A fatoração LU consiste em transformar a matriz A no produto de outras duas $\mathbf{A} = \mathbf{LU}$: $\mathbf{L}$ triangular inferior e $\mathbf{U}$ triangular superior. Assim, o sistema fica da forma $\mathbf{LUx} = \mathbf{b}$ e a solução é realizada através da solução de dois sistemas triangulares $\mathbf{Ly} = \mathbf{b}$ e $\mathbf{Ux} = \mathbf{y}$. O cálculo dos fatores LU pode ser feita utilizando uma eliminação gaussiana e pode ser encontrado em HEATH [16]. Uma vantagem desse método é que se for necessário resolver sistemas para diferentes lados direito $\mathbf{b}$ a fatoração pode ser reutilizada e não precisa ser calculada novamente. Para matrizes simétricas positivas definidas, existe a fatoração de Cholesky, similar a fatoração LU, onde $\mathbf{A} = \mathbf{C}^T\mathbf{C}$, com C triangular superior.

Precondicionador ILU

Os precondicionadores baseados em fatoração incompletas (ILU) apresentados em MEIJERINK e VAN DER VORST [11] calculam uma aproximação para a fatoração LU da matriz. A aproximação é realizada desconsiderando algumas entradas das matrizes $\mathbf{L}$ e $\mathbf{U}$. Um parâmetro do método é o nível do ILU que controla a quantidade de entradas que a fatoração incompleta possui; quando o nível é zero, a fatoração possui não zeros nas mesmas posições em que a matriz $\mathbf{A}$ possui entradas não-nulas. Há várias outras alternativas baseadas em aproximações da fatoração LU, ver [10].

Capítulo 5

Operador Multiescala

Nesse capítulo, serão apresentadas a construção do operador grosso Multiescala e suas aplicações no contexto das equações apresentadas no Capítulo 2. Daqui para frente, o grid original do problema será denominado grid fino e o operador associado a esse grid $\mathbf{K}^h$ será chamado de operador fino. A partir do grid fino será construído um grid grosso onde cada elemento é uma aglomeração de elementos do grid fino, o operador associado a esse grid será chamado de operador grosso $\mathbf{K}^H$.

O método *Multiscale Finite Element Method* (MSFEM) foi introduzido por THOMAS Y. HOU [17] e inicialmente foi aplicado para obter soluções aproximadas do grid fino através do grid grosso para casos de transferência de calor de materiais e meios porosos com propriedades aleatórias. A principal conclusão desse trabalho foi que o método MSFEM é capaz de reproduzir a solução do problema fino através da construção das funções de base com solução de problemas locais baseados no operador original que consegue reproduzir as heterogeneidades do mesmo. Mais tarde, o método MSFEM se mostrou problemático para conservação de massa em problemas de transporte e, por isso, foram criado os métodos *Mixed Multiscale Finite Element* (MMSFEM) em CHEN e HOU [18] e *Multiscale Finite Volume Method* (MSFV) em JENNY e TCHELEPI [19] que garantem a conservação.

Apesar do método MSFV garantir a conservação da massa, as soluções do grid grosso prolongadas para o grid fino não se mostravam adequadas para todas as situações, assim, foi apresentado o método multiescala iterativo em HAJIBEYGI *et al.* [20] que ao invés de realizar uma aproximação para a solução fina, utiliza a solução do sistema grosso juntamente com uma relaxação do operador fino para resolver o sistema na malha fina através de iterações de Richardson. A convergência da solução depende da quantidade de vezes que uma relaxação do grid fino é aplicada e varia de problema para problema.

Os estudos citados acima precisam do grid do problema para que seja possível aplicar o método multiescala o que é uma desvantagem quando é necessário utilizar o método como um solver "caixa-preta". Essa desvantagem é parcialmente

resolvida através do multiescala algébrico, apresentado em ZHOU e TCHELEPI [21], que constroem o operador grosso utilizando uma ordenação Wire-Basket que guarda a topologia da malha em questão. Métodos com vários níveis, semelhantes ao multigrid também já foram desenvolvidos como em MANEA e ALMANI [22].

Os trabalhos citados nos parágrafos anteriores são de aplicações do método multiescala para problemas de fluxo que, historicamente, na indústria do petróleo têm maior apelo por conta da previsão de produção dos campos. Porém, estudos relacionados ao método multiescala para geomecânica também podem ser encontrados em N. CASTELLETTO [3], SOKOLOVA K. [8] e CASTELLETTO *et al.* [23]. É importante salientar que [3] e [8] apresentam acoplamento entre a simulação de fluxo com a geomecânica. Além disso, visto que (2.16) é similar ao problema de elasticidade linear para sólidos, pode-se encontrar trabalhos úteis com aplicações do método de multiescala neste domínio, por exemplo MARCO BUCK [24].

Implementações em paralelo do método multiescala também podem ser encontradas, por exemplo em MANEA *et al.* [12] que faz comparação com o método multigrid mostrando que o multiescala é uma opção competitiva e tem escalabilidade similar ao multigrid.

5.1 Construção de Operador Grosso Multiescala

A ideia do método consiste na construção de um operador grosso semelhante ao realizado no multigrid de forma que a solução desse operador pode ser utilizada para ajudar na solução da escala original do problema ou ainda ser utilizada como uma aproximação para a mesma. Para isso, é necessário construir o próprio operador grosso e também operadores que façam a transferência de escala do grid fino para o grosso (restrição) e do grid grosso para o grid fino (prolongamento).

O primeiro passo para a construção do operador multiescala é gerar um novo grid com menos elementos que o grid original do problema (grid fino), mas que ainda represente o mesmo domínio Ω. Esse novo grid será chamado de grid grosso e as variáveis relacionadas com ele serão denotadas com o sobrescrito H. Dessa forma, o grid grosso possui um conjunto τ^H de elementos onde cada elemento será uma aglomeração de elementos do grid fino. Por exemplo, a Figura 5.1 apresenta um grid grosso 3×3 construído a partir de um grid fino 7×7. As linhas finas representam as divisões do grid fino e as grossas as divisões do grid grosso, já os quadrados azuis destacam os nós que pertencem ao grid grosso e ao grid fino simultaneamente. Apesar de serem apresentadas diferentes quantidade de elementos sendo aglomerados para formar o grid grosso, as implementações deste trabalho consideram uma quantidade fixa de elementos em cada direção. Define-se o fator de engrossamento de acordo com (5.1). Analogamente podem ser definidos os fatores C_{r_x} e C_{r_y} que

representam os engrossamento em cada direção, onde, $C_r = C_{r_x} \times C_{r_y}$.

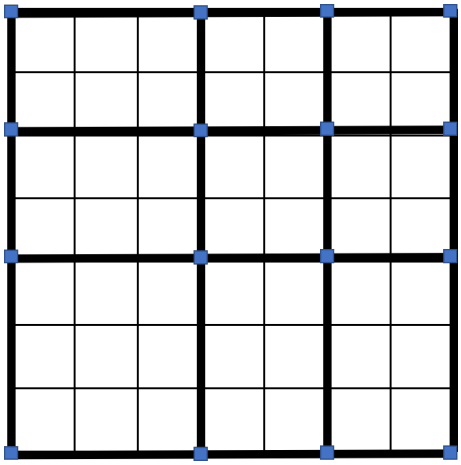


Figura 5.1: Exemplo de grid fino 7×7 e um grid grosso 3×3. O elemento inferior esquerdo é composto por 9 elementos do grid fino enquanto o elemento superior direito é composto com 4 elementos do grid fino.

$$C_r = \frac{n_{\mathrm{e}}^h}{n_{\mathrm{e}}^H} \tag{5.1}$$

Do mesmo modo que apresentado para o grid fino no Capítulo 3, cada grau de liberdade grosso d_i^H, para $i = 1, 2, ..., n_u^H$, será associada a uma função de base $\mathbf{N}_i^H$ e cada nó associado a uma condição de contorno de Dirichlet $\bar{d}_i^H$, para $i = 1, 2, ..., n_{\bar{u}}^H$, uma função de base $\mathbf{N}_{i+n_u^H}^H$. E então, é possível encontrar uma solução aproximada no grid fino $\mathbf{u}^h \approx \mathbf{u}_{ms}^h$ através de (5.2).

$$\mathbf{u}_{ms}^h = \mathbf{u}^H = \sum_{i=1}^{n_u^H} \mathbf{N}_i^H d_i^H + \sum_{i=1}^{n_{\bar{u}}^H} \mathbf{N}_{i+n_u^H}^H \bar{d}_i^H \tag{5.2}$$

Diferentemente do caso fino, as funções $\mathbf{N}_i^H$ serão uma combinação linear das funções $\mathbf{N}_i^h$ calculadas a partir de problemas homogêneos locais para cada elemento $T^K \in \tau^H$ apresentados em (5.3), conforme proposto em [3].

$$\nabla_S^T (D \nabla_S \mathbf{N}_i^H) = \mathbf{0}, \text{ em } T^K \tag{5.3a}$$

$$\nabla_{||S}^T (D \nabla_{||S} \mathbf{N}_i^H) = \mathbf{0}, \text{ em } \partial T^K \tag{5.3b}$$

$$\mathbf{N}_i^H(\mathbf{x}_j) = \delta_{ij}\mathbf{e} \tag{5.3c}$$

onde $\nabla_{||S}$ está definido em (5.4) para as coordenadas $\hat{x}$ e $\hat{y}$ que representam a direção paralela e perpendicular a ∂T^K como mostrada na Figura 5.2, $\mathbf{x}_j$ representa as coordenadas do vértice relacionado ao grau de liberdade d_j^H e $\mathbf{e} = [1 \quad 0]^T$ se $\mathbf{N}_i^H$ é relacionada a um grau de liberdade x, caso contrário, $\mathbf{e} = [0 \quad 1]^T$.

$$\hat{\nabla}_{||S} = \begin{bmatrix} \frac{\partial}{\partial \hat{x}} & 0 \\ 0 & 0 \\ 0 & \frac{\partial}{\partial \hat{x}} \end{bmatrix} \tag{5.4}$$

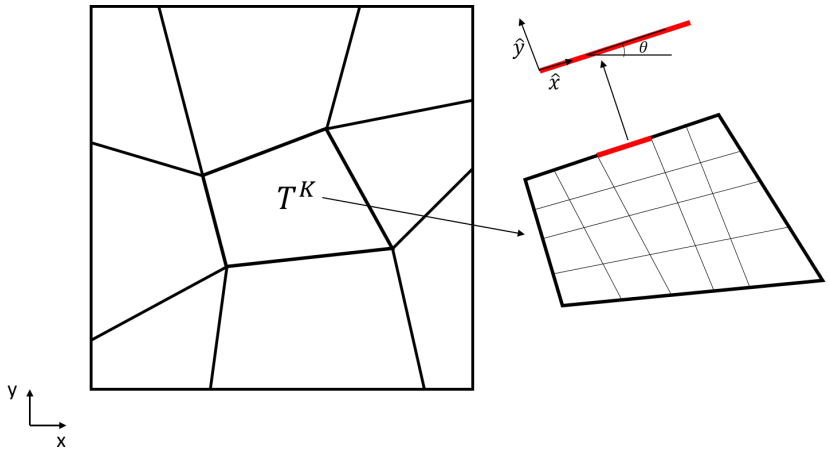


Figura 5.2: Direções para definir operador dos problemas locais na fronteira. Adaptado de [3].

Um fato importante é que (5.3a) mostra que a função de base $\mathbf{N}_i^H$ satisfaz ao mesmo operador que o problema fino, de acordo com THOMAS Y. HOU [17] essa é uma das vantagens do método multiescala, pois as funções de base tentam reproduzir o operador localmente. Essa vantagem vem com o custo da solução dos problemas locais que também de acordo com THOMAS Y. HOU [17] são caros. Estimativas do custo desse cálculos são encontradas em MANEA *et al.* [12] e MARCO BUCK [24].

Dessa forma, dado um elemento quadrilátero T^K, este possui oito funções $\mathbf{N}_i^H$ que são não nulas nesse domínio (duas para cada um dos vértices). Considerando uma numeração local do elemento e representando as funções como $\mathbf{N}_{e_i}^H$ para $i = 1, 2, \ldots, 8$. A Figura 5.3 apresenta as funções associadas a cada um dos nós na numeração local.

Para cada uma das funções $\mathbf{N}_{e_i}^H$ é necessário resolver (5.3). A solução é dividida em duas etapas, primeiro é resolvido (5.3b) utilizando como condição de contorno (5.3c), a resposta representa os valores de $\mathbf{N}_{e_i}^H$ em ∂T^K. Essa etapa pode ser realizada utilizando o método dos elementos finitos onde a matriz relativa a cada elemento, que nesse caso são segmentos de reta, pode ser encontrada no apêndice de [3].

Com os valores para fronteira T^K definidos, eles são utilizados como condição de contorno de Dirichlet para (5.3a). Esse problema também é resolvido com elementos finitos onde os resultados encontrados serão chamados de coeficientes p_{ij} que descrevem $\mathbf{N}_i^H$ como combinação linear de $\mathbf{N}_i^h$ conforme mostrado em (5.5).

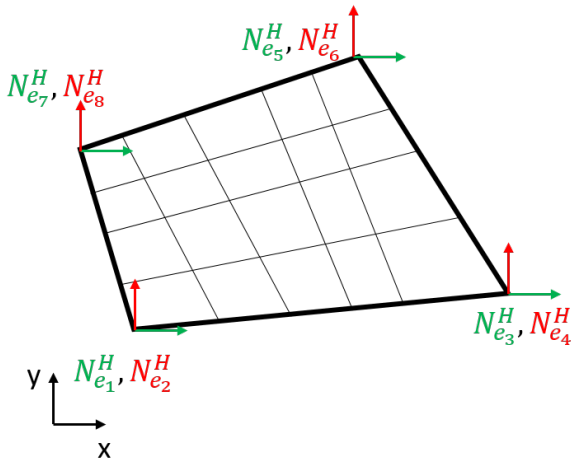


Figura 5.3: Funções de base associadas a cada um dos nós de um elementos quadrilátero. As cores verdes se referem as funções associadas a x e vermelho a y. As setas representam o valor da função no seu vértice correspondente (5.3c).

$$\mathbf{N}_i^H = \sum_{j=1}^{n_u^h} p_{ji} \mathbf{N}_j^h \tag{5.5}$$

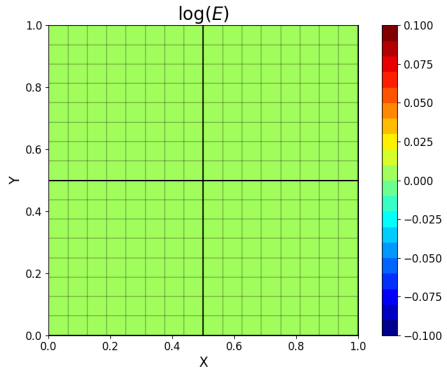
É importante perceber que os coeficientes p_{ji} fazem uma associação entre o espaço grosso e fino. A matriz $\mathbf{P}$ que será o operador de prolongamento terá justamente esses coeficientes como entradas. De forma que, $[\mathbf{P}]_{i,j} = p_{ij}$ e $\mathbf{P}$ tem dimensão $n_u^h \times n_u^H$.

A Figura 5.4 mostra um exemplo de função de base relativa a um grau de liberdade x de um grid grosso construído a partir de um grid 16×16 transformado em 2×2. Dois casos são mostrados, um em que as propriedades são constantes e outro em que existe variação no módulo de Young. No caso homogêneo a função tem forma parecida com uma função bilinear padrão de elementos finitos, já no outro caso pode-se perceber alteração na função de base afim de tentar representar as heterogeneidades das propriedades. Além disso, as Figuras 5.4(e) 5.4(f) mostram a influência dos graus de liberdade x na interpolação do deslocamento em y.

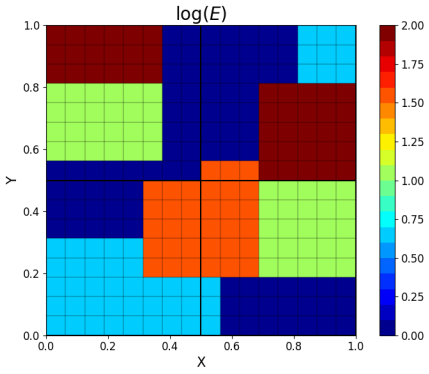
Com (5.5) as funções de base do grid grosso estão determinadas e então é possível formular um novo problema (5.6) com espaços $\mathcal{V}^H$ e $\mathcal{S}^H$ do grid grosseiro de maneira análoga a realizada em (3.3).

Encontrar $\mathbf{u} \in \mathcal{S}^H$ tal que

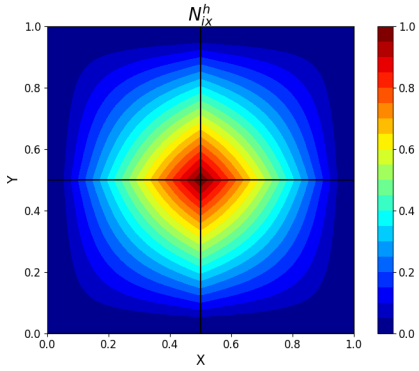
$$\int_\Omega (\nabla_S \mathbf{w})^T \mathbf{D} \nabla_S \mathbf{u} \, d\Omega - \int_{\Gamma_\sigma} \mathbf{w}^T \bar{\mathbf{t}} d\Gamma = \int_\Omega (\nabla_S \mathbf{w})^T \mathbf{m} P_p \, d\Omega \quad \forall \mathbf{w} \in \mathcal{V}^H \tag{5.6}$$



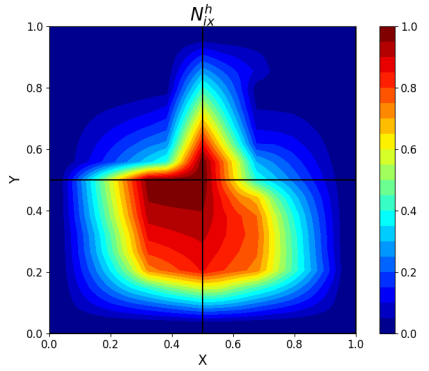
(a) Logaritmo do Módulo de Young



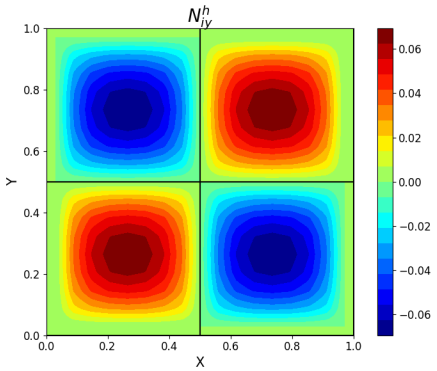
(b) Logaritmo do Módulo de Young



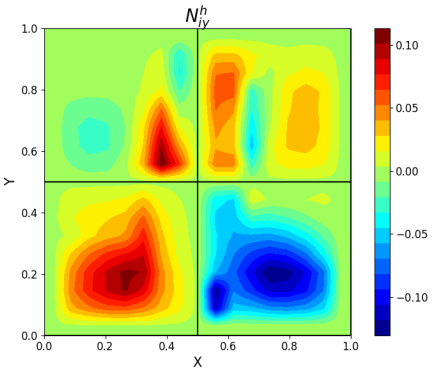
(c) $\mathbf{N}_{i_x}^H$



(d) $\mathbf{N}_{i_x}^H$



(e) $\mathbf{N}_{i_y}^H$



(f) $\mathbf{N}_{i_y}^H$

Figura 5.4: Exemplo de função de base gerada para módulo de Young homogêneo (lado esquerdo) e para módulo de Young heterogêneo (lado direito). Coeficiente de Poisson considerado constante e igual a 0.2.

onde $\mathcal{V}^H = \text{span}\{\mathbf{N}_1^H, \mathbf{N}_2^H, \ldots, \mathbf{N}_{n_u^H}^H\}$ e $\mathcal{S}^H = \text{span}\{\mathbf{N}_1^H, \mathbf{N}_2^H, \ldots, \mathbf{N}_{n_u^h+n_u^H}^H\}$. Portanto, analogamente ao grid fino, a matriz de rigidez relativa ao grid grosso tem entradas dadas por (5.7).

$$[\mathbf{K}^H]_{i,j} = \int_\Omega (\nabla_S \mathbf{N}_i^H)^T \mathbf{D} \nabla_S \mathbf{N}_j^H \, d\Omega \tag{5.7}$$

Substituindo (5.5) em (5.7).

$$\begin{aligned}
[\mathbf{K}^H]_{i,j} &= \int_\Omega (\nabla_S \mathbf{N}_i^H)^T \mathbf{D} \nabla_S \mathbf{N}_j^H \, d\Omega \\
&= \int_\Omega (\nabla_S \sum_{k=1}^{n_u^h} p_{ki} \mathbf{N}_k^h)^T \mathbf{D} \nabla_S (\sum_{l=1}^{n_u^h} p_{lj} \mathbf{N}_l^h) \, d\Omega \\
&= \sum_{k=1}^{n_u^h} \sum_{l=1}^{n_u^h} \int_\Omega p_{ki} (\nabla_S \mathbf{N}_k^h)^T \mathbf{D} \nabla_S \mathbf{N}_l^h p_{lj} \, d\Omega \\
&= \sum_{k=1}^{n_u^h} \sum_{l=1}^{n_u^h} p_{ki} \left(\int_\Omega (\nabla_S \mathbf{N}_k^h)^T \mathbf{D} \nabla_S \mathbf{N}_l^h \, d\Omega \right) p_{lj} \\
&= \sum_{k=1}^{n_u^h} \sum_{l=1}^{n_u^h} \int_\Omega p_{ki} [\mathbf{K}^h]_{k,l} p_{lj} \, d\Omega \\
&= \sum_{k=1}^{n_u^h} \sum_{l=1}^{n_u^h} \int_\Omega [\mathbf{P}]_{k,i} [\mathbf{K}^h]_{k,l} [\mathbf{P}]_{l,j} \, d\Omega \\
&= \sum_{k=1}^{n_u^h} \sum_{l=1}^{n_u^h} \int_\Omega [\mathbf{P}^T]_{i,k} [\mathbf{K}^h]_{k,l} [\mathbf{P}]_{l,j} \, d\Omega
\end{aligned} \tag{5.8}$$

Portanto, a (5.8) mostra que o operador grosseiro pode ser construído utilizando (5.9) diferentemente de realizar loop nos elementos grossos como o realizado pelo método de elementos finitos clássico.

$$\mathbf{K}^H = \mathbf{P}^T \mathbf{K}^h \mathbf{P} \tag{5.9}$$

De forma análoga, pode-se encontrar que a relação entre os lados direito dos dois operadores é $\mathbf{f}^H = \mathbf{P}^T \mathbf{f}^h$. Assim, o operador de restrição aqui será $\mathbf{R} = \mathbf{P}^T$. Com isso, dado um sistema linear da forma apresentada em (3.11) é possível encontrar (5.10) como aproximação para a solução do grid fino.

$$d^h \approx d_{ms}^h = \mathbf{P} (\mathbf{K}^H)^{-1} \mathbf{P}^T \mathbf{f}^h \tag{5.10}$$

Essa aproximação consiste de três passos que são descritas abaixo:

- A multiplicação por $\mathbf{P}^T$ representa a restrição do lado direito para o espaço grosseiro.
- o termo $(\mathbf{K}^H)^{-1}$ representa o cálculo da solução do espaço grosseiro.
- a multiplicação por $\mathbf{P}$ é o prolongamento da solução do espaço grosseiro para espaço fino.

Inicialmente, a solução aproximada (5.2) foi utilizada como solução aproximada para o grid fino em [17]. No caso, tomando como base (5.10), pode-se definir $\mathbf{M}_{\text{ms}}^{-1} = \mathbf{P}(\mathbf{K}^H)^{-1}\mathbf{P}^T$ como um precondicionador multiescala. Em ZHOU e TCHELEPI [25], a solução do sistema é encontrado através de iterações de Richardson junto com um precondicionador do espaço fino $\mathbf{M}_{\text{h}}^{-1}$, pois o artigo mostra que $\mathbf{M}_{\text{ms}}^{-1}$ não pode ser utilizado sozinho devido a sua convergência não ser garantida visto que é deficiente de posto tendo autovalores 0 e, portanto, a matriz de iteração $(\mathbf{I} - \mathbf{M}_{\text{ms}}^{-1}\mathbf{A})$ tem autovalores iguais a um. Essa combinação é feita de maneira multiplicativa de forma que o precondicionador conjunto é dado por (5.11).

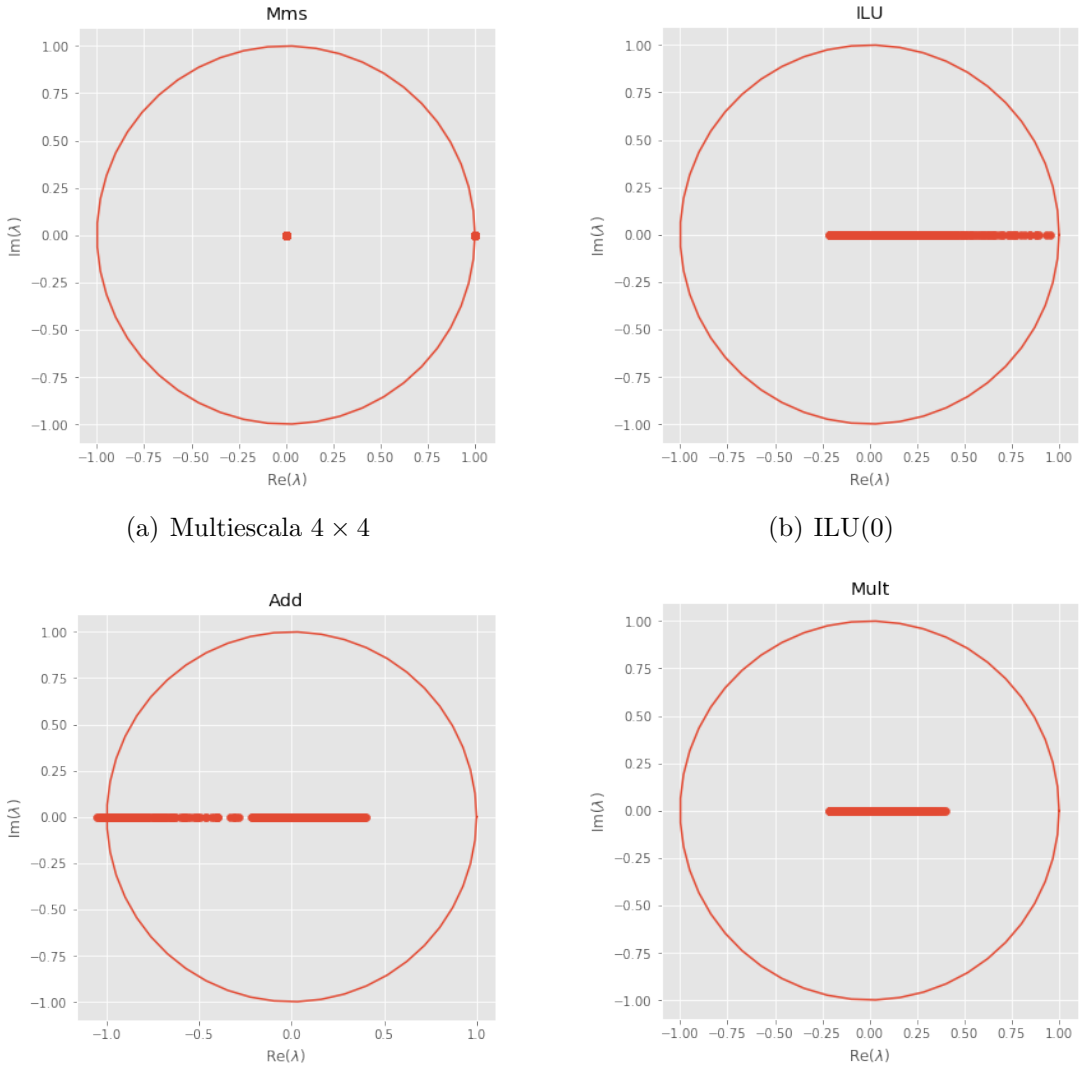
$$\mathbf{M}^{-1} = \mathbf{M}_{\text{ms}}^{-1} + \mathbf{M}_{\text{h}}^{-1}(\mathbf{I} - \mathbf{K}^h\mathbf{M}_{\text{ms}}^{-1}) \tag{5.11}$$

Outra forma de realizar a combinação dos dois precondicionadores é de maneira aditiva que é dado por (5.12) que será abordada nessa dissertação. É importante perceber que o precondicionador multiplicativo tem um produto matriz vetor intrínseco devido ao termo $\mathbf{K}^h$ que aparece, enquanto o aditivo não possui.

$$\mathbf{M}^{-1} = \mathbf{M}_{\text{h}}^{-1} + \mathbf{M}_{\text{ms}}^{-1} \tag{5.12}$$

Visando reproduzir os resultados de [25] para os operadores de elasticidade linear, a Figura 5.5 apresenta os autovalores relativos a um caso homogêneo de um grid 40×40 para a matriz de iteração de Richardson para quatro diferentes precondicionadores: multiescala, ILU(0), multiplicativo (multiescala + ILU(0)) e aditivo (multiescala + ILU(0)).

Pode-se verificar que o precondicionador multiescala não tem convergência garantida, em concordância com a prova mostrada em [25]. O combinativo multiplicativo apresenta bom resultado, pois os autovalores aparecem próximos de zero, entretanto, o combinativo aditivo não tem convergência garantida o que mostra que não é interessante utilizá-lo com iterações de Richardson. Ainda sobre isso, [3] mostra que o operador multiplicativo tem melhor desempenho quando utilizado junto com um método de Krylov, naquele contexto utilizando o Bicgstab, do que quando utilizado com o método de Richardson, inclusive, mostra que para várias configurações



(a) Multiescala 4×4

(b) ILU(0)

(c) Aditivo (Multiescala + ILU(0))

(d) Multiplicativo (Multiescala + ILU(0))

Figura 5.5: Autovalores para matriz de iteração de Richardson para caso homogêneo e isotrópico. Grid de tamanho 40×40.

o solver não consegue convergir. Dado esse resultado, o presente trabalho vai fazer comparações entre os métodos utilizando métodos de Krylov.

A Figura 5.6 apresenta os valores da matriz de rigidez pré-condicionada ($\mathbf{M}^{-1}\mathbf{K}^h$) para o mesmo grid de 40×40, os gráficos apresentam resultados para engrossamento 2x2, 4x4 e 8x8, todas as figuras são apresentadas com os mesmos eixos para ser possível se observar as espalhamento dos autovalores.

É fato que nos dois casos o aumento do engrossamento causa um maior espalhamento dos autovalores, desse modo é de se esperar que sejam necessárias mais iterações para a convergência do solver. Além disso, o precondicionador aditivo possui distribuição pior dos autovalores em relação ao multiplicativo e, portanto, é esperado que utilize mais iterações. Esse aumento tem que ser compensado com uma iteração mais barata por conta da multiplicação matriz vetor que não é realizada

pelo precondicionador aditivo.

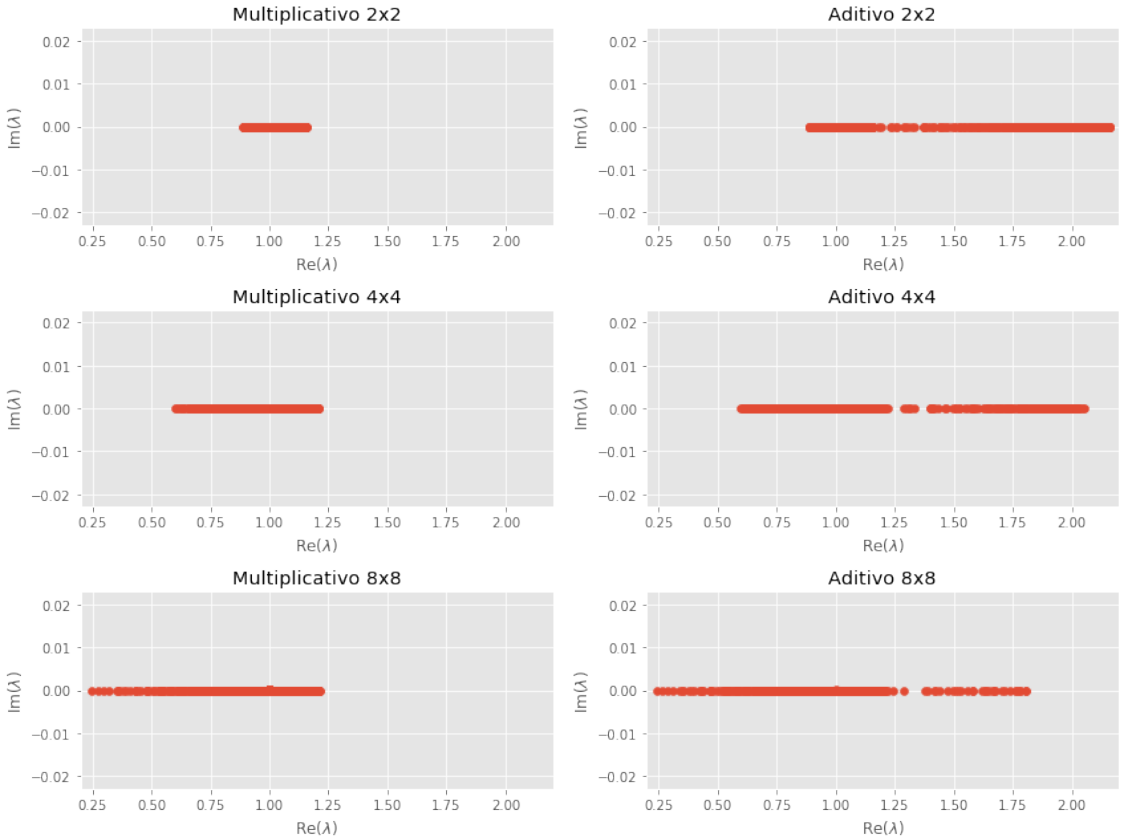


Figura 5.6: Autovalores da matriz pré-condicionada para caso homogêneo.

5.2 Construção do Operador de Prolongamento

A solução dos problemas locais descritos na seção anterior tem como resultado as entradas p_{ij} do operador em cada um dos elemento de τ^H. Para encontrar esses valores é necessário primeiro construir a matriz de rigidez relativa ao elemento em questão que denominaremos de $\mathbf{K}^{T^K}$. Essa matriz será utilizada para a solução de todas as oito funções de base relacionadas com esse elemento, uma maneira de adicionar mais facilmente as condições de contorno nesse caso é utilizar o método de condição de contorno apresentado em (3.37), dessa forma, o lado direito do sistema conterá apenas os valores da condição de contorno de Dirichlet que é justamente a solução de (5.3b).

Com o Algoritmo 8 é possível obter $\mathbf{P}$ e, com ele, obter $\mathbf{K}^H$ através de (5.9) e a construção do precondicionador $\mathbf{M}_{\text{ms}}^{-1} = \mathbf{P}(\mathbf{K}^H)^{-1}\mathbf{P}^T$ fica completa. Uma representação matricial do prolongamento é mostrada em (5.13), onde cada coluna representa uma função de base (essa representação possui um abuso de notação visto que $\mathbf{N}_i^H$ representa uma função e não entradas da matriz $\mathbf{P}$).

Algoritmo 8: Cálculo do Prolongamento

início
  $\mathbf{P} \leftarrow \mathbf{0}$
  para cada *elemento* $T^K \in \tau^H$ **faça**
    Calcular Matriz de Rigidez $\mathbf{K}^{T^K}$ do Problema Local de T^K
    para $i = 1, \ldots, 8$ **faça**
      Resolver (5.3b) em ∂T^K com condição de contorno (5.3c)
      Construa $\mathbf{f}^{T^K}$ com valores da solução do problema de fronteira.
      $\mathbf{P}_{\text{local}} \leftarrow (\mathbf{K}^{T^K})^{-1}\mathbf{f}^{T^K}$
      Atribua em $\mathbf{P}$ o valores $\mathbf{P}_{\text{local}}$ nas posições correspondentes
    fim
  fim
fim

$$\mathbf{P} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \mathbf{N}_1^H & \mathbf{N}_2^H & \ldots & \mathbf{N}_{n_u^H}^H \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{5.13}$$

5.3 Cálculo do NNZ do operador de prolongamento

Em um método multiescala, além da construção de um operador grosso, precisa-se realizar transformações dos vetores entre os espaços fino e grosso. Essas transformações são realizadas através de multiplicação pelos operadores $\mathbf{P}$ e $\mathbf{P}^T$ e, essa seção, descreve uma estimativa para a quantidade de não zeros desses operadores.

As dimensões do operador de prolongamento dependem dos graus de liberdade do operador fino e do grosso, valendo $n_u^h \times n_u^H$. Isso e importante pois, conforme visto no Capítulo 4, a multiplicação de uma matriz esparsa por um vetor é da ordem do número de elementos não nulos da matriz, assim, apesar de um maior engrossamento do grid reduzir o número de colunas de $\mathbf{P}$, não necessariamente ocorre uma redução dos elementos não nulos. Uma motivação para esse fato é que apesar da quantidade de colunas do operador de prolongamento diminuir quando há um engrossamento maior, cada coluna precisa ter uma quantidade maior de não zeros, pois o alcance das funções de base aumenta. A Figura 5.7 apresenta o mesmo grid com engrossamento 3×3 e 4×4, olhando para a quantidade de não zeros da linha do prolongamento relativa ao nó em verde nos dois casos é a mesma. Isso ocorre, pois, os dois tem influência de funções de base de quatro nós vermelhos em ambos os casos.

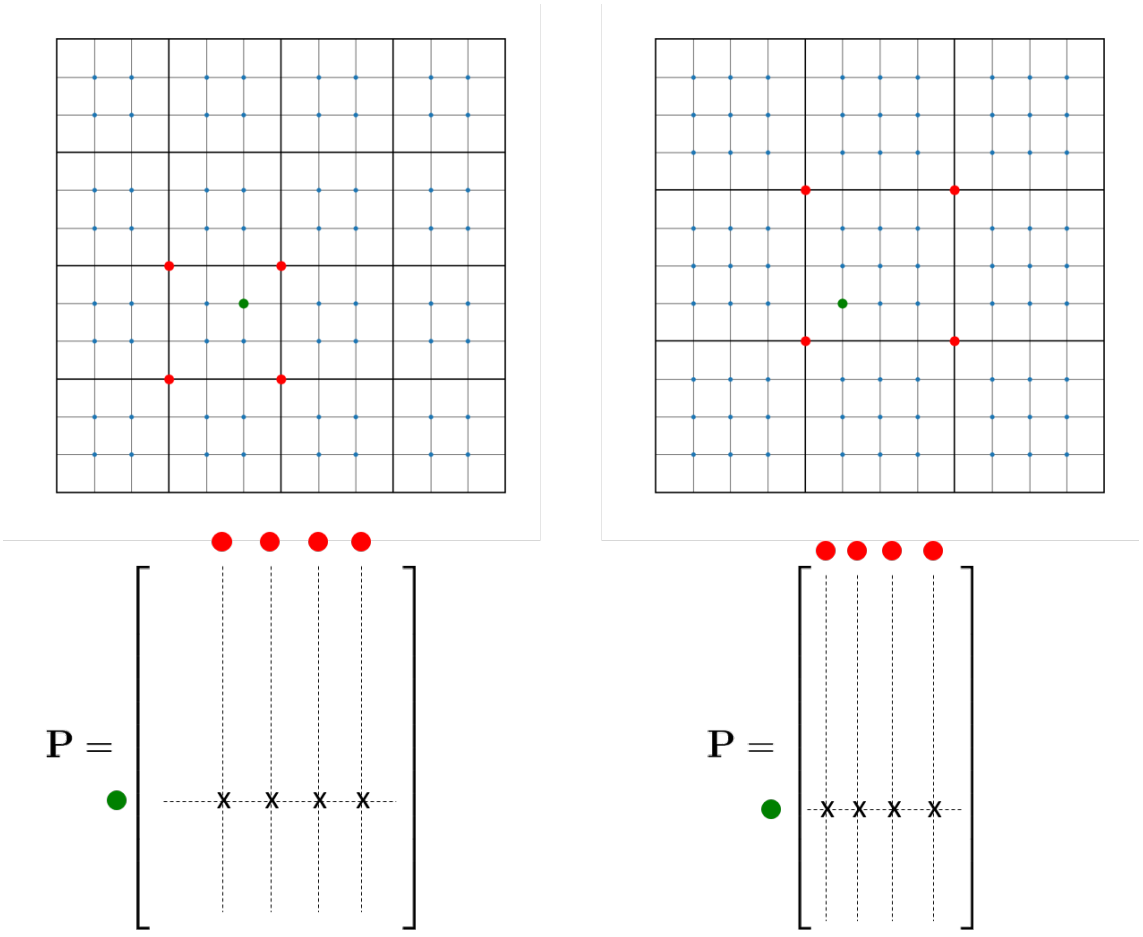


Figura 5.7: Nós interiores dos elementos grossos para um grid fino 8x8 transformado em um grid grosso 2x2.

Considerando agora um grid fino com $n_{e_x}^h \times n_{e_y}^h$ elementos, um grid grosso onde cada elemento tem dimensões $C_{r_x} \times C_{r_y}$ e, ainda, que mesmo os nós de fronteira com condição de contorno de Dirichlet não são removidos da matriz de rigidez, pode-se encontrar um limite inferior para a quantidade de não zeros do operador de prolongamento (nnz_p) considerando apenas os nós que estão no interior de um elemento grosso. Por exemplo, a Figura 5.8, mostra os nós interiores das células grossas para um grid 16×16 transformado em um grid de 2×2.

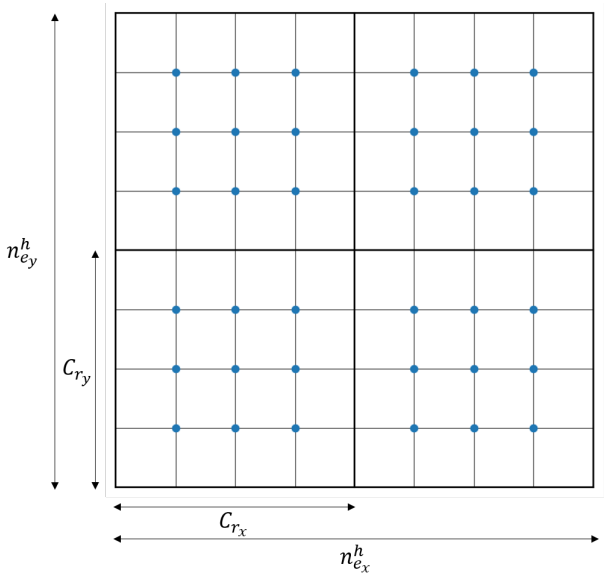


Figura 5.8: Quantidade de não zeros da linha do pronlogamento do nó verde para dois diferentes engrossamento 3×3 e 4×4.

No caso, cada célula grossa possui $(C_{r_x} - 1)(C_{r_y} - 1)$ nós interiores. Cada um dos dois graus de liberdade de um nó interior recebe influência de cada uma das oito funções de base relacionadas com os vértices do elemento grosso. Dessa forma pode-se escrever (5.14).

$$\text{nnz}_p \geq \underbrace{(\frac{n_{e_x}^h}{C_{r_x}} \times \frac{n_{e_y}^h}{C_{r_y}})}_{\text{qtd. elem. grossos}} \times 8 \times 2 \times (C_{r_x} - 1)(C_{r_y} - 1) \tag{5.14}$$

Desenvolvendo,

$$\begin{aligned} \text{nnz}_p &\geq 16 \times \frac{n_{e_x}^h}{C_{r_x}}(C_{r_x} - 1) \times \frac{n_{e_y}^h}{C_{r_y}}(C_{r_y} - 1) \\ &\geq 16(n_{e_x}^h - \frac{1}{C_{r_x}})(n_{e_y}^h - \frac{1}{C_{r_y}}) \\ &\geq 16(n_{e_x}^h - 1)(n_{e_y}^h - 1) \end{aligned} \tag{5.15}$$

Assim, (5.15) mostra que nnz_p é no mínimo na ordem do número de elementos

do grid fino $n^h_{e_x} \times n^h_{e_y}$. Um argumento análogo pode ser feito para mostrar que a quantidade de não zeros não é maior que a ordem da quantidade de elementos finos, basta majorar nnz_p considerando que todos os nós contribuem com o mesmo número de não zeros de um nó interior, visto que são eles que tem a maior quantidade de funções de base com valor diferente zero, em comparação com os nós pertencentes as arestas e vértices dos elementos grossos. Dessa forma, pode-se escrever (5.16).

$$\text{nnz}_p \leq 8 \times 2 \times \underbrace{(n^h_{e_x} + 1)(n^h_{e_y} + 1)}_{\text{qtd. nós finos}} \tag{5.16}$$

Portanto, independente do nível engrossamento, a quantidade de não zeros do prolongamento terá não zeros da ordem da dimensão do grid fino. Esse fato é importante, pois, a partir de um certo ponto, não será vantajoso engrossar mais o grid por conta do preço da multiplicação matriz vetor pelo prolongamento.

Capítulo 6

Implementação Computacional

Nesse capítulo serão apresentados detalhes da implementação realizada nesse trabalho. O código foi desenvolvido em Python com auxílio da biblioteca PETSc (BALAY *et al.* [26]). O PETSc é uma biblioteca famosa com diversas rotinas para computação científica, possui paralelismo utilizando MPI, estruturas de dados para matrizes esparsas, métodos de Krylov para solução de sistemas lineares, conjunto de precondicionadores (em particular os ILU), dentre outros. Estudos da performance da biblioteca podem ser encontrados em BALAY *et al.* [27]. O PETSc foi utilizado com a linguagem Python através dos *bindings* disponibilizados no petsc4py (DALCIN *et al.* [28]). O código aqui apresentado foi desenvolvido em serial.

6.1 Estrutura de Classes

Primeiramente, foi necessário desenvolver uma estrutura para resolver problemas através do método dos elementos finitos, para depois realizar a implementação com o método multiescala. Esse desenvolvimento possui três principais classes: **Element**, **Node**, **FemContext**. As responsabilidades das classes são descritas abaixo:

- **Node**: essa classe tem como intuito representar os nós do problema de elementos finitos. Guarda como propriedade as coordenadas x, y correspondentes, uma numeração global associada ao grid fino e se o nó pertence ou não a uma fronteira com condição de Dirichlet.

- **Element**: essa classe tem como intuito representar os elementos do problema de elementos finitos. Tem referência para os nós que pertencem ao elemento, guarda os valores do coeficiente de Young e Poisson, sabe calcular as funções de forma em coordenadas locais conforme (3.23) e também é responsável por calcular a matriz do elemento local conforme (3.22).

- **FemContext**: essa classe tem como intuito representar um problema de elementos finitos. Ela possui o conjunto de elementos e nós do problema em questão. Dá a numeração de cada um dos graus de liberdade dos nós. Calcula a matriz de rigidez através do Algoritmo 1 e o lado direito correspondente.

Essas três classes conjuntamente são capazes de montar e solucionar um problema de elementos finitos. Um fato importante é que a classe **FemContext** é responsável pela numeração dos graus de liberdade, pois, para calcular as funções de base, diferentes problemas locais tem que ser resolvidos e diferentes numeração locais são necessárias. Em uma implementação mais clássica de elementos finitos, o mais natural seria a numeração do grau de liberdade estar associada com a classe **Node**.

6.2 Montagem dos Operador Multiescala

Para a implementação do método multiescala, foi criada a classe **Multiscale**, essa classe tem como responsabilidade receber um objeto **FemContext** que representa o grid fino e calcular os operadores de prolongamento ($\mathbf{P}$), restrição ($\mathbf{P}^T$) e operador grosso ($\mathbf{K}^H$).

O primeiro passo para utilizar o método multiescala é a escolha do nível de engrossamento. A implementação aceita diferentes fatores para a direção x e y, apesar dos testes serem apresentados com o mesmo fator. O método **CoarseContext** da classe **Multiscale** é responsável por realizar a tarefa de construir $\mathbf{P}$ e $\mathbf{K}^H$. Essa tarefa é dividida nos seguintes passos:

- Criar objetos **FemContext** para cada um dos elementos grossos T^K.
- Adicionar os nós e elementos correspondentes em cada um dos seus respectivos **FemContext**
- Para cada **FemContext** calcular a matriz de rigidez $\mathbf{K}^{T^K}$.
- Resolver os oito problemas associados as funções de base encontrando $\mathbf{N}^H_{e_1}, \mathbf{N}^H_{e_2}, \ldots, \mathbf{N}^H_{e_8}$.
- Fazer o Assembly do operador de prolongamento.
- Calcular $\mathbf{K}^H = \mathbf{P}^T\mathbf{K}^h\mathbf{P}$

A Figura 6.1 apresenta um esboço da construção desse operador. Pode-se perceber que os problemas de cada **FemContext** são independentes, exceto pela solução dos problemas das arestas em comum. A implementação desse trabalho resolveu as

arestas da fronteira duas vezes, uma implementação mais otimizada pode resolver apenas uma vez cada problema de fronteira. Em relação a resolver os oito problemas locais, foi realizada a fatoração LU da matriz que foi reaproveitada para a solução desses problemas.

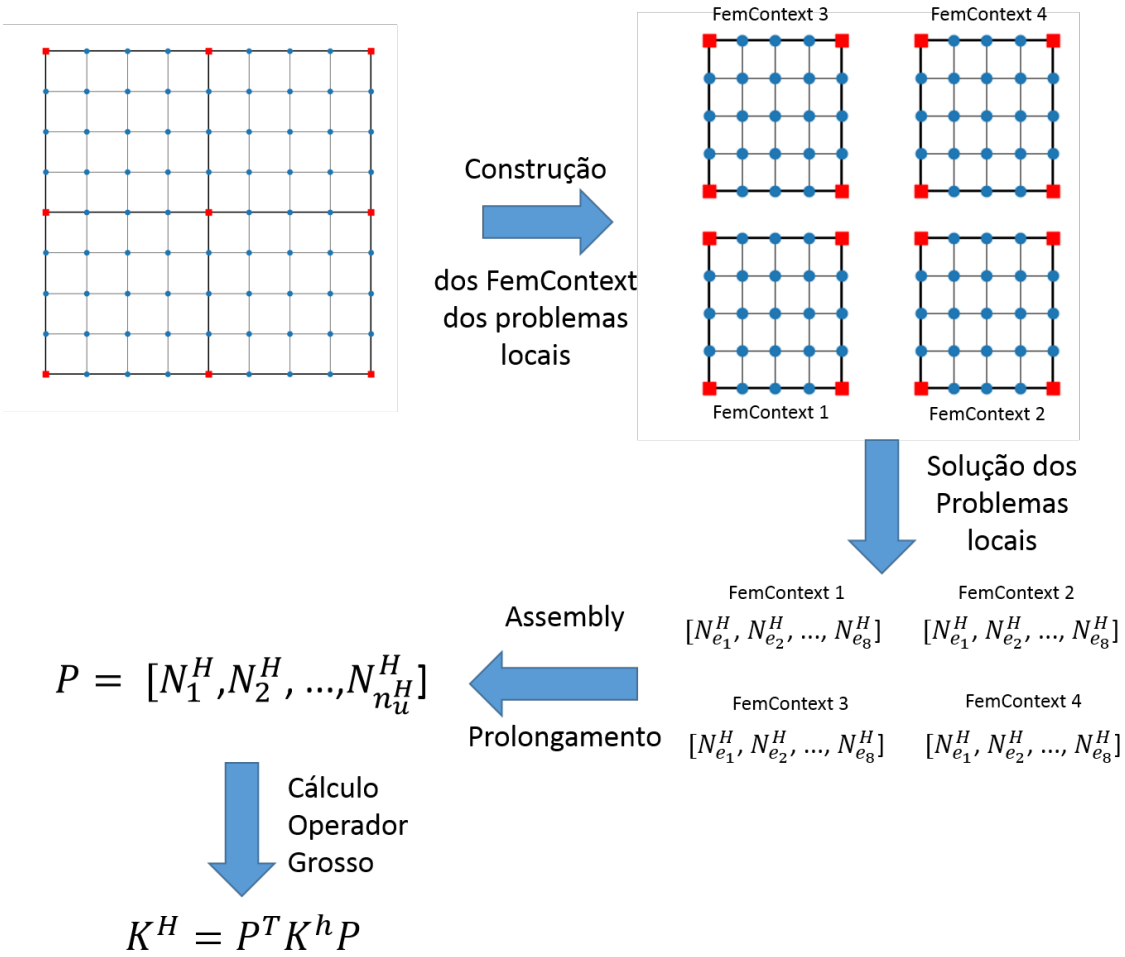


Figura 6.1: Esquema de construção dos operadores grossos. Mostrando um grid 8x8 sendo descomposto em um grid grosseiro 2x2 onde cada elemento grosso é formado por 4x4 elementos finos.

O cálculo das matrizes dos operadores locais $\mathbf{K}^{T^K}$ tem os mesmos coeficientes presentes na matriz de rigidez $\mathbf{K}^h$ visto que o operador de (2.16) é o mesmo de (5.3a). Assim, para o código executar mais rápido a matriz de rigidez de cada elemento foi armazenada para não precisar ser calculada novamente, isso torna o custo da memória maior, pois esses valores não precisam ser necessariamente guardados. Outra opção é perceber que a matriz de um operador local é uma submatriz do operador original e, portanto, poderia ser utilizado o método **GetSubMatrix** das matrizes do Petsc para construir $\mathbf{K}^{T^K}$ a partir de $\mathbf{K}^h$ conforme mostra a Figura 6.2. Essa ideia é similar a utilizar uma reordenação Wire-Basket como apresentada em [3] e [8].

A classe **Multiscale** também possui um método solve, que só pode ser chamado

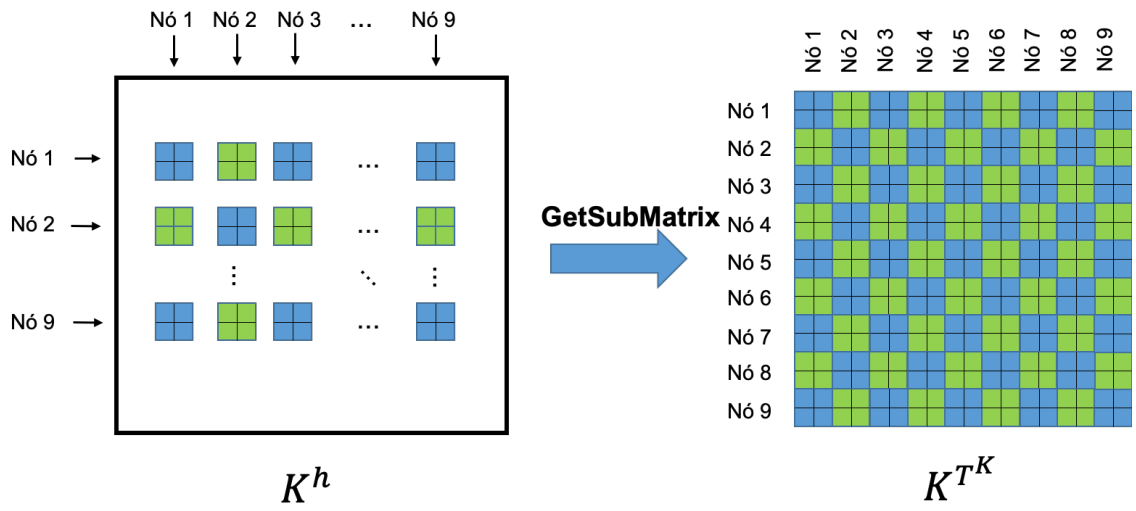


Figura 6.2: Exemplo de construção de uma matriz relacionada a um problema local através do **GetSubMatrix**.

após o método **CoarseContext** ter sido executado. Esse método recebe como entrada um vetor $\mathbf{f}^h$ relacionado a um lado direito do grid fino e resolve o problema no grid grosso e retorna para o grid fino. Em outras palavras, esse método retorna o seguinte valor $\mathbf{P}(\mathbf{K}^H)^{-1}\mathbf{P}^T\mathbf{f}^h$. Para a solução do sistema $(\mathbf{K}^H)^{-1}$ a solução utilizada foi uma fatoração LU. Essa fatoração é guardada para ser utilizada entre as iterações do método de Krylov quando o objeto o método **Multiescala** está sendo utilizado como precondicionador.

Capítulo 7

Experimentos Numéricos

Nesse capítulo, serão apresentados os resultados obtidos utilizando o método multiescala para solução dos sistemas lineares decorrentes do operador apresentado no Capítulo 2. Os resultados são apresentados na seguinte ordem: resultados em casos que a solução analítica é conhecida, visualização das soluções do espaço grosseiro, comparação entre precondicionador aditivo e multiplicativo, comparações do precondicionador multiescala, multigrid e ILU e estudo do número de iterações do solver linear com variação da tolerância do solver grosso. As comparações serão realizadas apenas para a solução dos sistemas lineares e não serão consideradas os tempos de construção dos precondicionadores. Apesar desse tempo ser relevante, ele pode ser diluído se o precondicionador for reutilizado para a solução de diferentes sistemas. As execuções foram realizadas no processador Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz.

7.1 Soluções Analíticas

Inicialmente, é necessário atestar que o código de elementos finitos está resolvendo corretamente o operador descrito em (2.16). Para tanto, foi montado um teste semelhante ao proposto em [8], que consiste em resolver o problema que possui como solução analítica apresentada em (7.1) em um domínio $[0, L] \times [0, W]$.

$$\begin{aligned} u_x &= 10^{-5} sen(\frac{\pi x}{L}) sen(\frac{\pi y}{W}) \\ u_y &= 10^{-5} cos(\frac{\pi (L-x)}{L}) sen(\frac{\pi y}{W}) \end{aligned} \tag{7.1}$$

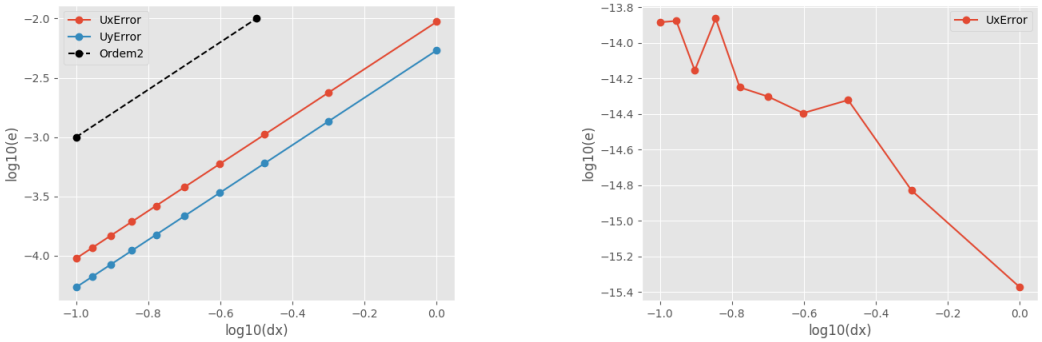
Dada a solução, é possível calcular o lado direito correspondente a essa solução aplicando o lado esquerdo de (2.16) e obtendo a função $\mathbf{f} : R^2 \to R^2$. Dessa forma, as equações que representam o problema resolvido são apresentadas em (7.2) onde são utilizadas condições de Dirichlet na fronteira com os valores da solução (7.1).

$$\begin{aligned} \nabla_S^T \mathbf{D} \nabla_S \mathbf{u} &= \mathbf{f}(x, y), \\ L = W &= 10. \end{aligned} \tag{7.2}$$

O erro entre a solução calculada pelo método dos elementos finitos pode então ser comparada com a solução analítica através do erro relativo mostrado em (7.3).

$$\epsilon_\infty = \frac{|\mathbf{u}_{fem} - \mathbf{u}_{ref}|_\infty}{|\mathbf{u}_{ref}|_\infty} \tag{7.3}$$

onde $u_{ref} = [u_x(x_0, y_0), u_y(x_1, y_1), ..., u_x(x_{n_n}, y_{n_n})]^T$ é a solução analítica e u_{fem} é a solução obtida com o método dos elementos finitos. A variação do erro com o tamanho da discretização do domínio é mostrado na Figura 7.1(a) onde no eixo x o logaritmo do tamanho de cada elemento (Δx) de cada elemento e no eixo y o logaritmo de ϵ_∞. Também como referência é mostrada uma reta de coeficiente angular dois para comprovar o decaimento quadrático do erro.



(a) (b)

Figura 7.1: Gráficos do logaritmo do erro (7.3) em função do logaritmo do tamanho Δx dos elementos do grid.

Um segundo caso com resultado analítico é o cisalhamento puro (*Simple Shear*). Nesse caso, o problema é definido na forma apresentada em (7.4) e solução é dada por $\mathbf{u}(x, y) = [10^{-6}y, 0]^T$ em um domínio $\Omega = [0, 1] \times [0, 1]$. Como a solução é um polinômio do primeiro grau, essa pode ser representada exatamente no espaço gerado pelas funções de base bilineares e, então, os erros de truncamento, nesse caso, não existem, restando apenas erros de arredondamento. Os erros de truncamento estão relacionados com a aproximação realizada pelo método dos elementos finitos enquanto que os erros de arredondamento estão relacionados às aproximações feitas pela aritmética de ponto flutuante.

Tabela 7.1: Tabela com os casos que serão apresentados os resultados.

Nome	$n^h_{e_x}$	$n^h_{e_y}$	**Caso Real**
caso A	100	100	Não
caso B	320	320	Não
caso C	103	56	Não
caso D	244	71	Sim
caso E	582	336	Sim

$$\begin{aligned} \nabla_S^T \mathbf{D} \nabla_S \mathbf{u} &= \mathbf{0}, \\ \mathbf{u} &= [10^{-6}, 0]^T, \text{ em } \{x, y \in \Gamma | y = 1\}, \\ \mathbf{u} &= [10^{-6} y, 0]^T, \text{ em } \{x, y \in \Gamma | x = 0 \text{ ou } x = 1\}, \\ \mathbf{u} &= [0, 0]^T, \text{ em } y = 0. \end{aligned} \tag{7.4}$$

A variação do erro com o tamanho da malha é apresentado na Figura 7.1(b) e pode-se observar que nesse caso o maior erro relativo é menor que 10^{-12} que é bem menor que o do caso anterior, por conta do erro de truncamento ser zero, e também não decai com o tamanho da malha.

Atestado o bom funcionamento do método dos elementos finitos, os resultados das próximas seções irão tomar como referência as soluções no grid fino e serão comparadas com os resultados do método multiescala.

7.2 Modelos de simulação utilizados

A Tabela 7.1 apresenta os tamanhos dos modelos utilizados neste trabalho. Cinco modelos foram selecionados onde dois deles são de modelos de campos reais. Os casos A e B são baseados nos casos sintéticos apresentados em [3] e [8], onde foram utilizados uma compressibilidade vertical uniaxial c_M desenvolvida em BAU *et al.* [29] apresentada em (7.5), onde c_M e σ''_{yy} são expressas em [bar^{-1}] e [bar]. σ''_{yy} representa a tensão vertical efetiva, e a tensão efetiva na rocha é calculada através de (7.6) que considera um gradiente de pressão de 0.1 bar/m e coeficiente de Biot igual a um.

$$c_M = 0.01241 |\sigma''_{yy}|^{-1.1342} \tag{7.5}$$

$$\sigma''_{yy} = \sigma_{yy} - P_p = 0.12218|y| - 0.1|y| \tag{7.6}$$

$$E = \frac{(1-2\upsilon)(1+\upsilon)}{(1-\upsilon)c_M} \tag{7.7}$$

Assim, o módulo de Young pode ser calculado em função apenas da profundidade substituindo os valores de (7.5) e (7.6) em (7.7). Já o coeficiente de Poisson foi considerado constante e igual a 0.2 em todo o modelo.

O grid utilizado é apresentado na Figura 7.2 com as dimensões baseadas no exemplo mostrado em [3]. Esse grid teve cada uma das células divididas com 10 cortes verticais e 10 cortes horizontais igualmente espaçados para gerar o caso A e, de forma análoga, dividida em 32 cortes horizontais e 32 cortes verticais para gerar o caso B. A localização do reservatório aparece na Figura 7.2 em vermelho. Foi considerada também a depleção de 100 bar de pressão no reservatório.

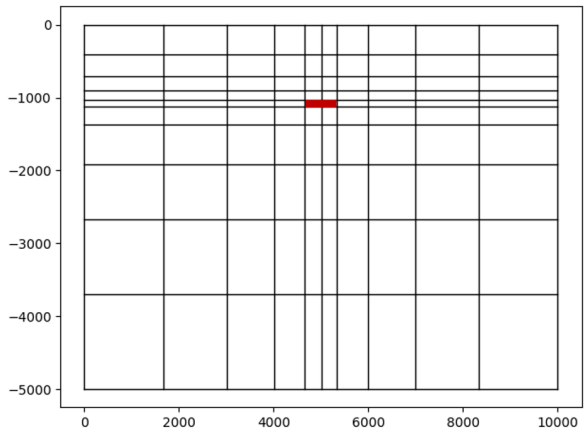


Figura 7.2: Grid base utilizado para construção dos casos A e B.

O caso C é um caso de reservatório sintético com grid de geometria mais próxima aos reservatórios reais e, diferentemente dos casos A e B, os valores de Poisson não são constantes ao longo do domínio. A Figura A.1 no Apêndice A apresenta o grid e os valores do módulo de Young e módulo de Poisson. São apresentados também no apêndice Figuras dos casos D e E.

As condições de contorno utilizadas para todos os modelos são representadas na Figura 7.3, a base do modelo é considerada fixa ($\mathbf{u} = 0$), as bordas laterais tem movimento possível apenas na direção y e topo do modelo é livre de tração.

7.3 Resultados Método Multiescala

7.3.1 Visualização da solução grossa

Dado que o operador multiescala gera um operador grosso com solução que tenta reproduzir a solução do espaço fino, é possível comparar visualmente e também

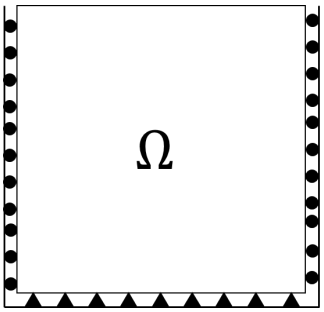


Figura 7.3: Esquema das condições de contorno das simulações. Borda inferior com deslocamentos nulos, bordas laterais com deslocamentos em y permitidos e borda superior livre.

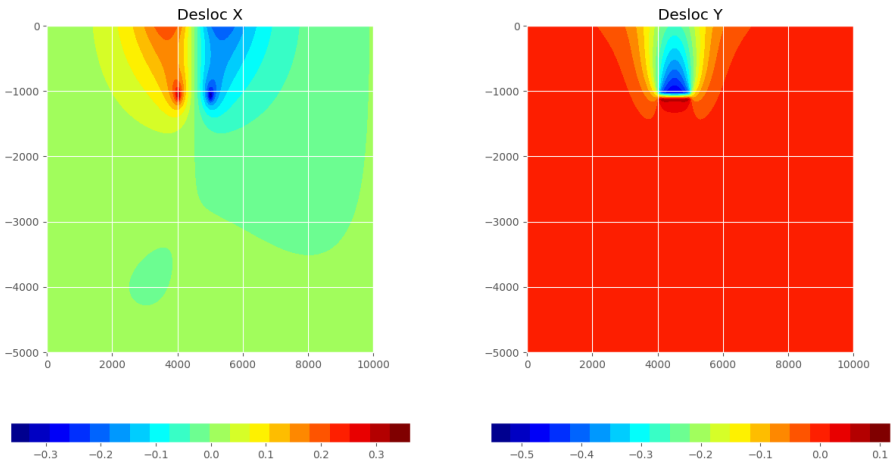
Tabela 7.2: Erro relativo da solução fina em relação a grossa para diferentes níveis de engrossamento.

Elemento Grosso ($C_{r_x} \times C_{r_y}$)	**Erro Relativo**
8x8	0.124267
16x16	0.190326
32x32	0.273388
64x64	0.763958

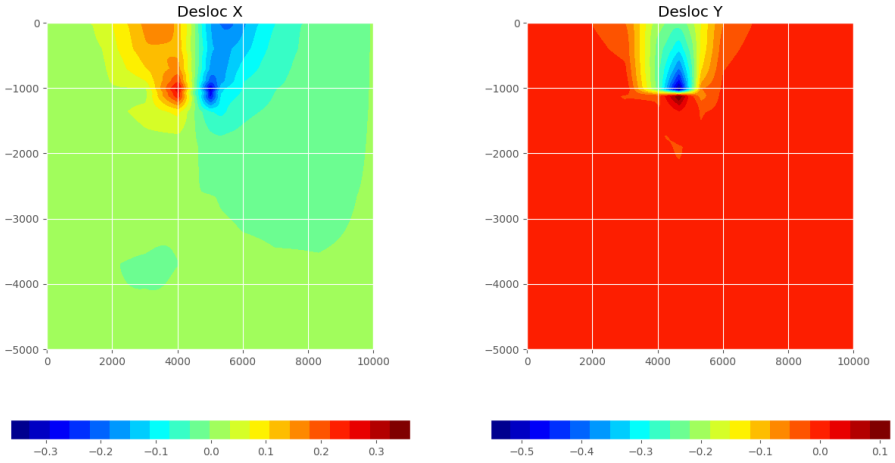
através do erro entre a solução fina e a solução da malha grossa. A aproximação para a solução da malha fina $\mathbf{u}^h_{ms} = \mathbf{P}\mathbf{u}^H$ é utilizada nessa comparação de forma que o erro que será apresentado é mostrado em (7.8).

$$\epsilon_\infty = \frac{|\mathbf{u}^h - \mathbf{P}\mathbf{u}^H|_\infty}{|\mathbf{u}^h|_\infty} \tag{7.8}$$

As Figuras 7.4 apresentam a comparação entre a solução do grid fino e grosso com fator de engrossamento 32×32. Pode-se notar que a solução é distorcida mas mesmo assim guarda similaridades com a solução original. Para uma avaliação quantitativa é apresentada a Tabela 7.2, a primeira coluna apresenta o fator de engrossamento e a segunda o erro relativo. Fica claro que quanto maior o nível de engrossamento, mais distante da solução original da malha fina, com o engrossamento de 64×64 já mostrando uma solução bem afastada da original. A Figura 7.5 mostra a subsidência do topo do domínio para a solução fina e grossa e, pode-se constatar, uma boa aproximação da subsidência pelo grid grosso.



(a) Solução do grid fino. À esquerda, o deslocamento em x e, à direita, o deslocamento em y



(b) Solução do grid grosso. À esquerda, o deslocamento em x e, à direita, o deslocamento em y

Figura 7.4: Comparação da solução do grid fino com a solução do grid grosso para engrossamento 32×32.

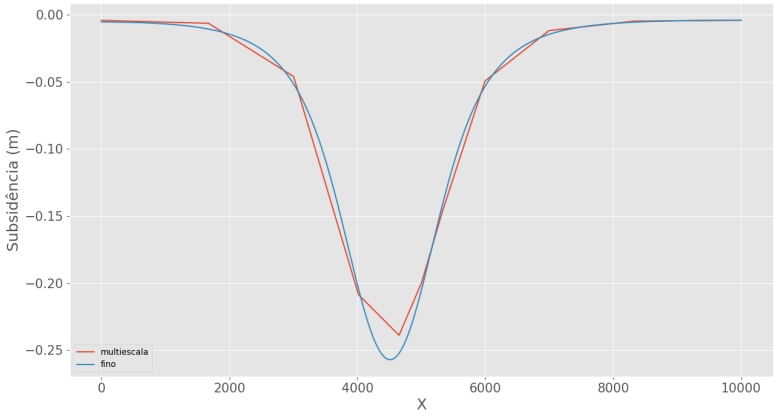


Figura 7.5: Subsidência do topo do caso B, em azul na solução do grid fino e de vermelho a solução do grid grosso (multiescala).

7.3.2 Comparação entre precondicionadores aditivos e multiplicativos

O trabalho [3] apresenta a utilização do precondicionador multiescala ($\mathbf{M}_{ms}$) em conjunto com o um precondicionador ($\mathbf{M}_h$) no grid fino de forma multiplicativa. Essa combinação visa reduzir os erros de alta frequência através do $\mathbf{M}_h$ enquanto os erros de baixa frequência são eliminados pelo precondicionador multiescala. O acoplamento multiplicativo tem a desvantagem de precisar de uma multiplicação matriz vetor além da aplicação dos precondicionadores e, portanto, o número de iterações tem que ser reduzido o suficiente para compensar todas essas operações. Um alternativa é aplicação dos precondicionadores de forma aditiva, pois, nesse caso, não é necessária a multiplicação matriz vetor adicional. Outra vantagem do operador aditivo é que caso os $\mathbf{M}_{\mathrm{h}}^{-1}$ e $\mathbf{M}_{\mathrm{ms}}^{-1}$ sejam simétricos o precondicionador conjunto também é simétrico e pode ser utilizado juntamente com o Gradiente Conjugado.

As Tabelas 7.3, 7.4, 7.5 e 7.6 mostram a quantidade de iterações e o tempo utilizando utilizando o precondicionador aditivo e o multiplicativo com o solver Bicgstab. Nesses testes o precondicionador $\mathbf{M}_{\mathrm{h}}^{-1}$ foi o ILU(0). É importante perceber que para menores níveis de engrossamento a quantidade de iterações feitas pelo aditivo é bem maior chegando a 120% para o caso E. Porém, ao aumentar o nível de engrossamento, em geral a quantidade de iterações fica entre 10% e 20% maior. Isso deve ocorrer pelo fato da solução grossa estar mais distante da fina e o acoplamento aditivo ou multiplicativo começam a ficar com número de iterações semelhantes. Além disso, nos casos B e D os tempos de solução ótimos foram obtidos com o precondicionador aditivo, tornando-o uma alternativa competitiva ao multiplicativo com melhorias de em torno de 15%.

Tabela 7.3: Comparação de precondicionador aditivo contra multiplicativo para caso A utilizando como solver linear o método Bicgstab para diferentes níveis de engrossamento.

	Precondicionador					
	$\mathbf{M}_{\mathrm{ms}}^{-1} + \mathbf{M}_{\mathrm{h}}^{-1}(\mathbf{I} - \mathbf{K}^h\mathbf{M}_{\mathrm{ms}}^{-1})$		$\mathbf{M}_{\mathrm{h}}^{-1} + \mathbf{M}_{\mathrm{ms}}^{-1}$		Diferença	
Elemento Grosso	Iterações	Tempo (ms)	Iterações	Tempo (ms)	Iterações	Tempo
2x2	3	100.97	6	106.45	100.00%	5%
5x5	7	28.72	9	29.57	28.57%	3%
10x10	13	39.30	15	35.98	15.38%	-8%
20x20	23	64.99	26	55.99	13.04%	-14%

Tabela 7.4: Comparação de precondicionador aditivo contra multiplicativo para caso B utilizando como solver linear o método Bicgstab para diferentes níveis de engrossamento.

	Precondicionador				Diferença	
	$\mathbf{M}_{ms}^{-1} + \mathbf{M}_{h}^{-1}(\mathbf{I} - \mathbf{K}^{h}\mathbf{M}_{ms}^{-1})$		$\mathbf{M}_{h}^{-1} + \mathbf{M}_{ms}^{-1}$			
Elemento Grosso	Iterações	Tempo (ms)	Iterações	Tempo (ms)	Iterações	Tempo
2x2	4	2862.19	6	2793.66	50.00%	-2%
4x4	5	504.18	6	503.37	20.00%	0%
8x8	12	480.78	12	397.59	0.00%	-17%
16x16	21	710.77	24	627.88	14.29%	-12%
32x32	41	1476.73	47	1255.16	14.63%	-15%

Tabela 7.5: Comparação de precondicionador aditivo contra multiplicativo para caso D utilizando como solver linear o método Bicgstab para diferentes níveis de engrossamento.

	Precondicionador				Diferença	
	$\mathbf{M}_{ms}^{-1} + \mathbf{M}_{h}^{-1}(\mathbf{I} - \mathbf{K}^{h}\mathbf{M}_{ms}^{-1})$		$\mathbf{M}_{h}^{-1} + \mathbf{M}_{ms}^{-1}$			
Elemento Grosso	Iterações	Tempo (ms)	Iterações	Tempo (ms)	Iterações	Tempo
2x2	4	742.54	6	787.70	50.00%	6%
4x4	6	179.77	8	187.48	33.33%	4%
8x8	9	134.78	11	130.19	22.22%	-3%
16x16	10	136.04	11	113.89	10.00%	-16%
32x32	16	204.28	16	154.87	0.00%	-24%

Tabela 7.6: Comparação de precondicionador aditivo contra multiplicativo para caso E utilizando como solver linear o método Bicgstab para diferentes níveis de engrossamento.

	Precondicionador				Diferença	
	$\mathbf{M}_{ms}^{-1} + \mathbf{M}_{h}^{-1}(\mathbf{I} - \mathbf{K}^{h}\mathbf{M}_{ms}^{-1})$		$\mathbf{M}_{h}^{-1} + \mathbf{M}_{ms}^{-1}$			
Elemento Grosso	Iterações	Tempo (ms)	Iterações	Tempo (ms)	Iterações	Tempo
4x4	10	1766.14	22	2477.61	120.00%	40%
8x8	31	2250.98	39	2237.97	25.81%	-1%
16x16	41	2677.35	50	2511.19	21.95%	-6%
32x32	57	3795.57	65	3246.58	14.04%	-14%

7.3.3 Comparação Multiescala com Multigrid e ILU

Nessa seção, são apresentadas comparações entre os precondicionadores multiescala aditivo, multigrid e ILU. Para esse comparação foi utilizado o PyAmg, que é um solver multigrid descrito em OLSON e SCHRODER [15]. Dado a grande quantidade de parâmetros necessários para a configuração dos solver multigrid, como: a quantidade de níveis devem ser utilizados, quantidade de relaxações em cada nível, qual o tipo de relaxação será utilizada, dentre outras variáveis, foi utilizado o script **solver_diagnotics.py** disponibilizado pela equipe do PyAmg no repositório (https://github.com/pyamg/pyamg-examples). Esse script testa diferentes configurações de solver multigrid para uma dada matriz e seleciona aquele mais eficiente para o problema proposto. A configurações geradas pelo script estão no Anexo B. Para o caso B e caso E, o script não convergiu e, para o caso B foi utilizado a mesma configuração de solver que o caso A, por terem sido montados com as mesmas correlações. Para o caso E foi utilizada a mesma escolha que o caso D, por ser o único outro reservatório real. Abaixo seguem algumas características importantes das configurações encontradas:

- máximo de quinze níveis multigrid
- relaxação "Gauss Seidel Symmetric"
- ciclos W
- multigrid como precondicionador para o Gradiente Conjugado

A Figura 7.6 apresenta o tempo de solução do sistema utilizando o método multiescala e multigrid (PyAmg) como precondicionador para o gradiente conjugado para o caso B. São apresentados o resíduo ao longo das iterações, o tempo de execução do solver, a quantidade de iterações do solver linear e o tempo da iteração.

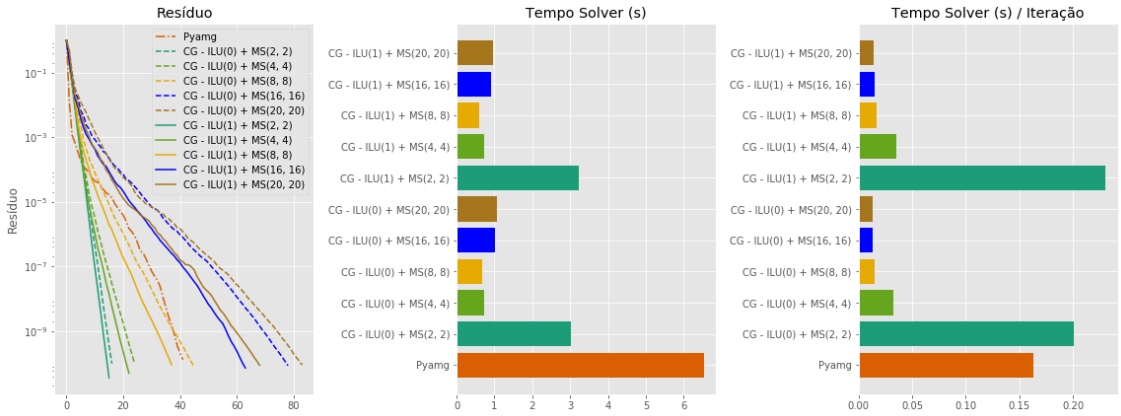


Figura 7.6: Resultados para caso B. Histórico do resíduo relativo ao longo das iterações, tempo do solver em segundos, número de iterações e tempo do solver por iteração.

Um primeiro ponto a se observar é o aumento do número de iterações a cada vez que se aumenta o fator de engrossamento da malha. Isso ocorre, pois a solução do problema grosso se torna cada vez mais distante da solução da malha fina fazendo com que o precondicionador funcione pior. Entretanto, quanto mais grossa a malha, mais fácil a solução sistema linear grosso e então existe uma solução de compromisso entre o engrossamento da malha e o tempo de execução. É importante lembrar também que sempre é necessário pagar o custo da multiplicação pelo operador de prolongamento e de restrição que é da ordem do grid fino. Uma constatação desse fato aparece na Figura 7.7 que mostra a proporção do tempo gasto do precondicionador aditivo entre a aplicação de um ILU(1) e de um multiescala com diferentes níveis de engrossamento, pode-se perceber que quanto maior o engrossamento a fração de tempo utilizada pelo precondicionador multiescala fica menor, pois quanto maior o engrossamento mais desprezível fica o tempo de solução do grid grosso sendo o tempo do precondicionador multiescala dominado pelo prolongamento e restrição.

No caso B, a solução de menor tempo é quando o nível grosso é construído ao se montar elementos utilizando 8x8 elementos finos. Ainda sobre o número de iterações, quando utilizado um elemento grosso de 2x2 o número de iterações é o menor encontrado, porém o custo de solução do sistema grosso impossibilita a utilização desse nível de engrossamento. Uma maneira de aproveitar essa redução do número de iterações seria utilizar um precondicionador multiescala multinível conforme apresentado em [22] para volumes finitos, desse modo, não seria necessário resolver o sistema linear na malha 2x2 sendo necessário apenas aplicar uma relaxação nesse nível similarmente ao multigrid.

Na Figura 7.8 é apresentado a comparação da solução do sistema utilizando o melhor método multiescala com o gradiente conjugado utilizando como precondicionador o ILU(0), ILU(1) e solver multigrid PyAmg. É importante notar que há uma

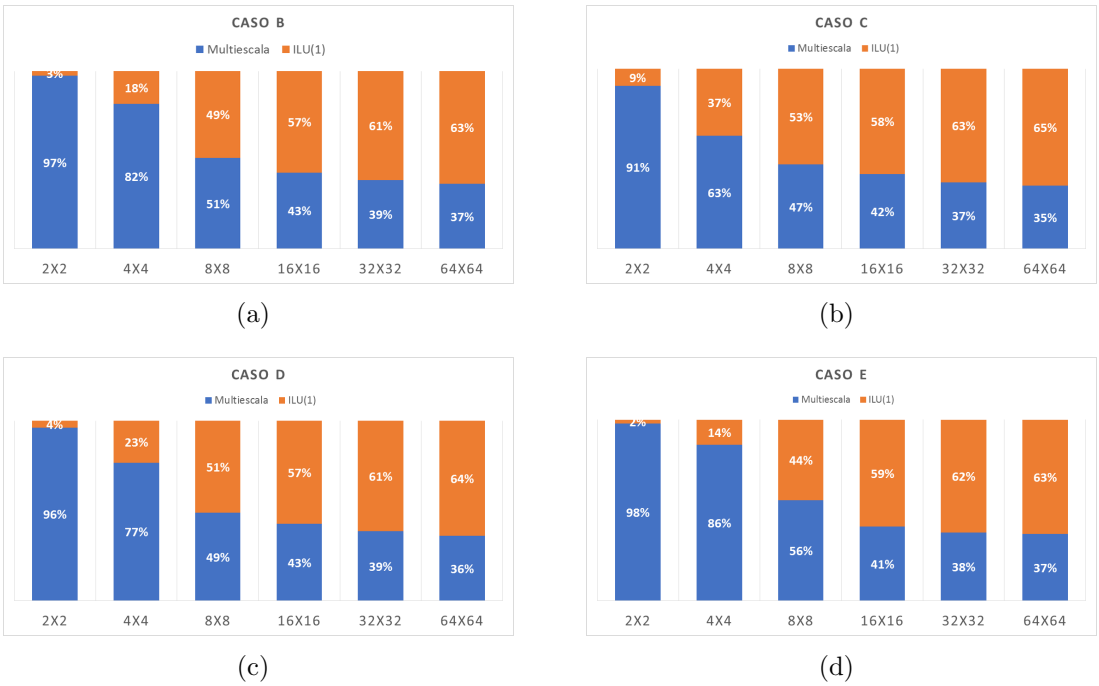


Figura 7.7: Proporção do tempo gasto entre ILU(1) e multiescala em precondicionador aditivo para o casos B, C, D e E.

redução expressiva no número de iterações do PyAmg, e multiescala em relação aos ILU (redução em torno de 83%) nos dois casos. Essa redução vem com o preço de um precondicionador com um custo computacional mais caro. Já uma comparação de tempo mostra uma redução de 72% com MS(8,8)+ILU(1) em relação a execução com o precondicionador ILU(1). A comparação com o tempo do PyAmg pode não condizer com a realidade por se tratarem de implementações diferentes, posteriormente serão mostradas outros tipos de comparação com o PyAmg.

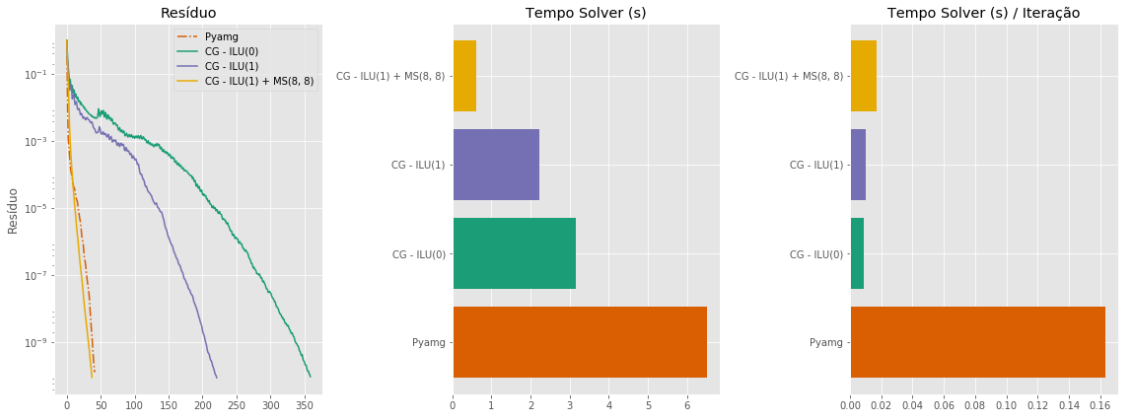


Figura 7.8: Resultados para caso B. Histórico do resíduo relativo ao longo das iterações, tempo do solver em segundos e tempo do solver por iteração.

A seguir, as Figuras 7.9(a), 7.9(b) e 7.9(c) apresentam os resultados para os casos C, D e E. Em todos os gráficos é mostrado apenas o precondicionador multiescala

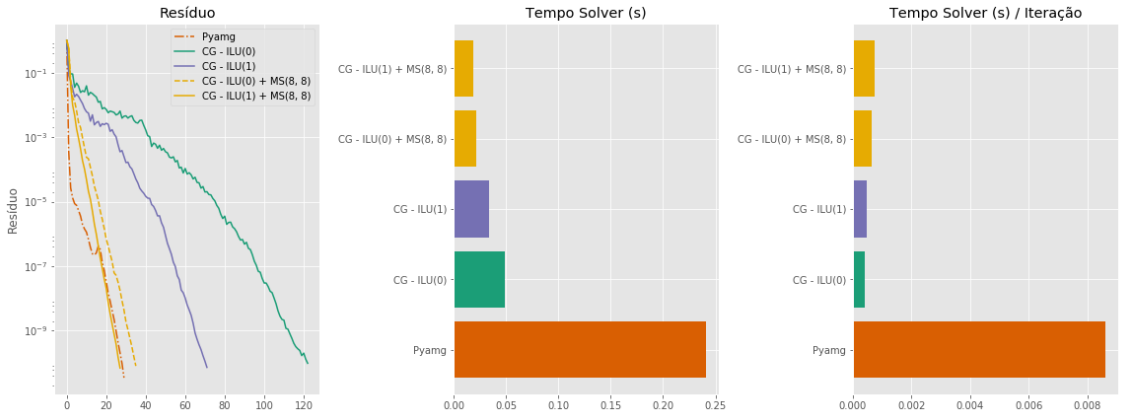
que obteve o melhor desempenho entre os fatores de engrossamento de 2x2, 4x4, 8x8, 16x16, 32x32, junto com os resultados do ILU(0), ILU(1) e PyAmg.

Nos casos C e D novamente a quantidade de iterações do PyAmg e multiescala foram semelhantes, mostrando um bom desempenho dos dois métodos, porém, no caso E o precondicionador multigrid começa decrescendo rapidamente mas passa por um ponto de resíduo 10^{-6} que a inclinação da queda muda rapidamente fazendo com que o solver multigrid tenha um número de iterações próxima ao do ILU(1).

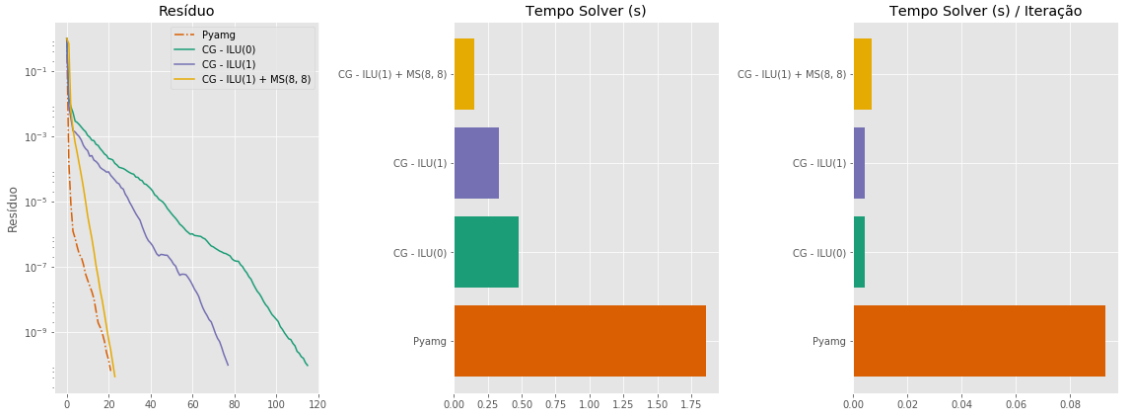
A Tabela 7.7 apresenta a quantidade de iterações do solver, tempo e speed-up de cada caso do precondicionador multiescala em relação ao ILU(1). Pode-se notar que os maiores speed-ups ocorreram nos casos com maiores dimensões (caso B e caso E) pois foram esses que obtiveram, proporcionalmente, as maiores reduções no número de iterações.

Em relação ao PyAmg, a Tabela 7.8 apresenta a quantidade de não zeros somando todos os níveis dos operadores de prolongamento do multigrid, o número de não zeros de todas as operações de prolongamento de um ciclo W e o número de não zeros do operador de prolongamento do multiescala. Nesse caso, pode-se constatar que a quantidade de não zeros dos operador de prolongamento do multiescala tem em torno de 2 a 2,2 vezes a quantidade de não zeros das multiplicações matriz vetor de um ciclo W do método multigrid sendo então uma vantagem do método multigrid.

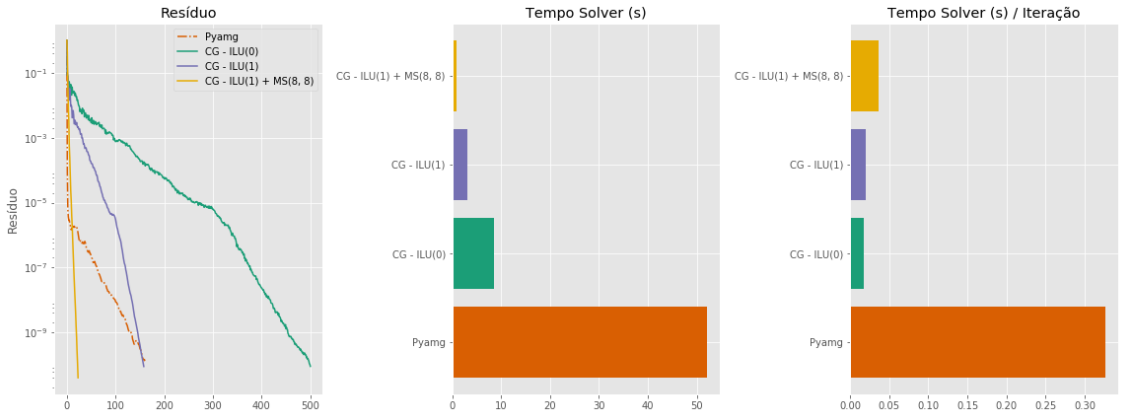
Por outro lado, a Tabela 7.9 apresenta a complexidade dos operadores multigrid gerados para cada um dos casos. Além disso, possui a complexidade do ciclo W que considera as relaxações feitas em cada nível. Portanto, em relação as relaxações o multiescala necessita de bem menos capacidade computacional que fica em torno de 4,8 e 5,7 vezes menor que no multigrid que o torna mais eficiente nesse quesito. Assim, em relação ao caso E fica clara a vantagem do multiescala dado que o número de iterações realizadas foram 23 enquanto no multiescala foram 160 fazendo com o problema do prolongamento seja superado por ter feito menos que duas vezes a quantidade de iterações. Para os outros casos, o multiescala se saiu melhor dado que, apesar do prolongamento serem mais caras (entre 1.99 e 2.18), as operações de relaxação são bem mais custosas no multigrid (entre 4,8 e 5,18 vezes mais custosas). Um outro ponto que pesa a favor do multiescala é que a multiplicação matriz vetor é mais facilmente paralelizável que as relaxações. Por exemplo, para paralelização dos Gauss-Seidel e ILU é necessário reordenamento de equações através de colorações que podem afetar a convergência do método.



(a) Caso C



(b) Caso D



(c) Caso E

Figura 7.9: Histórico do resíduo, número de iterações e tempo do solver por iteração para caso C, D , E. O tempo é a media entre 10 rodadas.

Tabela 7.7: Tabela com comparação entre gradiente conjugado com precondicionador ILU(1) e precondicionador aditivo multiescala

	CG - ILU(1)		**CG - ILU(1) + MS(8, 8)**		
Caso	**Iterações**	**Tempo(s)**	**Iterações**	**Tempo**	**SpeedUp**
Caso B	221	2.224371	37	0.608054	3.66
Caso C	71	0.034168	27	0.018995	1.80
Caso D	77	0.334961	23	0.148704	2.25
Caso E	158	3.082180	23	0.803692	3.84

Tabela 7.8: Tabela com comparação entre número de não zeros das multiplicações pelos prolongamentos de cada ciclo. Valores normalizados pela quantidade de não zeros da matriz.

Caso	**NNZ Mg**	**NNZ W**	**NNZ Ms**	**NNZ Ms/NNZ W**
Caso B	0.15	0.24	0.49	2.05
Caso C	0.14	0.21	0.45	2.18
Caso D	0.14	0.24	0.48	1.99
Caso E	0.15	0.24	0.49	2.02

Tabela 7.9: Tabela entre complexidade dos operadores multigrid e multiescala.

Caso	**Complex. do Grid**	**Complex. do Ciclo W**	**Complex. Multiescala**
Caso B	1.53	5.18	1.041
Caso C	1.57	4.83	1.033
Caso D	1.63	6.53	1.040
Caso E	1.55	5.65	1.041

7.3.4 Acurácia da solução grossa

Nos resultados mostrados na seção anterior, era utilizado um solver direto para resolver o sistema grosso. Em casos que a malha foi suficientemente reduzida esse tempo de solução é desprezível, porém, para casos o fator de engrossamento é pequeno resolver exatamente o nível grosso aumenta o tempo de execução (por exemplo, o caso ILU(1) + MS(2, 2) na Figura 7.6). Além disso, modelos muito refinados podem ter espaço grosseiro grande o suficiente para necessitarem de métodos iterativos, como, por exemplo, para soluções de casos com centenas de milhões de elementos como os modelos mostrados em [30].

A Figura 7.10 apresenta a quantidade de iterações realizadas pelo gradiente conjugado com tolerância 10^{-10} no eixo y e a tolerância do solver do grid grosso no eixo x. Para solução do grid grosso foi utilizado um gradiente conjugado precondicionador ILU(0). Pode-se constatar um aumento do número de iterações com o aumento da tolerância do solver grosso que era esperado. Mesmo com esse aumento o tempo de convergência total melhora para o caso 2x2.

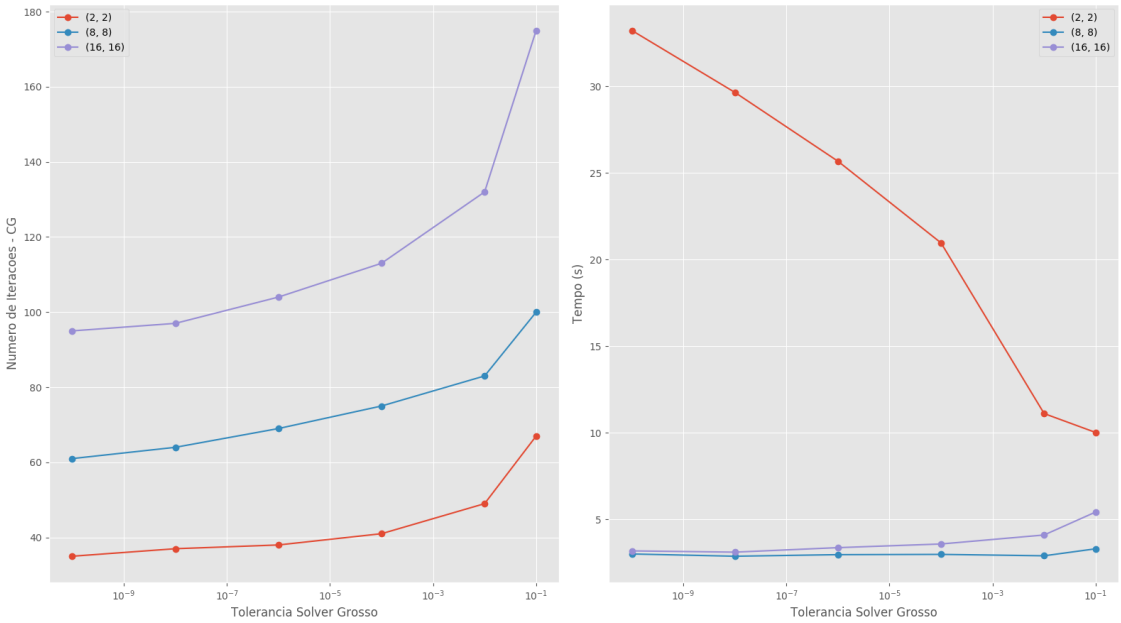


Figura 7.10: Variação do número de iterações do gradiente conjugado com tolerância do grid grosso para Caso E.

Capítulo 8

Conclusão e Trabalhos Futuros

Essa dissertação apresentou a teoria necessária para a construção de um simulador de geomecânica elástico utilizando o método dos elementos finitos e método multiescala como precondicionador para sistemas lineares. Como resultados foram apresentadas comparações entre a abordagem de combinar precondicionadores de forma aditiva e multiplicativa que mostram que, para fatores de engrossamento grandes o suficientes, pode-se obter soluções em torno de 10% a 15% mais rápidas utilizando o precondicionador aditivo. Além disso, foram apresentadas comparações utilizando o método do gradiente conjugado com precondicionador multiescala, ILU e multigrid para modelos sintéticos e também para casos reais. A comparação entre o multiescala e o ILU mostrou que o primeiro reduz o número de iterações e encontra soluções mais rapidamente que o ILU mostrando speed-ups de até 3,8 vezes. A comparação entre o multigrid e multiescala mostrou que apenas no caso E o multigrid realizou mais iterações enquanto que nos outros a quantidade de iterações foi similar. Porém, com relação a métrica do número de operações realizadas, as relaxações dos vários níveis do multigrid tem um custo bem maior por iteração, tornando o multiescala uma opção mais atrativa. Outro ponto importante é que os resultados mostraram também a eficácia do método para a simulação de modelos com dados reais de campos que possuem propriedades heterogêneas e grids mais complexos.

Visto o bom resultado do método para os sistemas lineares geomecânicos, uma implementação paralela e algébrica tem grandes possibilidades de reduzir o tempo de simulação em modelos em três dimensões. Além disso, como o operador multiescala também possui uma dimensão bem menor que a do operador do grid original, ele pode ser pensando com um precondicionador grosseiro geral de todo o modelo em uma tentativa de resolver o problema do aumento do número de iterações dos solvers quanto mais o domínio é dividido. Com essas ideias pode-se melhorar ainda mais os resultados obtidos em FIGUEIREDO *et al.* [30] tornando as execuções do simulador geomecânico da Petrobras ainda mais rápidas. Essas novas implementações tem diversos desafios em relação ao paralelismo como a coordenação da solução dos

problemas locais em paralelo, implementação de produtos matriz-matriz esparsas para cálculo do operador grosso, comunicação dos resultados de problemas locais nas fronteiras do processos, dentre outros.

Referências Bibliográficas

[1] VERRUIJT, A. *Computational Geomechanics*. 2 ed. , John Wiley & Sons Ltd., 1988. ISBN: 0-471-91552-1.

[2] ZOBACK, M. *Reservoir Geomechanics*. 1 ed. , Cambridge University Press, 2010. ISBN: 978-0-521-14619-7.

[3] N. CASTELLETTO, H. HAJIBEYGIB, H. T. "Multiscale finite-element method for linear elastic geomechanics", *Journal of Computational Physics*, v. 331, pp. 337–356, 2017.

[4] HUGHES, T. J. R. *The Finite Element Method.* 1 ed. , Dover Publications Inc., 2000. ISBN: 978-0-486-41181-1.

[5] FISH, J., BELYTSCHKO, T. *A First Course in Finite Elements*. 1 ed. , John Wiley & Sons Ltd., 2007. ISBN: 978-0-470-03580-1.

[6] GAI, X. *A Coupled Geomechanics and Reservoir Flow Model on Parallel Computers*. Tese de Doutorado, The University of Texas at Austin, 2004.

[7] CUI, L., KALIAKIN, V. N., ABOULEIMAN, Y., etal. "Finite Element Formulation and Application of Poroelastic Generalized Plane Strain Problems", *International Journal of Rock Mechanics and Mining Sciences*, v. 34, pp. 953–962, 1997.

[8] SOKOLOVA K., BASTISYA M. G., H. H. "Multiscale finite volume method for finite-volume-based simulation of poroelasticity", *Journal of Computational Physics*, v. 331, pp. 337–356, 2017.

[9] LEWIS, R. W., SCHREFLER, B. A. *The Finite Element Method in th Static and Dynamic Deformation and Consolidation od Porous Media*. 2 ed. , John Wiley & Sons Ltd., 2000. ISBN: 0-471-92809-7.

[10] SAAD, Y. *Iterative Methods for Sparse Linear Systems*. 2 ed. , Society for Industrial and Applied Mathematics, 2003. ISBN: 0898715342.

[11] MEIJERINK, J. A., VAN DER VORST, H. A. "An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix", *Math. Comp.*, v. 31, pp. 148–162, 1977. doi: http://dx.doi.org/10.1016/j.advwatres.2011.04.013.

[12] MANEA, A. M., SEWALL, J., TCHELEPI, H. A. "Parallel Multiscale Linear Solver for Highly Detailed Reservoir Models", *SPE Journal*, v. 21, pp. 2062–2078, 2016.

[13] SHEWCHUK, J. R. *An introduction to the conjugate gradient method without the agonizing pain*. Relatório técnico, 1994.

[14] BRIGGS, W. L., HENSON, V. E., MCCORMICK, S. F. *A Multigrid Tutorial*. 2 ed. Philadelphia, SIAM, 2000. ISBN: 0-89871-462-1.

[15] OLSON, L. N., SCHRODER, J. B. "PyAMG: Algebraic Multigrid Solvers in Python v4.0". 2018. Disponível em: <`https://github.com/pyamg/pyamg`>. Release 4.0.

[16] HEATH, M. *Scientific computing : an introductory survey*. New York, McGraw-Hill, 1997. ISBN: 0070276846.

[17] THOMAS Y. HOU, X.-H. W. "A Multiscale Finite Element Method for Elliptic Problems in Composite Materials and Porous Media", *Journal of Computational Physics*, v. 134, pp. 169–189, 1997.

[18] CHEN, Z., HOU, T. Y. "A Mixed Multiscale Finite Element Method for elliptic problems with Oscillating Coefficients", *Mathematics Of Computation*, v. 72, pp. 541–576, 2002.

[19] JENNY, P., TCHELEPI, H. A. "Multi-scale finite-volume method for elliptic problems in subsurface flow simulation", *Journal of Computational Pyshics*, v. 187, pp. 47–67, 2003.

[20] HAJIBEYGI, H., BONFIGLI, G., HESSE, M. A., etal. "Iterative Multiscale finite-volume method", *Journal of Computational Physics*, v. 227, pp. 8604–8621, 2008.

[21] ZHOU, H., TCHELEPI, H. A. "Operator-Based Multiscale Method for Compressible Flow", *SPE Journal*, v. 13, pp. 267–273, 2008.

[22] MANEA, A., ALMANI, T. "A Multi-Level Algebraic Multiscale Solver (ML-AMS) For Reservoir Simulation". In: *Proc. 16th European Conference on the Mathematixs of Oil Recovery*, v. 16. EAGE, 2018.

[23] CASTELLETTO, N., KLEVTSOV, S., HAJIBEYGI, H., etal. "Multiscale two-stage solver for Biot's poroelasticity equations in subsurface media", *Computational Geosciences*, 2018. Disponível em: <https://doi.org/10.1007/s10596-018-9791-z>.

[24] MARCO BUCK, OLEG ILLIEV, H. A. "Multiscale finite element coarse spaces for the application to linear elasticity", *Central European Journal of Mathematics*, v. 11, pp. 680–701, 2013.

[25] ZHOU, H., TCHELEPI, H. A. "Two-Stage Algebraic Multiscale Linear Solver for Highly Heterogeneous Reservoir Models", *SPE Journal*, v. 17, pp. 2062–2078, 2012.

[26] BALAY, S., ABHYANKAR, S., ADAMS, M. F., etal. *PETSc Users Manual*. Relatório Técnico ANL-95/11 - Revision 3.8, Argonne National Laboratory, 2017.

[27] BALAY, S., GROPP, W. D., MCINNES, L. C., etal. "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries". In: Arge, E., Bruaset, A. M., Langtangen, H. P. (Eds.), *Modern Software Tools in Scientific Computing*, pp. 163–202. Birkhäuser Press, 1997.

[28] DALCIN, L. D., PAZ, R. R., KLER, P. A., etal. "Parallel distributed computing using Python", *Advances in Water Resources*, v. 34, n. 9, pp. 1124–1139, 2011. doi: http://dx.doi.org/10.1016/j.advwatres.2011.04.013. New Computational Methods and Software Tools.

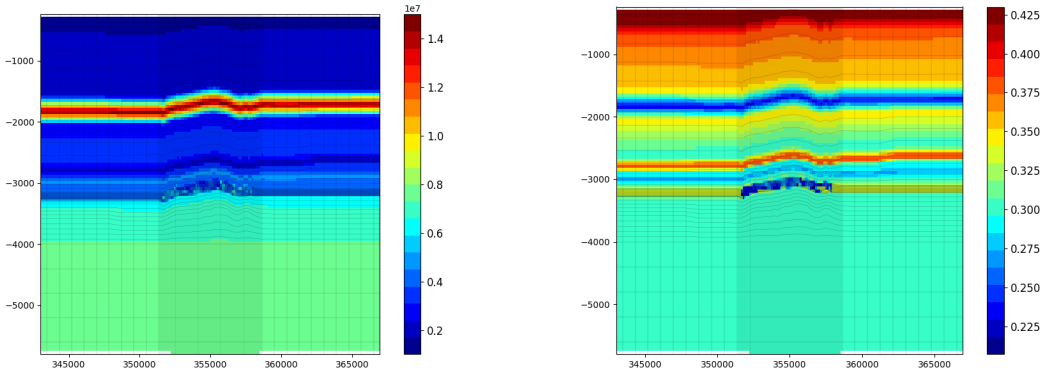
[29] BAU, D., FERRONATO, M., GAMBOLATI, G., etal. "Basin-scale compressibility of the northern Adriatic by the radioactive marker technique", *Géotechnique*, v. 52, pp. 605–616, 2002.

[30] FIGUEIREDO, M. O., RODRIGUES, J. R. P., ET∼AL. "Paralelização de Simulador de Geomecânica Utilizando MPI". In: *Rio Oil and Gas 2018*, v. 19. IBP, 2018.

Apêndice A

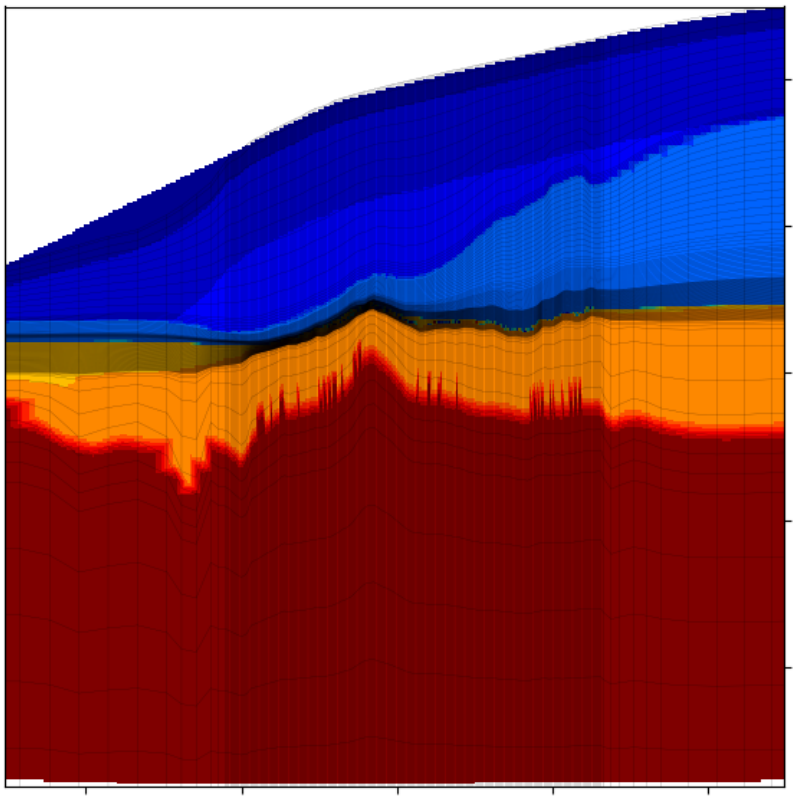
Figuras dos reservatórios

Aqui estão apresentadas os grids dos casos C, D e E que foram utilizados para a exposição dos resultados. As escalas dos casos D e E foram omitidas por se tratarem de modelos reais e os dados, portanto, são sensíveis para a Petrobras.

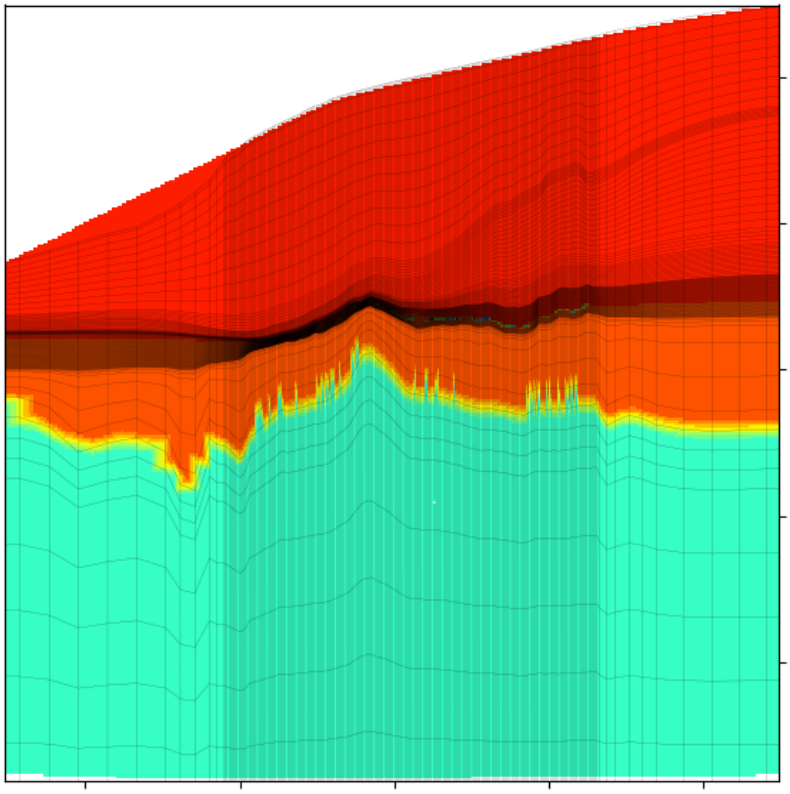


(a) (b)

Figura A.1: Propriedades módulo de Young (a) e coeficiente de poisson (b) para caso C

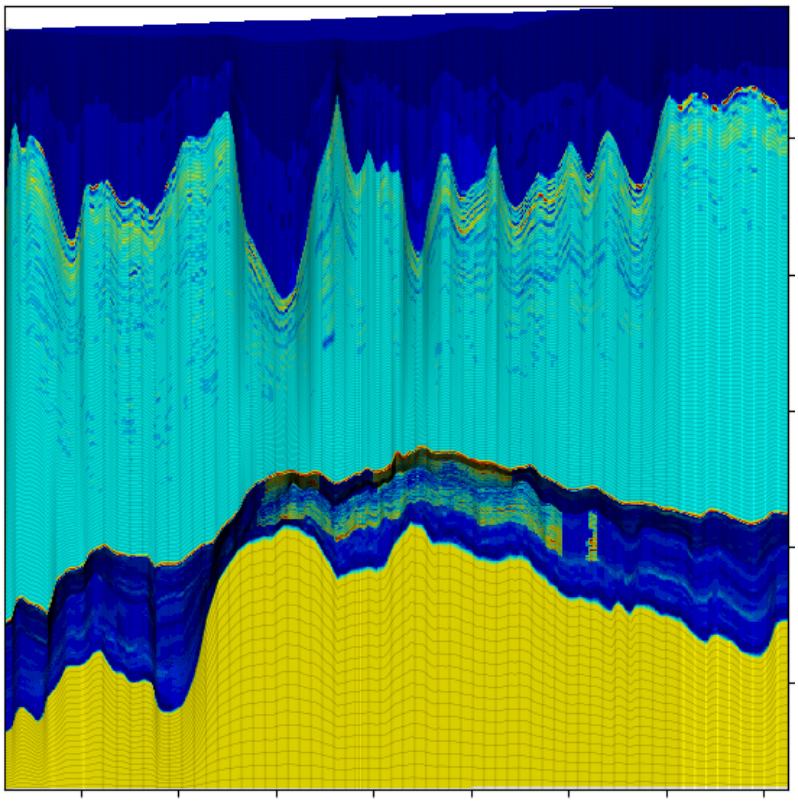


(a)

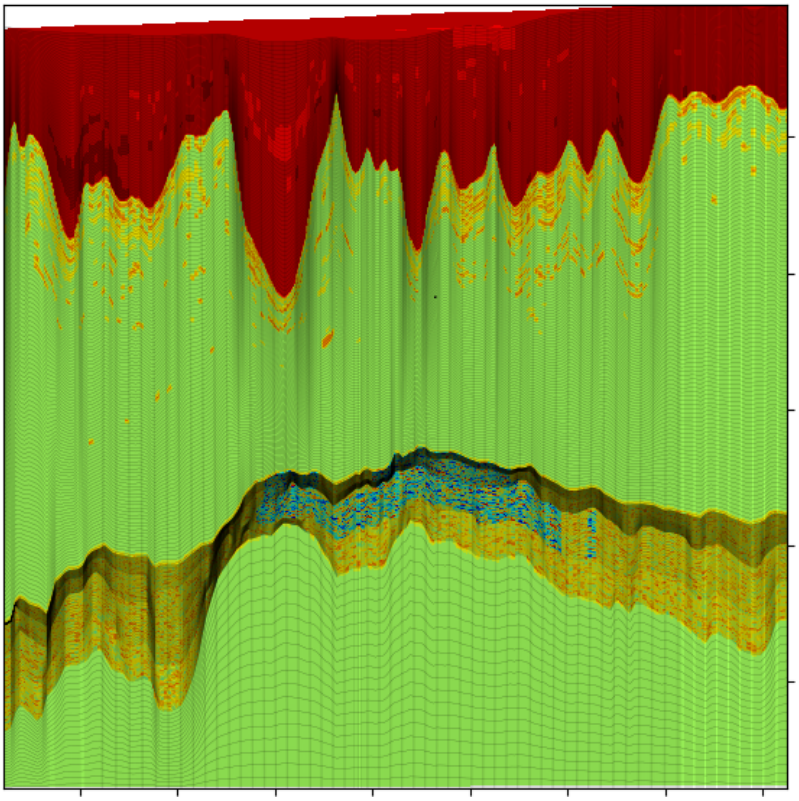


(b)

Figura A.2: Propriedades módulo de Young (a) e coeficiente de poisson (b) para caso D

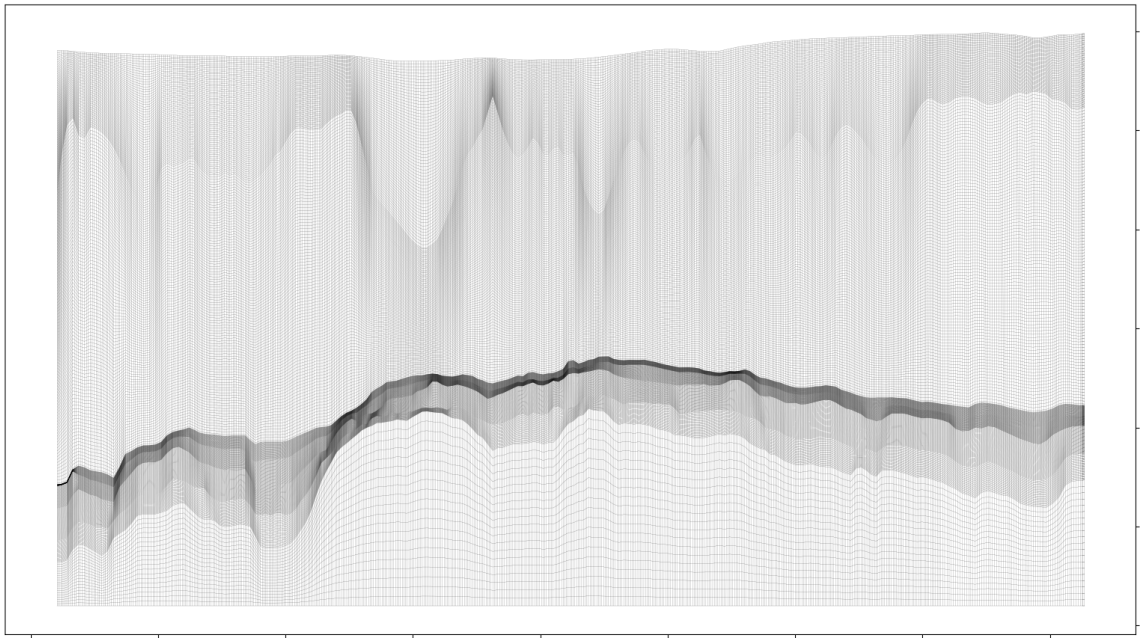


(a)

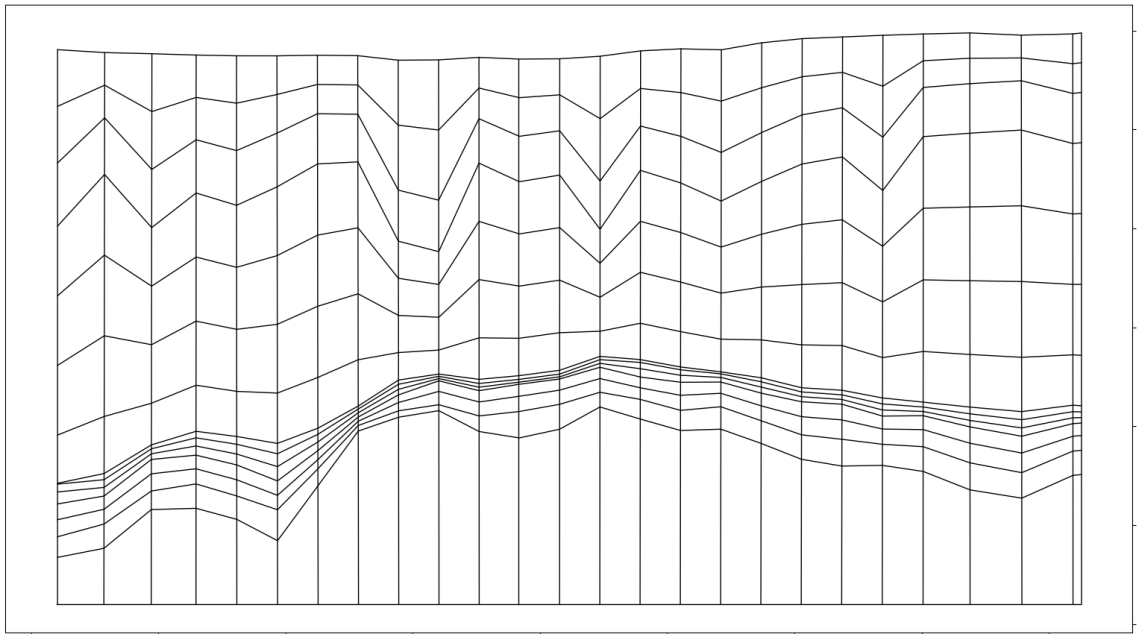


(b)

Figura A.3: Propriedades módulo de Young (a) e coeficiente de poisson (b) para caso E



(a)



(b)

Figura A.4: Grid de simulação original (a) e após refinamento 24 x 24(b) para caso E

Apêndice B

Configurações de solver utilizadas pelo PyAMG

Seguem abaixo as chamadas com os parâmetros para construção dos solvers multigrid utilizados pelo PyAMG para os resultados apresentados no Capítulo 7. Esses foram retornados pelo script **solver_diagnostics.py**. Em todos os casos a chamada do solver multigrid foi sugerida com o ciclo W e como pré-condicionador para o Gradiente Conjugado.

Caso A e Caso B

```
smoothed_aggregation_solver(A, B=B, BH=BH,
  strength=('evolution', {'k': 2, 'proj_type': 'l2', 'epsilon': 2.0}),
  smooth=('energy', {'krylov': 'cg', 'maxiter': 2,
          'degree': 1, 'weighting': 'local'}),
  improve_candidates=[('block_gauss_seidel',
                       {'sweep': 'symmetric', 'iterations': 4}),
                       None, None, None, None, None, None,
                       None, None, None, None, None,
                       None, None, None],
  aggregate="standard",
  presmoother=('block_gauss_seidel',
               {'sweep': 'symmetric', 'iterations': 1}),
  postsmoother=('block_gauss_seidel',
                {'sweep': 'symmetric', 'iterations': 1}),
  max_levels=15,
  max_coarse=300,
  coarse_solver="pinv")
```

Caso C

```
smoothed_aggregation_solver(A, B=B, BH=BH,
  strength=('evolution', {'k': 2, 'proj_type': 'l2', 'epsilon': 2.0}),
  smooth=('energy', {'krylov': 'cg', 'maxiter': 2,
                     'degree': 1, 'weighting': 'local'}),
  improve_candidates=[('block_gauss_seidel',
                       {'sweep': 'symmetric', 'iterations': 4}),
                       None, None, None, None, None, None, None,
                       None, None, None, None, None, None, None],
  aggregate="standard",
  presmoother=('block_gauss_seidel',
               {'sweep': 'symmetric', 'iterations': 1}),
  postsmoother=('block_gauss_seidel',
                {'sweep': 'symmetric', 'iterations': 1}),
  max_levels=15,
  max_coarse=300,
  coarse_solver="pinv")
```

Caso D e Caso E

```
smoothed_aggregation_solver(A, B=B, BH=BH,
  strength=('evolution', {'k': 2, 'proj_type': 'l2', 'epsilon': 2.0}),
  smooth=('energy', {'krylov': 'cg', 'maxiter': 2,
                     'degree': 1, 'weighting': 'local'}),
  improve_candidates=None,
  aggregate="standard",
  presmoother=('block_gauss_seidel',
               {'sweep': 'symmetric', 'iterations': 1}),
  postsmoother=('block_gauss_seidel',
                {'sweep': 'symmetric', 'iterations': 1}),
  max_levels=15,
  max_coarse=300,
  coarse_solver="pinv")
```