HUGO VERIDIANO GONÇALVES

PÓS-PROCESSAMENTO DE CÓDIGO DE ELEMENTOS FINITOS NA ANÁLISE DE COLUNAS DE PERFURAÇÃO

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do diploma de Engenharia de Petróleo.

SANTOS

2020

HUGO VERIDIANO GONÇALVES

PÓS-PROCESSAMENTO DE CÓDIGO DE ELEMENTOS FINITOS NA ANÁLISE DE COLUNAS DE PERFURAÇÃO

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do diploma de Engenharia de Petróleo.

Área de concentração: Perfuração

Orientador: Ronaldo Carrion

SANTOS

2020

FICHA CATALOGRÁFICA

Gonçalves, Hugo Veridiano

PÓS-PROCESSAMENTO DE CÓDIGO DE ELEMENTOS FINITOS NA ANÁLIS DE COLUNAS DE PERFURAÇÃO / H.V. Gonçalves – São Paulo, 2020.
43 p.

Trabalho de Formatura – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Minas e de Petróleo.

1.Python 2.Método dos Elementos Finitos 3.Pós-processamento.
I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Minas e de Petróleo II.t.

AGRADECIMENTOS

Gostaria de agradecer a Deus pelas oportunidades e pelos sutis toques no meu caminho.

Agradecer ao meu pai, Ttel, que me vigia e me guarda de um lugar melhor.

À minha mãe Fernanda, por fazer o possível e impossível para garantir minha educação e por criar filhos sadios emocionalmente.

À minha vó Regina, por sempre incentivar meus estudos e discutir conceitos comigo.

À minha tia Renata e meu irmão Pedro, que deram suporte para que eu conseguisse sempre atender às minhas aulas.

À minha namorada Lara, por esses anos de incrível companheirismo e suporte. Obrigado por ter passado os domingos estudando ao meu lado e por compreender e me ajudar com as minhas angústias.

Ao meu orientador Prof. Ronaldo Carrion pela grande parceria neste ano e por ter agarrado este projeto comigo.

Aos meus colegas do curso de Engenharia de Petróleo da Escola Politécnica da USP, muito obrigado por todas as experiências e aprendizados que me proporcionaram.

A todos os docentes e funcionários, muito obrigado.

Ao futuro Hugo, que possa encontrar boas lembranças da época da faculdade ao ler este trabalho novamente. Nós conseguimos.

“Aqueles que são loucos o suficiente

para achar que podem mudar o mundo,

são os que realmente o fazem.”

(Steve Jobs)

RESUMO

Com o avanço da tecnologia na área de Engenharia de Petróleo, novos métodos vão surgindo para ajudar na realização das rotinas executadas. *Softwares* computacionais são grande parte desse avanço, pois auxiliam os profissionais tanto em atividades simples do cotidiano quanto em projetos que demandam mais tempo e atenção. No entanto, boa parte desses *softwares* são pagos e possuem pouca abertura para que o usuário faça adaptações. Pensando nisso, este trabalho tem como objetivo produzir um código computacional na linguagem *Python* que realize o pós-processamento automatizado e otimizado do cálculo de tensões e deslocamentos obtidos pelo método dos elementos finitos em colunas verticais, fornecendo gráficos que auxiliem na compreensão dos resultados, utilizando um código autoral. Foram obtidos gráficos para diferentes testes realizados que puderam representar os resultados de forma clara e facilitar a compreensão ao realizar a leitura.

Palavras-chave: Python, Método dos elementos finitos, Pós-processamento.

ABSTRACT

With the advancement of technology in the area of Petroleum Engineering, new methods are emerging to give help in carrying out the routines performed. Computational softwares are a big part of this advance, as it helps professionals both in simple daily activities and in projects that demand more time and attention. However, most of those softwares are paid for and have small openings for the user to make adaptations. With this in mind, this work aims to produce a computational code in Python that performs the automated and optimized post-processing of stress and displacement calculations of vertical columns, obtained by the Finite element method, providing graphics that assist in the understanding of the results, using a original code. Graphs for different performed tests were obtained, representing the result clearly and making it easier to understand.

Keywords: Python, Finite element method, Post-processing.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da tecnologia na área de Engenharia de Petróleo, novos métodos vão surgindo para ajudar na realização das rotinas executadas por profissionais da área. Com o surgimento dos computadores e dos *softwares* computacionais, alguns problemas, que antes demandavam muito tempo e esforço humano para serem calculados, foram sendo ressignificados.

Ano após ano surgem renovações na parte tecnológica da indústria, que altera sua dinâmica operacional para se alinhar às novas tecnologias, que acabam por trazer um resultado positivo financeiramente. Os *softwares* computacionais são grande parte dessa revolução tecnológica, pois auxiliam os profissionais tanto em atividades simples do cotidiano quanto em complexos projetos demorados.

No entanto, boa parte destes *softwares* tem licenças pagas e são desenvolvidos por grandes empresas para serem vendidos para outras grandes empresas com claros objetivos comerciais. Alguns destes *softwares* possuem licenças estudantis para universidades, mas a dinâmica de utilização dos programas acaba fazendo com que os estudantes se familiarizem com o uso dos mesmos para se adequarem a utilizá-los no mercado de trabalho, sem que entendam como os programas realizam suas funções.

Boa parte dos *softwares* comerciais possui um código fechado, de modo que o usuário não tenha acesso à implementação do mesmo. Isso acaba por gerar dificuldades na compreensão do funcionamento do programa e de como ele realiza seus cálculos, além de dificultar que o usuário personalize o programa e adicione extensões que sejam úteis para ele mesmo, vedando novamente o aprendizado do aluno.

No entanto, existem *softwares* e plataformas de programação de “código aberto”, ou seja, que são manipuláveis pelo usuário, de forma que o usuário possa alterar o programa ou utilizar dessas linguagens para escrever o código de um programa próprio, o qual realizará as funções e os cálculos que deseja.

Premissas básicas de liberdade de expressão, acesso à informação e coletividade do conhecimento, que deve ser disponibilizado de forma democrática, são princípios do código aberto (STEFANUTO et al., 2005).

Uma das principais linguagens de programação de código aberto é o *Python*. Uma linguagem de grande potencial e muito usada academicamente pela fácil leitura e compreensão da sintaxe, além de oferecer suporte para bibliotecas externas que podem ser utilizadas para simplificar o código do usuário.

Entre essas bibliotecas que podem ser implementadas, existem várias que poderiam auxiliar neste trabalho para simplificar o código do programa a ser desenvolvido, entre elas destacam-se Fipy, FEniCS e Sfepy, todas escritas em Python.

No entanto, o uso dessas bibliotecas com foco exclusivo nos tópicos alvo a serem abordados, desvia da filosofia proposta deste trabalho, que é desenvolver um código autoral.

Em muitos programas não-comerciais desenvolvidos para o cálculo de equações diferenciais, principalmente o de resolução pelo método dos elementos finitos, um dos grandes entraves é a representação dos dados após serem processados.

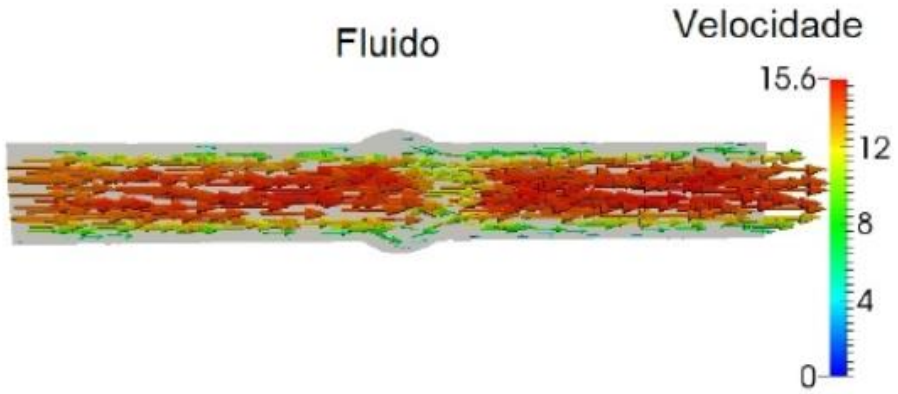
Os programas geralmente “retornam” os dados de forma numérica, dificultando a compreensão e a visualização da resolução do problema. Assim, utilizar programas que realizem o “pós-processamento” destes cálculos é importante para que haja compreensão visual dos cálculos realizados.

O pós-processamento é a etapa em que é realizado o tratamento dos dados obtidos de forma a representar as grandezas envolvidas no problema com clareza e objetividade (PENNA, 2007), auxiliando na compreensão dos resultados obtidos e na interpretação dos mesmos.

Com o pós-processamento, os dados numéricos se tornam representações bidimensionais ou até tridimensionais dos resultados, de maneira a facilitar a compreensão. Como exemplo dessas representações, pode-se mencionar os gráficos de calor (*heatmaps*), que utilizam a cor dos elementos como uma grandeza.

Um exemplo de pós-processamento é o da Figura 1 abaixo, realizado por um programa que utiliza a biblioteca FEniCS.

Figura 1 - Exemplo de representação utilizando a biblioteca FEniCS.



Fonte: (ABREU; CATABRIGA, 2017).

Entende-se que o cálculo das tensões em colunas verticais é essencial para um projeto de perfuração, tendo em vista que as condições em que a utilização do equipamento ocorre podem ser perigosas para as pessoas que trabalham no projeto. Geralmente esses equipamentos são testados em laboratório em condições desejadas de carregamento (VAISBERG et al, 2002).

Assim, pretende-se realizar cálculos por meio de um programa na linguagem *Python* a fim de computar as tensões distribuídas em colunas verticais de petróleo, realizando os cálculos pelo método de elementos finitos além de proporcionar a visualização desses cálculos obtidos por meio de gráficos que auxiliem a compreensão do resultado.

1.1 Objetivo

O trabalho tem como objetivo desenvolver um programa na linguagem *Python* para realizar cálculos de tensões em colunas verticais pelo método dos elementos finitos e realizar o pós-processamento dos resultados destes cálculos gerando gráficos que possam facilitar a compreensão da solução desenvolvida.

Os objetivos específicos desse trabalho são:

a) Apresentar a metodologia do método dos elementos finitos como forma de solução dos problemas propostos.
b) Produzir rotinas em Python para a produção do pós-processamento.
c) Apresentar os pós-processamentos obtidos em formas de gráficos claros produzidos pelo programa.

1.2 Justificativa

Atualmente na Engenharia a utilização de softwares para a realização das mais diversas tarefas é necessária. A simplificação de cálculos complexos permite mais eficiência em projetos e rapidez na entrega de informação. No entanto, boa parte dos *softwares* se apresenta com pouca “disposição” para que o usuário faça alterações de acordo com seu interesse. A elaboração de sistemas de software parece envolta em uma "ortodoxia técnica", e, assim, vista apenas como um processo “técnico”, a ser realizado por especialistas (CUKIERMAN; TEIXEIRA; PRIKLADNICKI, 2007).

Essa perspectiva desestimula que graduandos se proponham a elaborar seus próprios programas para a execução de diferentes tarefas, e assim, acabam se adaptando a utilização de *softwares* de código fechado.

A justificativa pessoal para a escolha do tema envolve a vontade de criar um programa “do zero”. Isso permite verificar todo o embasamento teórico presente nos cálculos que o programa realiza e assim ter completo compreendimento do funcionamento do programa.

Em rotinas de programação, o conhecimento pleno do funcionamento de um programa é essencial para poder reparar possíveis erros que são muito comuns nas fases iniciais da concepção do mesmo. Ao utilizar programas adicionais para desenvolver um *software* novo, a tarefa de rastrear os erros que aparecem no desenvolvimento do programa fica mais difícil, tornando o processo mais complicado e menos original.

Utilizando-se de códigos abertos, pode-se verificar a possibilidade do entendimento por trás da elaboração de um *software* de cálculo de tensões pelo método dos

elementos finitos, utilizando recursos gráficos para o pós-processamento dos resultados obtidos.

Programas de código aberto que calculam por meio do método dos elementos finitos e produzem gráficos para auxiliar na compreensão dos resultados não são novidade. Como mencionado anteriormente, existem bibliotecas em Python com esse propósito. Portanto, a ideia desse trabalho é escrever um programa autoral que não deixe de utilizar bibliotecas externas mas que tenha as partes de cálculo do método dos elementos finitos e pós-processamento feitas de forma autoral.

1.3 Organização do trabalho

No capítulo 1 são expostos conceitos gerais do trabalho e introdução dos temas discutidos no mesmo.

O capítulo 2 conta com uma revisão bibliográfica sobre o método dos elementos finitos, implementações computacionais do mesmo, abordagens históricas do método e também sobre a interação do método com a linguagem Python.

O capítulo 3 aborda a metodologia utilizada neste trabalho, analisando tanto a implementação do código do método dos elementos finitos quanto o pós-processamento do mesmo.

O capítulo 4 aborda os resultados dos testes realizados.

Por fim, o capítulo 5 analisa o que foi realisado no trabalho e conclui com os pareceres finais.

2 REVISÃO BIBLIOGRÁFICA

2.1 Método dos elementos finitos

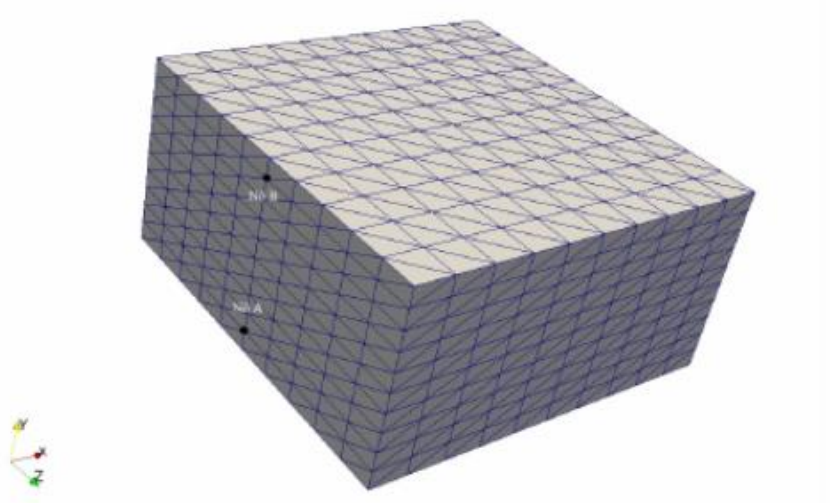
O Método dos Elementos Finitos (MEF) é um método numérico de discretização para obter valores aproximados de soluções de equações diferenciais. O método realiza a subdivisão do domínio em pequenas partes com geometrias conhecidas, denominadas elementos finitos (MEEK, 1996).

O procedimento consiste em realizar a solução de forma aproximada para as equações de campo em um subdomínio, para em seguida por um processo de “assemblagem” obter a solução em todo o domínio, preservando-se a continuidade ou o equilíbrio dos nós entre os elementos (MEEK, 1996).

O conceito da divisão do domínio em partes menores possui alguns benefícios, como a representação de geometrias complexas em modelos mais simples compostos por geometrias básicas. Com isso, a identificação de efeitos localizados se torna possível.

As subdivisões modeladas em geometrias básicas são unidas por “nós”, que são pontos em comum nas arestas das subdivisões. Um exemplo da divisão de um modelo em uma malha de subdivisões é apresentado Figura 2.

Figura 2 - Representação de uma geometria complexa dividida em uma malha de elementos finitos.



Fonte: (ABREU; CATABRIGA, 2017).

Não há como pontuar um momento na história como nascimento do método dos elementos finitos. Acredita-se que o método tenha surgido como uma solução para problemas no campo de análises estruturais nas áreas de Engenharia Civil e Engenharia Mecânica. Há diversas personalidades a serem nomeadas como pioneiras no estudo do MEF.

Richard Courant, na década de 1940, foi um dos primeiros a desenvolver a idéia de aproximar o domínio por subdomínios triangulares finitos para resolver um problema de equações diferenciais de segundo grau (GANDER; WANNER, 2012).

Walter Ritz foi um físico-matemático desenvolvedor do Método de Ritz. O princípio do método consiste na definição de um espaço finito de funções para achar uma aproximação para a solução do problema por meio de combinações lineares das funções determinadas. Ritz utilizou seu método na mecânica quântica, na área de linhas espectrais de átomos (GANDER; WANNER, 2012).

Boris Galerkin (nascido em 1871), matemático, engenheiro e ativista político soviético desenvolveu um método (Método de Galerkin ou, após as contribuições de Ivan Bubnov (1872-1919), Método de Bubnov-Galerkin) para otimizar a tarefa de se encontrar um determinado número limitado de equações algébricas para aproximar uma equação diferencial. O método atual derivado se chama Método de resíduos ponderados (GANDER; WANNER, 2012).

O método de Galerkin é um bom exemplo de um método fácil para codificar, sendo assim um dos mais utilizados em programas que fazem uso do método dos elementos finitos.

2.2 Implementação computacional do Método dos elementos finitos

Como demonstrado no item 2.1, as bases do cálculo realizado no método dos elementos finitos já eram conhecidas há tempos, no entanto o número grande de cálculos a serem realizados impedia a utilização do mesmo.

A precisão do método está atrelada à quantidade de subdivisões presentes no modelo, quanto mais e menores subdivisões, melhor a precisão. No entanto com o

aumento de subdivisões, o número de cálculos a serem realizados cresce (GANDER; WANNER, 2012).

Esse problema pode ser contornado ao “delegar” ao computador a tarefa de realizar esses cálculos, apresentando apenas as condições que devem ser calculadas.

2.2.1 Abordagem histórica da implementação computacional do MEF

Edward L. Wilson é considerado um dos precursores de análises computacionais utilizando o método dos elementos finitos. A autoria do primeiro programa computacional utilizando o método dos elementos finitos é atribuída a ele, em uma tese de mestrado orientada por Ray Clough, publicada em 1962 (CLOUGH; WILSON, 1962).

Em 1996, Graham Archer desenvolveu um programa que realiza cálculos em elementos finitos utilizando a linguagem C++, com foco no desenvolvimento do programa com programação orientada a objetos. O programa foi feito de forma a ser replicado de uma maneira fácil e que aceitasse a adição de novas funções (ARCHER, 1996).

Atualmente o *software* FEniCS é amplamente utilizado para análises utilizando o método dos elementos finitos. O projeto começou a ser desenvolvido em 2003 como uma pesquisa colaborativa entre a Universidade de Chicago e a Universidade de Chalmers, na Suécia. O *software* é gratuito e aberto e oferece bibliotecas em diferentes linguagens, inclusive Python, para que usuários utilizem dos recursos da melhor maneira possível (LOGG; MARDAL; WELLS, 2011).

2.3 Python

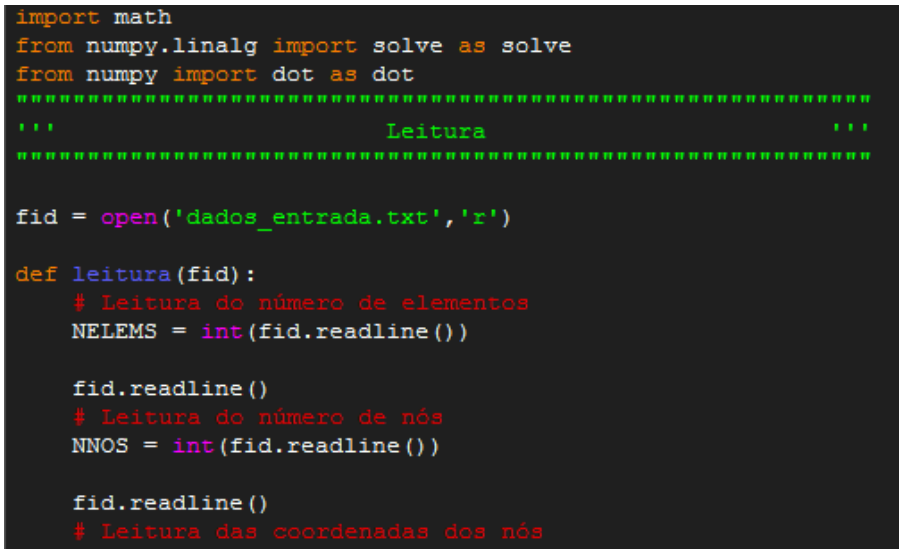
Python é uma linguagem de programação de alto nível de código aberto caracterizada pela fácil legibilidade dos códigos, o que faz com que a linguagem seja muito comum no ambiente acadêmico.

Foi desenvolvida por Guido van Rossum em 1990 no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI) (BORGES, 2010).

Python sempre teve um ideal de gratuidade, código aberto, disponibilidade e acessibilidade (é possível programar em Python em diversos sistemas operacionais, assim como celulares e tablets, entre outros sistemas) e com um foco principal na clareza da sintaxe.

A interface em Python, como na Figura 3, pode ser usada para realizar os cálculos e traduzir complexos modelos científicos em códigos de elementos finitos.

Figura 3 - Exemplo de código escrito em Python



```
import math
from numpy.linalg import solve as solve
from numpy import dot as dot
######################################################################
'''                        Leitura                                 '''
######################################################################

fid = open('dados_entrada.txt','r')

def leitura(fid):
    # Leitura do número de elementos
    NELEMS = int(fid.readline())

    fid.readline()
    # Leitura do número de nós
    NNOS = int(fid.readline())

    fid.readline()
    # Leitura das coordenadas dos nós
```

Fonte: Autoria própria (2020)

A linguagem de programação em Python pode ser muito eficiente em análises numéricas pela facilidade na sintaxe e boa relação com tratamentos de dados para obter uma interpretação dos resultados.

2.4 Pós-processamento em Python

Diversas bibliotecas que podem ser importadas para ajudar no pós-processamento dos dados obtidos pelo método dos elementos finitos já foram desenvolvidas em Python. Entre as que realizam o cálculo do método e fazem o pós-processamento, podemos citar FiPy, SfePy e principalmente FEniCS, que tem suporte não só em Python mas em outras linguagens.

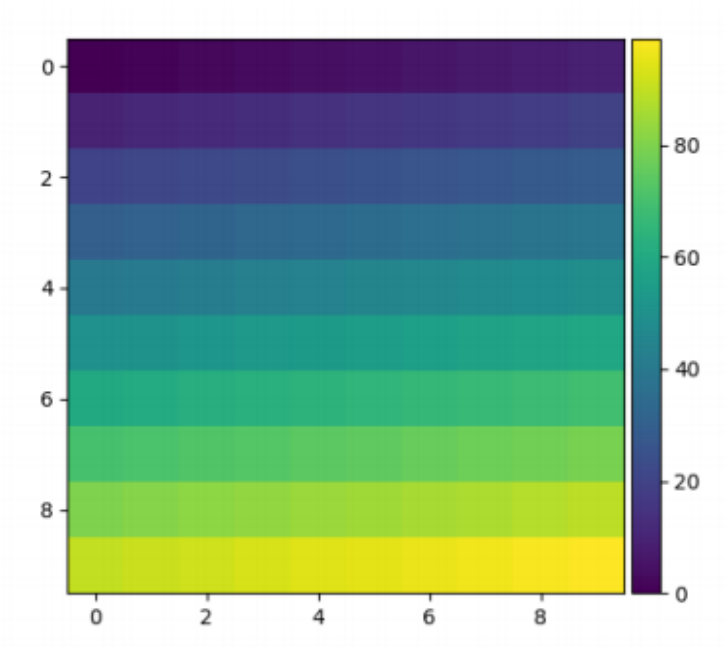
2.4.1 Matplotlib

Matplotlib é uma biblioteca de software desenvolvida em Python para a elaboração de gráficos e recursos visuais para visualização de dados, oferecendo diversos modelos para que os dados sejam representados. Foi desenvolvida por John Hunter em 2003 e continuada por Michael Droetboom e Thomas Caswell após o falecimento do autor original (HUNTER, 2007).

A interface de programação orientada a objetos oferece gráficos similares aos gráficos de MATLAB, mas com o intuito de ser um código *open source*, é programado em Python e não possui nenhum tipo de licença ou restrição.

Na Figura 4 é possível ver um exemplo de gráfico que pode ser desenvolvido utilizando a biblioteca Matplotlib.

Figura 4 - Exemplo de utilização da biblioteca Matplotlib



Fonte: HUNTER et al, 2020

2.5 Tensões em colunas verticais

Um dos problemas mais comuns em operações de perfuração utilizando colunas verticais é a falha por fadiga, que faz com que o custo da perfuração aumente de forma significativa (ZHENG et al., 2014).

Embora haja diversos estudos na área e o assunto seja discutido frequentemente, a frequência em que esses eventos ocorrem continua alta. De acordo com Brun, Aerts e Jerkø (2015), os custos com perfuração e completação representam de 40% a 50% do total do CAPEX em operações médias. Em maiores operações estes custos chegam a ser de 65%.

O estudo das tensões em colunas verticais pode ser um grande aliado na redução dos custos de operações de perfuração, visto que parte das falhas por fadiga ocorre devido às concentrações de tensões (VAISBERG et al., 2002).

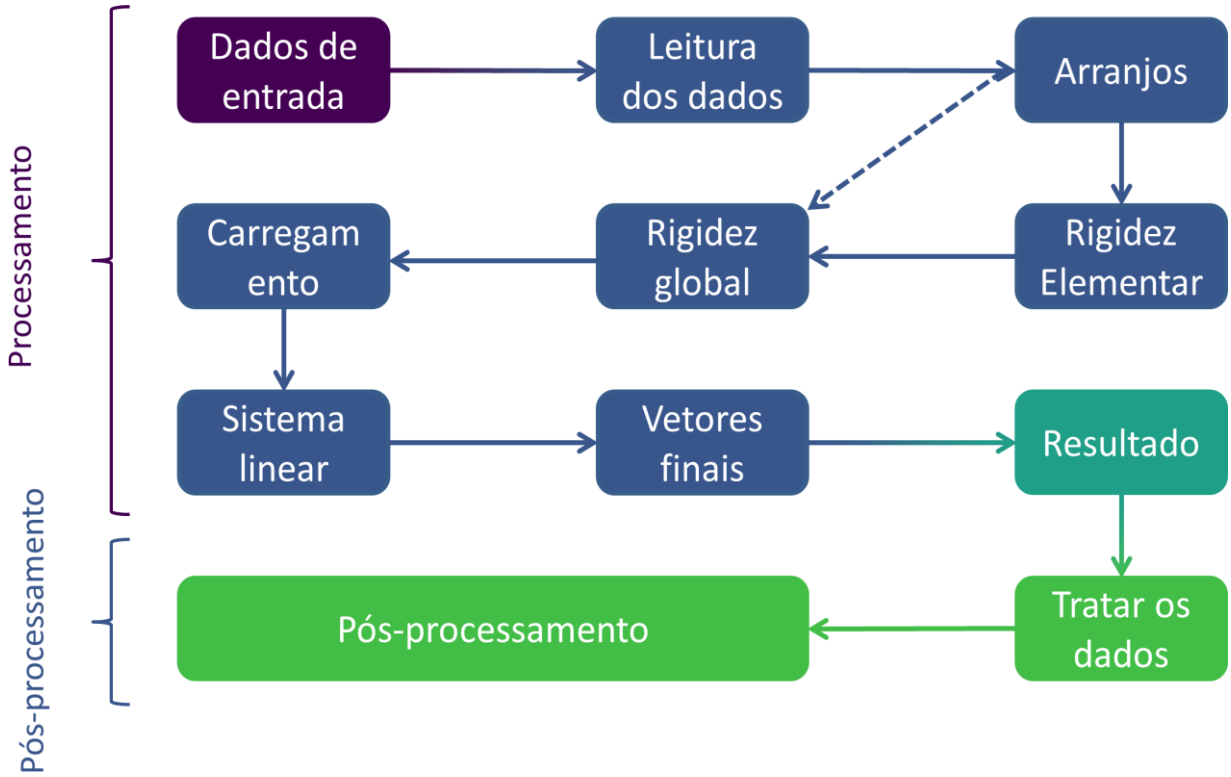
3 METODOLOGIA

Para realizar o processamento dos dados a partir de um certo *input*, foi necessário desenvolver um programa que realizasse o cálculo do método dos elementos finitos. O orientador Prof. Ronaldo Carrion (2020)[1] forneceu um programa de autoria própria desenvolvido na linguagem MATLAB para realizar os cálculos.

Este programa foi traduzido para a linguagem Python para ser integrado com o programa desenvolvido para realizar o pós-processamento dos dados e fornecer os recursos visuais.

Na Figura 5 pode-se visualizar o fluxograma do desenvolvimento dos códigos.

Figura 5 - Fluxograma do desenvolvimento dos códigos.



Fonte: Autoria própria (2020).

[1] Código fornecido pelo orientador do trabalho, Prof. Dr. Ronaldo Carrion, para adaptação e tradução para linguagem Python

3.1 Processamento

O código original em MATLAB que realizava a parte de processamento do programa, era dividido em diversas funções (como observado no fluxograma da Figura 5) que eram compiladas em um programa principal.

Na tradução para Python optou-se por escrever todas as funções em um só programa (no mesmo arquivo), a fim de centralizar e simplificar o processo de produção do código.

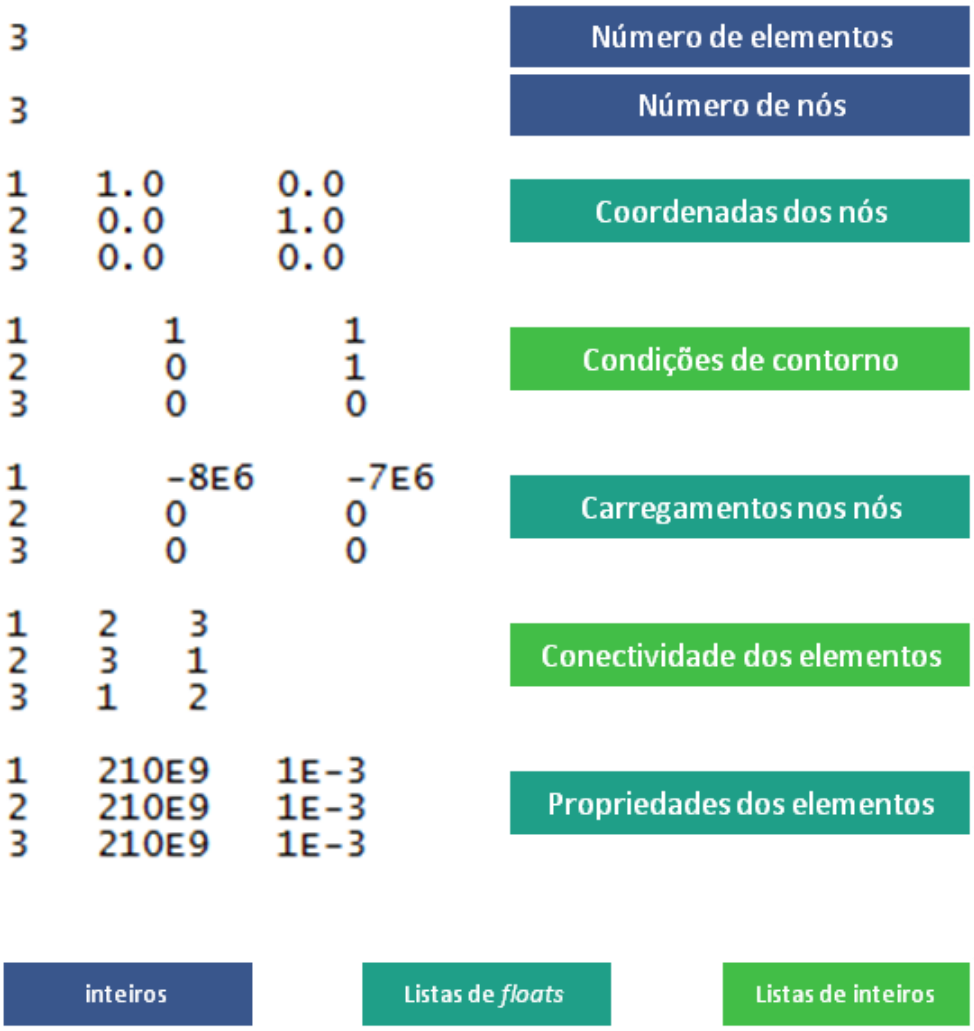
Naturalmente, é comum dividir programas em códigos diferentes e compilá-los em um só código. Isso auxilia no desenvolvimento do programa pelo fato de que quando são realizados testes para verificar o funcionamento mesmo, um programa fragmentado é mais fácil para identificar e solucionar erros no código. A decisão de centralizar o programa foi uma questão de preferência pessoal.

Nos tópicos abaixo são abordadas as etapas de funcionamento do programa descritas na Figura 5.

3.1.1 Dados de entrada

Os dados de entrada são fornecidos em um arquivo de texto com extensão .txt que será lido pelo programa. O formato dos dados contidos nesse arquivo é descrito na Figura 6 abaixo:

Figura 6 - Exemplo de arquivo de texto utilizado como input



Fonte: Autoria própria (2020).

3.1.2 Leitura dos dados

A função que realiza a leitura dos dados está no programa principal. Esta função lê os dados de entrada e armazena os valores em variáveis que serão utilizadas nas próximas funções.

As variáveis que o programa lê e retorna são: Número de elementos, número de nós, coordenadas do nó em x e y, condições de contorno do nó em x e y, carregamento do nó em x e y, conectividade dos elementos e propriedades dos elementos (módulo de Young e área).

Com exceção do número de elementos e do número de nós que são escalares inteiros, todos os outros outputs da função são matrizes (listas de listas).

3.1.3 Arranjos

Esta função calcula os arranjos com e sem as condições de contorno a partir dos vetores e escalares obtidos nos dados de entrada.

Os arranjos realizam a definição da posição dos elementos e nós na matriz de rigidez que irá compor o sistema linear a ser resolvido para se obter os vetores de solução.

3.1.4 Rigidez

As funções abaixo calculam as matrizes de rigidez utilizadas no programa.

3.1.4.1 Rigidez elementar

Função utilizada para calcular a matriz de rigidez de um elemento do objeto. Retorna uma matriz 4x4 que irá compor a matriz global de rigidez.

A partir de informações do comprimento e coordenadas do elemento, é obtido um ângulo θ, que fornece os parâmetros seno e cosseno. A matriz retornada tem seu formato apresentada na Figura 7 abaixo:

Figura 7 - Matriz de rigidez elementar

$$\begin{bmatrix} (\alpha)(\cos^2) & (\alpha)(\text{sen})(\cos) & (\alpha)(-\cos^2) & (\alpha)(-\text{sen})(\cos) \\ (\alpha)(\text{sen})(\cos) & (\alpha)(\text{sen}^2) & (\alpha)(-\text{sen})(\cos) & (\alpha)(-\text{sen}^2) \\ (\alpha)(-\cos^2) & (\alpha)(-\text{sen})(\cos) & (\alpha)(\cos^2) & (\alpha)(\text{sen})(\cos) \\ (\alpha)(-\text{sen})(\cos) & (\alpha)(-\text{sen}^2) & (\alpha)(\text{sen})(\cos) & (\alpha)(\text{sen}^2) \end{bmatrix}$$

Fonte: Autoria própria (2020).

Onde α representa a seguinte multiplicação:

$$\alpha = \frac{E * A}{L}$$

Na equação acima, “E” representa o módulo de elasticidade do elemento, “A” representa a Área do elemento e “L” representa a comprimento cartesiano do elemento.

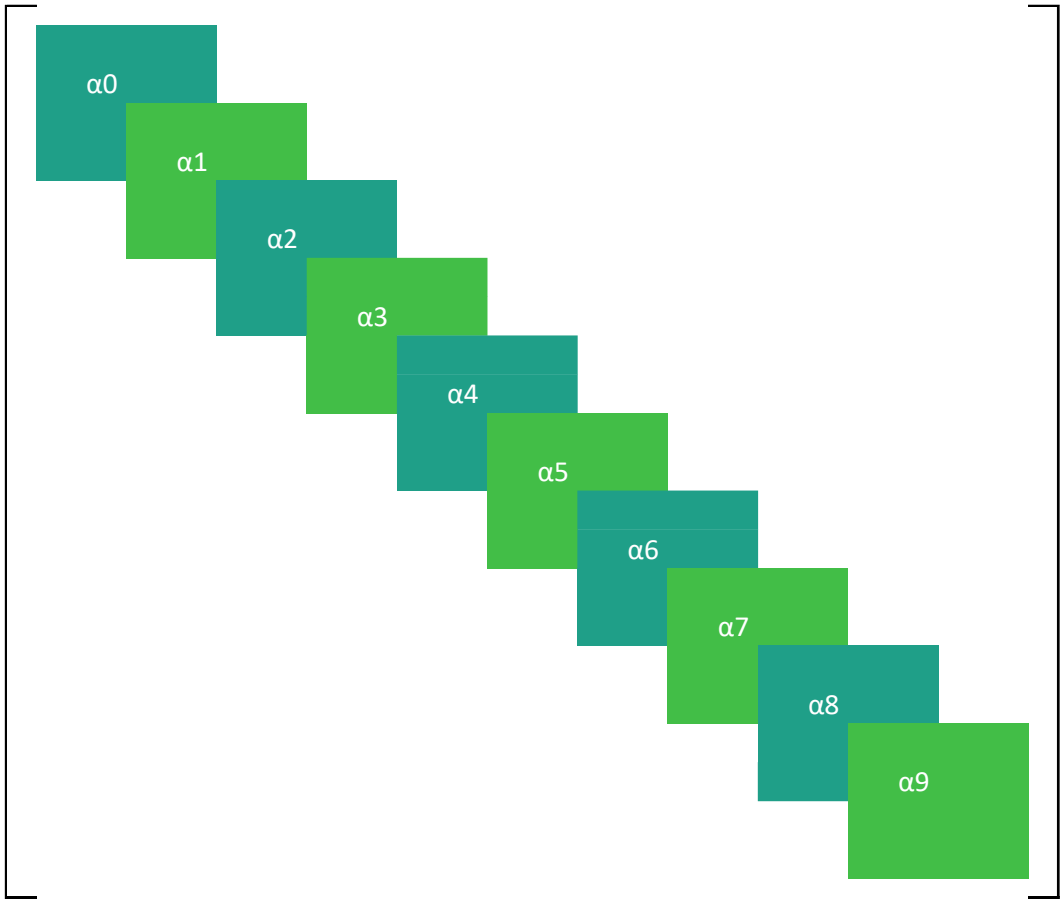
3.1.4.2 Rigidez global

Função utilizada para calcular a matriz de rigidez de todo o conjunto de elementos. Realiza iterações entre os elementos utilizando-se da função de Rigidez elementar para retornar a matriz de rigidez de cada elemento e assim compor a matriz de rigidez global.

A posição das matrizes de cada um dos elementos na matriz global é definida pelos vetores obtidos na função de arranjos.

Na Figura 8 abaixo é possível perceber como é estruturada a matriz de rigidez global:

Figura 8 - Exemplo de estruturação da matriz de rigidez global



Fonte: Autoria própria (2020).

Onde os quadrados coloridos representam as matrizes de rigidez elementar e α_n representa a posição da matriz de rigidez elementar do elemento na matriz de rigidez global, definida pela função de arranjos. A sobreposição de matrizes representa os graus de liberdade compartilhados entre os elementos.

3.1.5 Carregamento

Esta função realiza o cálculo do vetor de carregamento para os elementos do conjunto.

São realizados cálculos em iterações dos graus de liberdade de cada um dos nós, a fim de se obter um vetor final com o carregamento no conjunto.

3.1.6 Sistema linear

Durante a execução do código, é necessário resolver alguns sistemas lineares.

Há diversas formas de se resolver um sistema linear. Como o método de resolução do sistema linear não é do escopo deste trabalho, foi utilizada uma biblioteca externa para realizar esta tarefa.

O método linalg.solve() da biblioteca NumPy realiza a resolução do sistema linear utilizando a decomposição LU, conhecido método na álgebra linear.

3.1.7 Vetores finais

Por fim, com base nos dados calculados, a última função realiza o cálculo dos vetores de deslocamento e forças em cada um dos nós do conjunto.

A função retorna dois vetores: um para os deslocamentos e um para os carregamentos. Ainda é realizado outro cálculo para se obter as tensões. Por se tratar de um elemento de barra, basta dividir o carregamento pela área do elemento correspondente.

3.2 Pós-processamento

O pós-processamento dos dados é realizado em um programa separado, então naturalmente, pela dinâmica do Python, o primeiro passo do programa é “importar” a função principal do programa de processamento para que este realize os cálculos que fornecem os dados a serem pós-processados.

3.2.1 Tratamento dos dados

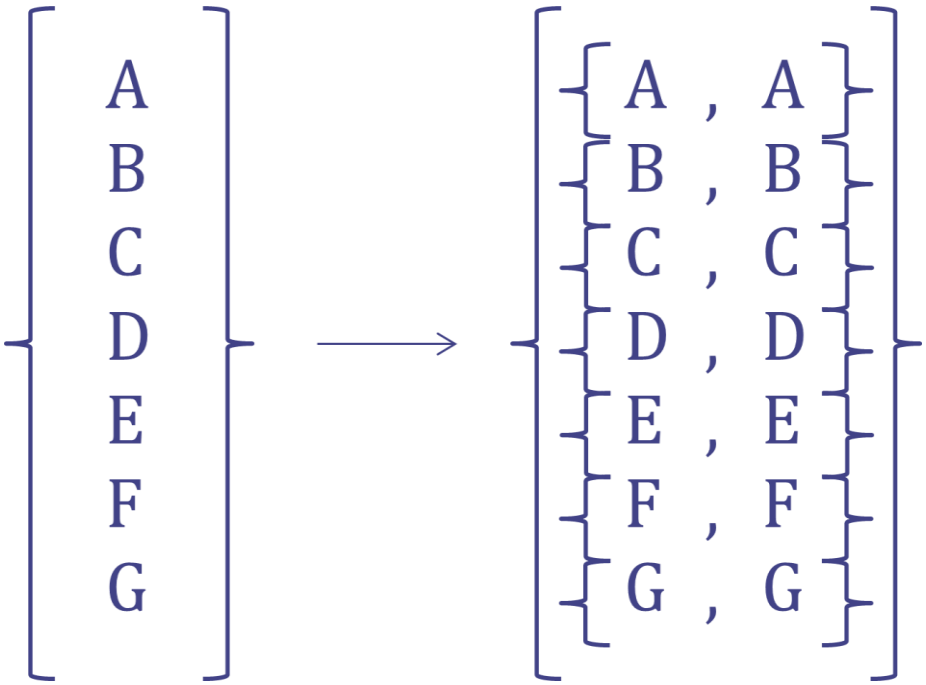
Com os dados em posse, um tratamento dos mesmos é realizado. Um primeiro filtro é aplicado em valores com módulo abaixo de 1E-10. Para estes valores é atribuído um valor de 0 para reduzir os ruídos indesejados nos gráficos a serem fornecidos.

Outra alteração realizada é a inversão dos vetores, visto que originalmente, os últimos dados dos vetores representam os valores “mais acima” na prática e os primeiros os “mais abaixo”. Essa inversão é crucial para que a visualização seja mais bem interpretada, pois assim o gráfico possui a orientação vertical correta (a parte de cima da coluna de perfuração fica na parte de cima do gráfico e vice-versa).

Para fornecer os gráficos foi utilizada a função pcolormesh da biblioteca Matplotlib. Essa função requer que os argumentos a serem representados estejam definidos em listas bi-dimensionais (listas de listas). Como os dados que serão representados são unidimensionais, é necessário transformá-los em bi-dimensionais para que seja possível utilizá-los na função pcolormesh.

A transformação é simples e ocorre “duplicando” os valores originais do vetor unidimensional em duas colunas com o vetor, tendo assim uma matriz Nx2 (onde N é o tamanho do vetor) em que as duas colunas são iguais, como descrito na Figura 9.

Figura 9 - Transformação de dados unidimensionais para dados bi-dimensionais



Fonte: Autoria própria (2020).

Além disso, foram desconsiderados os resultados para os deslocamentos e carregamentos no eixo X, visto que a análise que faremos é unidimensional e estes resultados seriam nulos.

O fato dos resultados do eixo X serem nulos foi utilizado para validar os resultados do programa, verificando que os testes realizados forneciam resultados coerentes.

3.2.2 *Setup* dos gráficos

Como a intenção do programa é gerar os gráficos para que fiquem posicionados lado a lado, visando uma melhor visualização e comparação entre as grandezas representadas, foi utilizada a função *subplots* (matplotlib.pyplot.subplots) para realizar esta tarefa.

Como o objeto a representar é unidimensional, foi necessário que a variável de visualização do eixo x fosse definida como não-visível. Isso foi realizado com a intenção de não confundir o leitor com um dado a mais a ser intepretado na leitura do gráfico.

Para a representação dos dados foi utilizada a função *pcolormesh* (matplotlib.pyplot.pcolormesh) pois é a principal função dentro da biblioteca *Matplotlib* para se produzir ”mapas de calor”, tipo de gráfico que será utilizado neste trabalho.

A função *colorbar* (matplotlib.pyplot.colorbar) foi utilizada para fazer a legenda posicionada à esquerda de cada uma das grandezas representadas.

4 RESULTADOS

Com o uso dos programas elaborados, foram realizados testes para verificar a performance do pós-processamento e os *outputs* fornecidos pelo mesmo.

4.1 Teste com 10 elementos

Para o primeiro exemplo testado, foi elaborado um arquivo que simulasse um objeto de 1000 metros de altura com uma força positiva de 10^4N no topo e dois carregamentos negativos de 5.10^3N distribuídos pelo objeto. O material considerado foi aço (Módulo de Young (E) = 210.10^9 N/m²) e a área da seção transversal considerada foi de $1{,}838.10^{-2}$ m² (tubo de 6⅝ in de diâmetro externo e 5,901 in de diâmetro interno) (LYONS e PLISGA, 2005). Uma representação dos dados pode ser visualizada na Figura 10.

Figura 10 - Exemplo de distribuição de forças no objeto

Fonte: Autoria própria (2020).

Para este teste, a disposição do arquivo de entrada é representada pela Figura 11, em um formato de arquivo análogo ao representado pela Figura 6 no capítulo 3. As reticências indicam intervalos onde os dados apresentados possuem um formato semelhante (seguem o mesmo padrão) que as linhas vizinhas. Assim sendo, o arquivo original não é exatamente igual ao representado na figura.

Figura 11 - Exemplificação dos dados de entrada do primeiro teste.

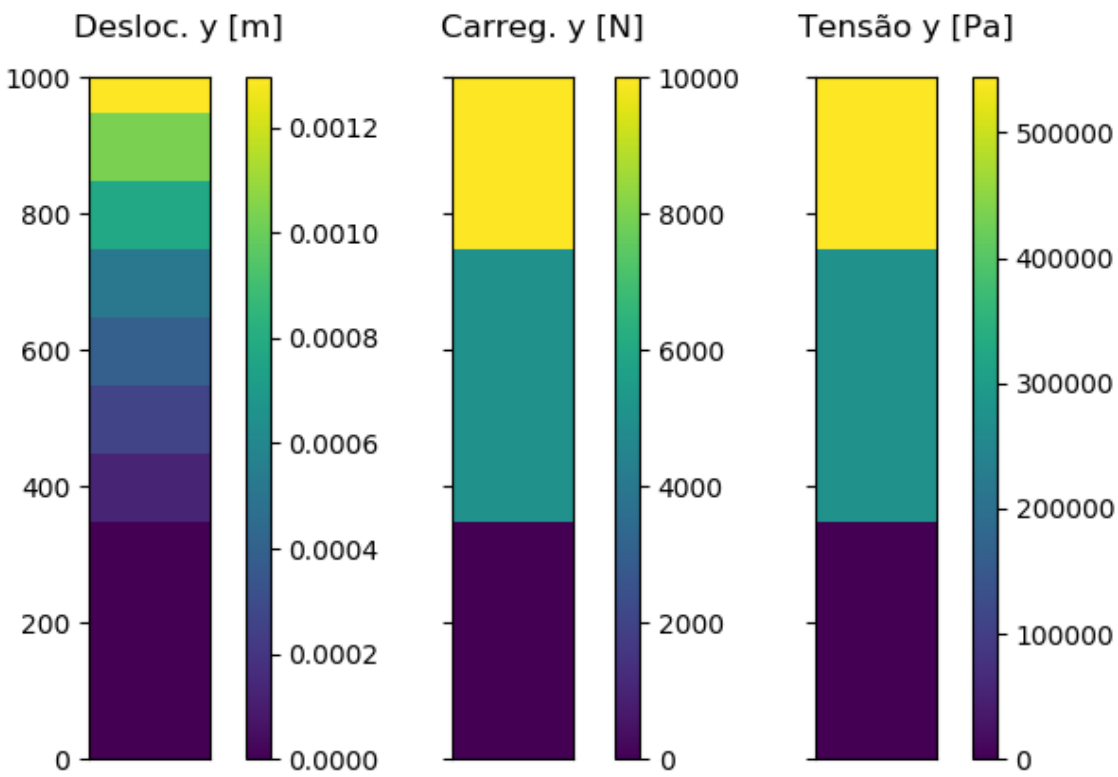
```
10

11

1   0.0     0.0
2   0.0     100.0
      ...
11  0.0     1000.0

1       0       0
2       0       1
      ...
11      0       1

1       0       0
2       0       0
3       0       0
4       0       -5E3
5       0       0
6       0       0
7       0       0
8       0       -5E3
9       0       0
10      0       0
11      0       1E4

1   1   2
2   2   3
   ...
10  10  11

1   210E9   1.838E-2
2   210E9   1.838E-2
      ...
10  210E9   1.838E-2
```

Fonte: Autoria própria (2020).

O pós-processamento dos resultados deste teste pode ser observado na Figura 12, onde os três gráficos representam os vetores de deslocamento, carregamento (força axial) e tensão obtidos para o eixo y, com suas respectivas legendas de cores ao lado esquerdo de cada grandeza, de forma a auxiliar a interpretação.

A execução por completo do programa foi realizada em 0,26 segundos, onde 0,002 segundos foram empregados no processamento e o resto no pós-processamento.

Figura 12 - Pós-processamento dos dados do primeiro teste com dez elementos.



Fonte: Autoria própria (2020).

4.2 Teste com 100 elementos

No segundo teste foi realizada uma análise com 100 elementos em condições similares à primeira. A diferença consta na disposição das forças: Ao invés de focos pontuais de forças aplicadas, serão distribuídas forças negativas em todos os elementos (além da força positiva no topo) de modo a simular a ação da “força peso” agindo sobre o objeto.

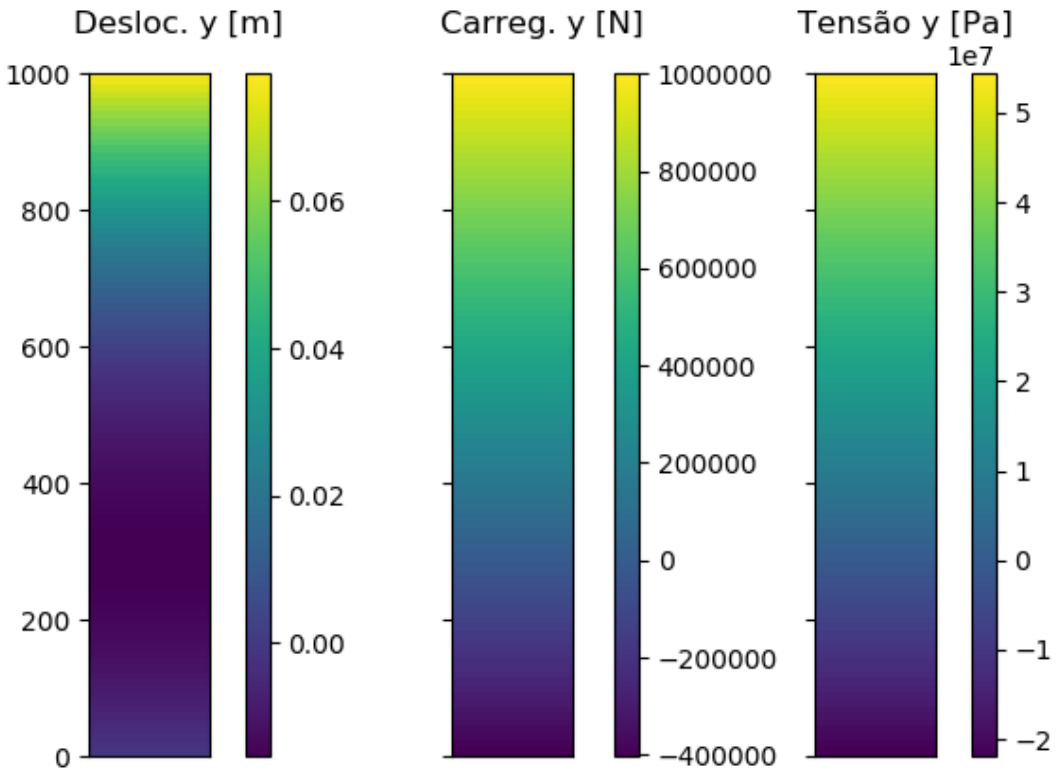
O arquivo de dados de entrada foi modelado de modo que a força positiva no topo seja inferior à soma das forças negativas dispostas ao longo do objeto, simulando o que ocorre em uma coluna de perfuração, onde, na parte inferior do objeto, há compressão. As informações sobre as propriedades do objeto simulado estão dispostas na Tabela 1.

Tabela 1 - Propriedades do segundo teste

Propriedade	**Medida**
Tamanho da seção (L)	1000 m
Módulo de Young (E)	210.10^9 Pa
Densidade (ρ)	7860 kg/m³
Diâmetro externo (OD)	6⅝ in
Diâmetro interno (ID)	5.901 in
Força de içamento (F)	1.10^6 N
Força peso nos nós (P)	-14172,9 N

Assim, o pós-processamento dos resultados é observado na Figura 13, onde são apresentados os gráficos de deslocamento, carregamento e tensão no eixo y. O tempo de execução para este exemplo foi de 0,3 segundos, onde 0,05 segundos foram empregados no processamento e 0,25 segundos no pós-processamento.

Figura 13 - Pós-processamento dos dados do segundo teste com cem elementos.



Fonte: Autoria própria (2020).

Na Figura 13 é possível observar valores negativos de deslocamento, carregamento e tensão, devido ao estado de compressão na região entre y = 200 e y = 400 metros.

4.3 Teste com 1000 elementos

Para o teste com 1000 elementos, optou-se por realizar uma simulação mais próxima à realidade, com diferentes dimensões de diâmetros internos e externos para diferentes tamanhos de seções, assim, simulando as partes que compõem uma coluna vertical de perfuração, o *Drillpipe*, o *Heavyweight drillpipe* e o *Drillcollar* (LYONS e PLISGA, 2005).

As dimensões e propriedades utilizadas em cada uma das seções definidas, assim como suas respectivas extensões verticais, são descritas na Tabela 2.

Assim como no segundo teste, foi distribuída uma força peso para cada um dos nós (exceto o último, onde se aplica uma força positiva de içamento).

Tabela 2 - Propriedades do terceiro teste.

Propriedade	***Drillpipe***	***HW Drillpipe***	***Drillcollar***
Tamanho da seção (L)	800 m	100 m	100 m
Módulo de Young (E)	210.10^9 Pa	210.10^9 Pa	210.10^9 Pa
Densidade (ρ)	7860 kg/m³	7860 kg/m³	7860 kg/m³
Diâmetro externo (OD)	6⅝ in	4½ in	5.00 in
Diâmetro interno (ID)	5.90 in	2.25 in	2.25 in
Força de içamento (F)	$1{,}55.10^6$ N	-	-
Força peso nos nós (P)	-1417,3 N	-2373,5 N	-3115,9 N

Fonte: Adaptado de (LYONS e PLISGA, 2005).

Pode se observar que a área da seção transversal dos nós do *HW Drillpipe* e do *Drillcollar* são maiores, pois apresentam uma diferença (OD – ID) maior que a respectiva diferença para o *Drillpipe*. Assim sendo, temos que:

$$A_{dc} > A_{HWdp} > A_{dp}$$

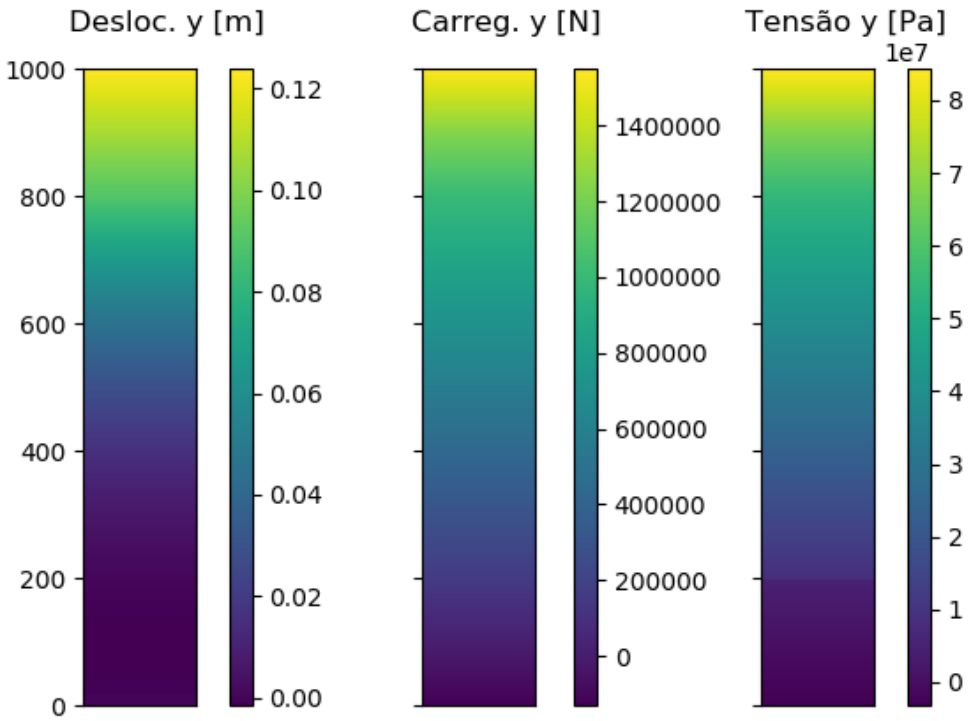
Onde A_{dc} representa a Área da seção transversal do *Drillcollar*, A_{HWdp} representa a área da seção transversal do *HW Drillpipe* e A_{dp} a área da seção do *Drillpipe*.

Como, nesta simulação, todas as partes da coluna são feitas com o mesmo material (aço), as três partes possuem densidades iguais. Assim sendo, a força peso respeita a mesma proporção das áreas descritas, assim como observado na Tabela 2.

Assim como nos testes anteriores, a coluna encontra-se engastada no nó y = 0, prevenindo o movimento do mesmo independente das forças aplicadas.

O resultado do pós-processamento do teste realizado com 1000 elementos pode ser observado na Figura 14. O tempo total de execução do programa foi de 4.24 segundos, onde 3,98 segundos foram empregados no cálculo do processamento e 0,26 segundos no cálculo do pós-processamento.

Figura 14 - Pós-processamento dos dados do terceiro teste com mil elementos.



Fonte: Autoria própria (2020).

É possível observar que a distribuição de tensão apresenta uma diferente linearidade entre y = 0m ~ y = 200m e entre y = 201m e y = 1000m. Isso se deve às diferentes áreas da seção transversal das diferentes partes da coluna.

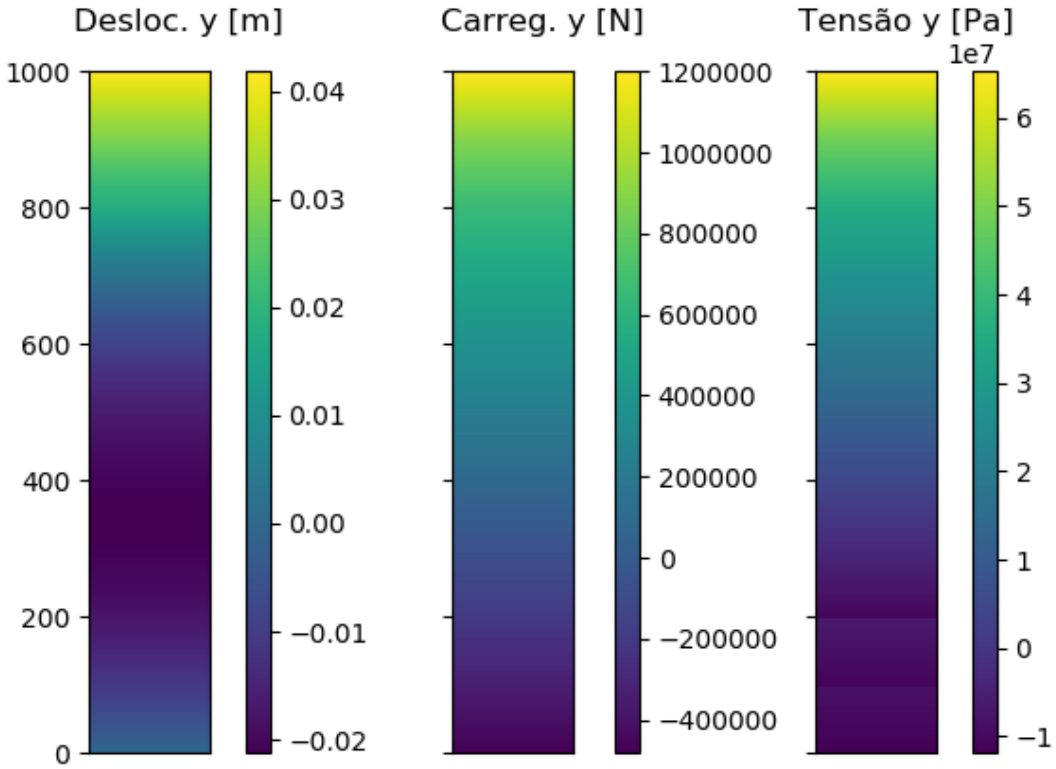
Pela análise do gráfico é possível observar que a região onde a Tensão é nula encontra-se entre y=0m e y = 100m, o que simula o que ocorre na prática, onde a variação do sinal da tensão ocorre geralmente no *drillcollar*.

É possível observar essa diferente distribuição de tensões nas diferentes partes da coluna ao se aplicar uma força de içamento menor, o que levaria a região de

compressão “mais para cima”, e assim obteríamos uma região maior com deslocamentos negativos.

Assim, foi realizado um teste em condições iguais ao do terceiro teste, porém com uma força de içamento de $1{,}2.10^6$N, ante uma força original de $1{,}55.10^6$N. Na Figura 15 é possível observar o resultado do pós-processamento deste teste.

Figura 15 - Pós-processamento do terceiro teste com força de içamento menor.



Fonte: Autoria própria (2020).

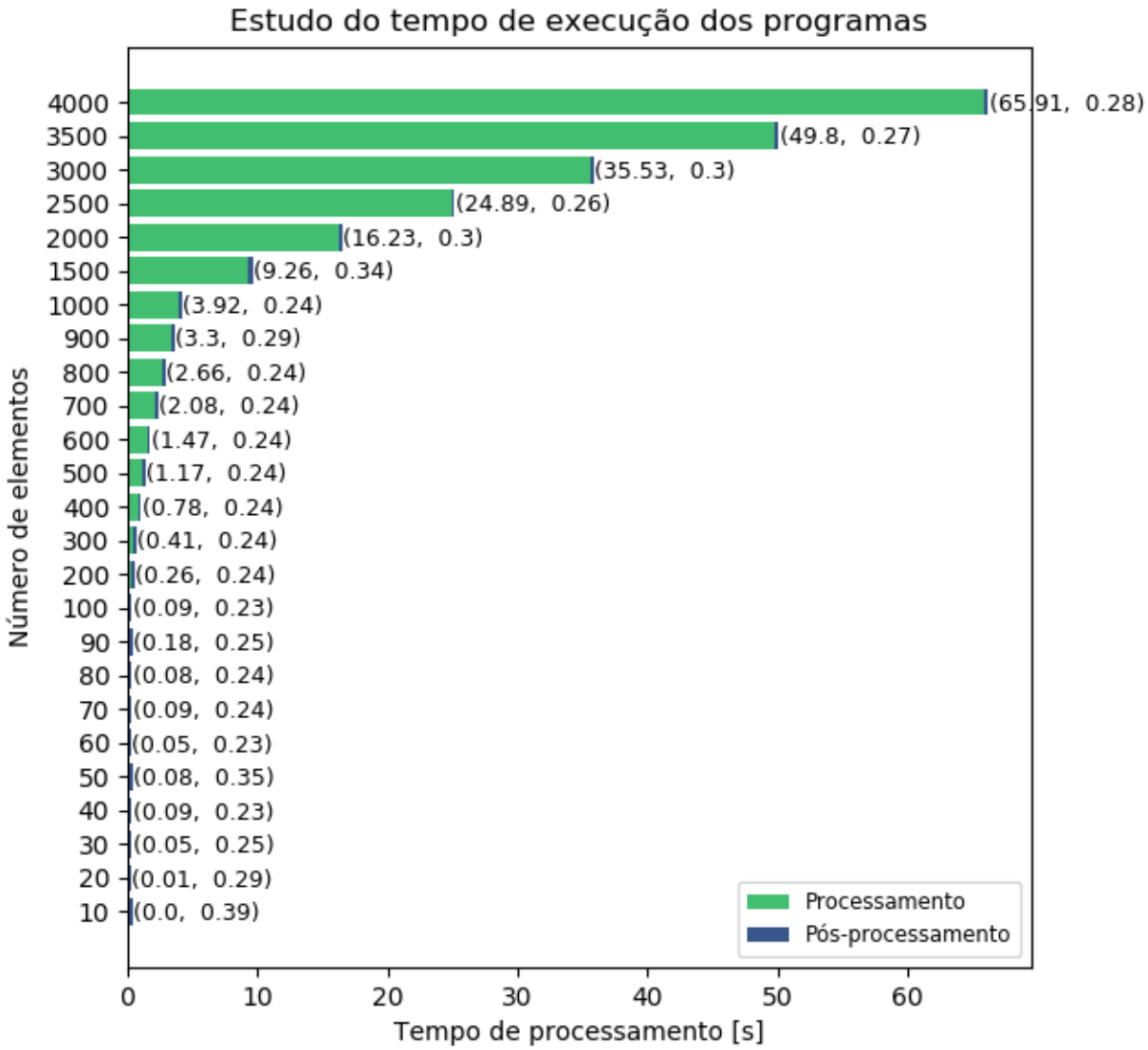
Na Figura 15 fica clara a distinção das diferentes distribuições de tensão para as três partes diferentes da coluna com diferentes áreas. Essa distinção se deve aos diferentes valores de área na seção transversal para as diferentes seções.

4.4 Tempo de processamento

Para dados de entrada contendo um grande número de elementos a resposta no programa não é imediata porque o tempo de execução do código começa a se mostrar significante.

Na Figura 16 é possível observar uma comparação entre o número de elementos contido no arquivo de entrada e os tempos de execução separados no seguinte formato: (tempo de execução do processamento, tempo de execução do pós-processamento).

Figura 16 - Comparação entre o tempo de execução dos programas e a variação do número de elementos



Fonte: Autoria própria (2020).

É possível observar que, apesar de algumas exceções, o tempo do pós-processamento se manteve constante na faixa de 0,23 a 0,29 segundos, com média de 0,266 segundos. Enquanto isso o tempo de processamento vai aumentando de forma não linear, chegando a mais de um minuto para simulações com 4000 elementos.

Outro fato a se notar é o tempo de pós-processamento para dez elementos (0,38s) maior que o tempo de pós-processamento para quatro mil elementos (0,28s). Acredita-se que tais tempos são tão pequenos que fatores aleatórios de processamento do computador (como o número de algarismos flutuantes que retornam das funções de processamento) influenciam este parâmetro, produzindo resultados inconsistentes quando se analisa sem levar em conta tais pontos.

5 CONCLUSÃO

Foi possível obter os resultados desejados, com os gráficos do pós-processamento do cálculo de tensões pelo MEF sendo representados por mapas de cores de forma a facilitar a visualização dos dados numéricos obtidos.

No computador utilizado para realizar este trabalho (Intel i3, 4GB RAM e Windows 7), foi possível testar exemplos com até ±4200 elementos, antes de erros de memória ocorrerem. Isso acontece pois, para um número muito grande de elementos, a tarefa de armazenar os valores no momento da solução do sistema linear demanda uma grande quantidade de memória. Seria possível contornar esse problema com o uso de bibliotecas externas de álgebra linear com foco nessa otimização da solução de sistemas lineares com grandes matrizes.

5.1 Contribuições do trabalho

Com este trabalho, esperou-se mostrar as ferramentas necessárias para realizar a produção de códigos simples que possam fornecer maneiras para visualizar dados obtidos de diversas formas. A utilização de colunas verticais da indústria do petróleo é um exemplo de aplicação dessas ferramentas.

5.2 Trabalhos futuros

Este trabalho realizou a análise do pós-processamento de exemplos estáticos, porém, com algumas mudanças seria possível realizar análises dinâmicas.

Seria necessário definir um conjunto de forças e realizar iterações no processamento para se obter uma imagem de pós-processamento para um momento diferente, com uma força diferente.

Com a biblioteca *Matplotlib* utilizada, é possível armazenar cada uma das figuras do pós-processamento em uma imagem com extensão .png e produzir um arquivo com extensão .gif com todas essas imagens. A biblioteca externa *imageio,* por exemplo, possui recursos para realizar esta função.

REFERÊNCIAS

ABREU, F.; CATABRIGA, L. **Problemas de interação fluido-estrutura via método dos elementos finitos utilizando a Biblioteca FEniCS.** 2017.

ARCHER, G.C. **Objected-Oriented Finite Element Analysis.** 1996.

BORGES, L. E. **Python para desenvolvedores.** 2010.

BRUN, A.; AERTS, G.; JERKØ, M. **How to achieve 50% reduction in offshore drilling costs.** 2015.

CLOUGH, R. W.; WILSON, E. L. **Stress Analysis of a Gravity Dam by the Finite Element Method.** 1962.

CUKIERMAN, H., TEIXEIRA, C. A. N., PRIKLADNICKI, R. **Um Olhar Sociotécnico sobre a Engenharia de Software.** 2007.

GANDER, M. J.; WANNER, G. **From euler, ritz, and galerkin to modern computing.** 2012

HUNTER, J. D. **Matplotlib: A 2D Graphics Environment .** 2007.

HUNTER, J. D.; DALE, D.; FIRING, E.; DROETTBOM, M. **Matplotlib Release 3.3.2**. 2020.

LOGG, A.; MARDAL, K.; WELLS, G. **Automated solutions of differential equations by the Finite Element Method.** 2011.

LYONS, W. C.; PLISGA, G. J. **Standard Handbook of Petroleum & Natural Gas Engineering.** 2005.

MEEK, J. **A brief history of the beginning of the finite element method. International journal for numerical methods in engineering.** 1996.

PENNA, S. **Pós-processador para Modelos Bidimensionais não-lineares do Método dos Elementos Finitos.** 2007.

STEFANUTO, G., FILHO, S., DE LUCCA, J. E., ALVES, A. M. **O impacto do Software Livre e de Código Aberto (SL/CA) nas Condições de Apropriabilidade na Indústria de Software Brasileira**. 2005.

VAISBERG, O.; VINCKÉ, O.; PERRIN, O.; SARDA, J.P.; FAŸ, J.B. **Fatigue of Drillstring: State of the Art.** 2002.

ZHENG, J.; QIU, H.; YANG, J.; BUTT, S. **Fatigue life prediction of Drill-String subjected to random loadings.** 2014.

APÊNDICE A - FUNÇÃO PRINCIPAL DO PÓS-PROCESSAMENTO

Neste anexo é apresentado o código da principal função do programa de pós-processamento, que realiza o tratamento dos dados e a elaboração dos gráficos.

```
import matplotlib.pyplot as plt
from processamento import main
import time

def Principal():
    # Iniciar o tempo
    tempo = time.time()

    # Relizar o processamento
    desloc, carreg, area, ycoord = main(arq = 'nomedoarquivo.txt')

    # Calcular o tempo do processamento
    t0 = time.time() - tempo

    # Formatar o vetor de área para obter a área dos nós
    area.append(area[-1])

    # Tratar o vetor de carregamento
    carreg = tira_x(carreg)
    carreg.reverse()
    carreg = tratamento(carreg)

    #Calcular a tensão
    tensao_y = []
    for i in range (len(carreg)):
        tensao_y.append([carreg[i]/area[i],carreg[i]/area[i]])
    tensao_y.reverse()

    # Tratamento de dados
    desloc_x, desloc_y = [],[]
    carreg_y = []
    eixo_y = []
    eixo_x = [0,1]
    for item in desloc:
        if abs(item[0]) < 1e-6:
            desloc_x.append([0,0])
        else:
            desloc_x.append([item[0],item[0]])
        if abs(item[1]) < 1e-6:
            desloc_y.append([0,0])
        else:
            desloc_y.append([item[1],item[1]])
    for i in range (len(carreg)):
        if abs(carreg[i]) < 1e-6:
```

```
                carreg_y.append([0,0])
            else:
                carreg_y.append([carreg[i],carreg[i]])
        carreg_y.reverse()

        eixo_y.append(ycoord[0])
        for i in range(len(ycoord)-1):
            eixo_y.append((ycoord[i]+ycoord[i+1])/2)
        eixo_y.append(ycoord[-1])

        # Definição da área de plotagem
        fig, (ax0, ax1, ax2) = plt.subplots(ncols=3, sharey = True, share
    x = True)
        ax0.title.set_text('Desloc. y [m]')
        ax0.title.set_position((0.75,1.035))
        ax1.title.set_text('Carreg. y [N]')
        ax1.title.set_position((0.75,1.035))
        ax2.title.set_text('Tensão y [Pa]')
        ax2.title.set_position((0.75,1.035))

        # Plot do deslocamento
        im = ax0.pcolormesh(eixo_x,eixo_y,desloc_y,vmin = achamin(desloc_
    y),vmax = achamax(desloc_y))
        ax0.set_position([0.12,0.11,0.1,0.75])
        ax0.xaxis.set_visible(False)
        fig.colorbar(im, ax = ax0, cax = fig.add_axes([0.25,0.11,0.02,0.7
    5]))

        # Plot do carregamento
        im = ax1.pcolormesh(eixo_x,eixo_y,carreg_y,vmin = achamin(carreg_
    y),vmax = achamax(carreg_y))
        ax1.set_position([0.42,0.11,0.1,0.75])
        ax1.xaxis.set_visible(False)
        fig.colorbar(im, ax = ax1, cax = fig.add_axes([0.555,0.11,0.02,0.
    75]))

        # Plot da tensão
        im = ax2.pcolormesh(eixo_x,eixo_y,tensao_y,vmin = achamin(tensao_
    y),vmax = achamax(tensao_y))
        ax2.set_position([0.72,0.11,0.1,0.75])
        ax2.xaxis.set_visible(False)
        fig.colorbar(im, ax = ax1, cax = fig.add_axes([0.85,0.11,0.02,0.7
    5]))

        # Tempo final
        t1 = time.time()-tempo - t0

        # Exibir o gráfico
        plt.show()
        return t1, t0
```

Universidade de São Paulo

Engenharia de Petróleo – Escola Politécnica

Número: 9882387USP **Data: 16/11/2020**

Pós-processamento de código de elementos finitos na anális de colunas de perfuração

Hugo Veridiano

Orientador: Prof. Ronaldo Carrion

Resumo

Com o avanço da tecnologia na área de Engenharia de Petróleo, novos métodos vão surgindo para ajudar na realização das rotinas executadas. *Softwares* computacionais são grande parte desse avanço, pois auxiliam os profissionais tanto em atividades simples do cotidiano quanto em projetos que demandam mais tempo e atenção. No entanto, boa parte desses *softwares* são pagos e possuem pouca abertura para que o usuário faça adaptações. Pensando nisso, este trabalho tem como objetivo produzir um código computacional na linguagem *Python* que realize o pós-processamento automatizado e otimizado do cálculo de tensões e deslocamentos obtidos pelo método dos elementos finitos em colunas verticais, fornecendo gráficos que auxiliem na compreensão dos resultados, utilizando um código autoral.

Abstract

With the advancement of technology in the area of Petroleum Engineering, new methods are emerging to give help in carrying out the routines performed. Computational softwares are a big part of this advance, as it helps professionals both in simple daily activities and in projects that demand more time and attention. However, most of those softwares are paid for and have small openings for the user to make adaptations. With this in mind, this work aims to produce a computational code in Python that performs the automated and optimized post-processing of stress and displacement calculations of vertical columns, obtained by the Finite element method, providing graphics that assist in the understanding of the results, using a original code.

1. Introdução

Com o surgimento dos computadores e dos *softwares* computacionais, alguns problemas que antes demandavam muito tempo e esforço humano para serem calculados, foram ressignificados. Os *softwares* computacionais fazem parte de umaa revolução tecnológica que auxilia profissionais em diversas atividades.

Boa parte dos *softwares* comerciais possui um código fechado, de modo que o usuário não tenha acesso à implementação do mesmo. Isso acaba por gerar dificuldades na compreensão do funcionamento do programa e de como ele realiza seus cálculos. No entanto, existem *softwares* e plataformas de programação de “código aberto”, ou seja, que são manipuláveis, de forma que o usuário possa alterar o programa para incluir as funções que deseja.

Premissas básicas de liberdade de expressão, acesso à informação e coletividade do conhecimento, que deve ser disponibilizado de forma democrática, são princípios do código aberto (STEFANUTO et al., 2005).

Uma das principais linguagens de programação de código aberto é o Python. Uma linguagem de grande potencial e muito usada academicamente pela fácil leitura e compreensão da sintaxe, além de oferecer suporte para bibliotecas externas que podem ser utilizadas para simplificar o código do usuário.

Entre essas bibliotecas que podem ser implementadas, existem várias que poderiam auxiliar neste trabalho para simplificar o código do programa a ser desenvolvido, entre elas destacam-se Fipy, FEniCS e Sfepy, todas escritas em Python.

Em muitos programas não-comerciais desenvolvidos para o cálculo de equações diferenciais, principalmente o de resolução pelo método dos elementos finitos, um dos grandes entraves é a representação dos dados após serem processados.

Os programas geralmente "retornam" os dados de forma numérica, dificultando a compreensão e a visualização da resolução do problema. Assim, utilizar programas que realizem o "pós-processamento" destes cálculos é importante para que haja compreensão visual dos cálculos realizados.

O pós-processamento é a etapa em que é realizado o tratamento dos dados obtidos de forma a representar as grandezas envolvidas no problema com clareza e objetividade (PENNA, 2007), auxiliando na compreensão dos resultados obtidos e na interpretação dos mesmos.

Um exemplo de pós-processamento é o da Figura 1 abaixo, realizado por um programa que utiliza a biblioteca FEniCS.

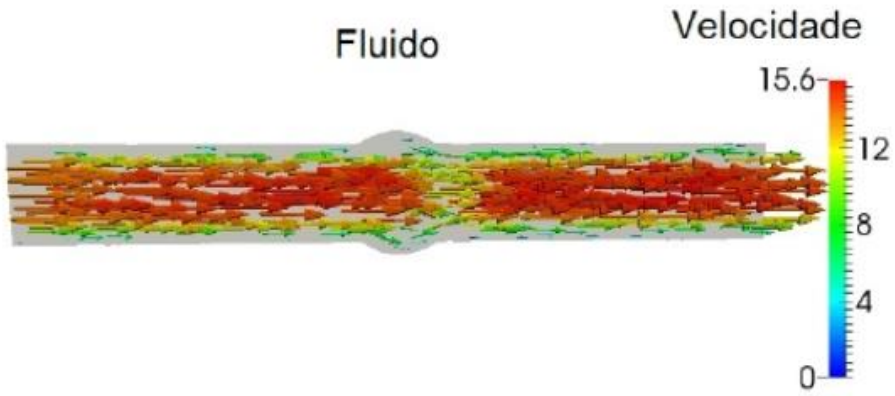


Figura 1 - Exemplo de representação utilizando a biblioteca FEniCS (ABREU; CATABRIGA, 2017).

Assim, pretende-se realizar cálculos por meio de um programa na linguagem Python a fim de computar as tensões distribuídas em colunas verticais de petróleo, realizando os cálculos pelo método de elementos finitos além de proporcionar a visualização desses cálculos obtidos por meio de gráficos que auxiliem a compreensão do resultado.

Entende-se que o cálculo das tensões em colunas verticais é essencial para um projeto de perfuração, tendo em vista que as condições em que a utilização do equipamento ocorre podem ser perigosas para as pessoas que trabalham no projeto. Geralmente esses equipamentos são testados em laboratório em condições desejadas de carregamento (VAISBERG et al, 2002).

2. Metodologia

Para realizar o processamento dos dados a partir de um *input*, foi necessário desenvolver um programa que realizasse o cálculo do método dos elementos finitos. O programa desenvolvido pelo orientador originalmente em MATLAB foi traduzido para a linguagem Python para ser integrado com o programa a ser desenvolvido para realizar o pós-processamento.

2.1. Processamento

O programa que realizava o processamento era dividido em diversas funções, que n tradução para o Python foram compiladas como diferentes funções em um mesmo programa. Nos tópicos abaixo são abordadas as funções do programa de processamento e o formato dos dados de entrada.

2.1.1. Dados de entrada

Os dados de entrada são fornecidos em um arquivo de texto com extensão .txt que será lido pelo programa.

2.1.2. Leitura dos dados

Esta função realiza a leitura do arquivo de entrada e retorna os valores contidos nele: Número de elementos, número de nós, coordenadas do nó em x e y, condições de contorno do nó em x e y, carregamento do nó em x e y, conectividade dos elementos e propriedades dos elementos (módulo de Young e área).

2.1.3. Arranjos

Esta função calcula os arranjos com e sem as condições de contorno a partir dos vetores e escalares obtidos nos dados de entrada. Os arranjos realizam a definição da posição dos elementos e nós na matriz de rigidez.

2.1.4. Rigidez

As funções abaixo calculam as matrizes de rigidez utilizadas no programa.

2.1.4.1. Rigidez elementar

A partir de informações do comprimento e coordenadas do elemento, é obtido um ângulo θ, que fornece os parâmetros seno e cosseno para o cálculo da matriz de rigidez do elemento, de formato 4x4.

2.1.4.2. Rigidez elementar

Realiza iterações entre os elementos utilizando-se da função de Rigidez elementar para retornar a matriz de rigidez de cada elemento e assim compor a matriz de rigidez global.

2.1.5. Carregamento

Esta função realiza o cálculo do vetor de carregamento para os elementos do conjunto. São realizados cálculos em iterações dos graus de liberdade de cada um dos nós, a fim de se obter um vetor final com o carregamento no conjunto.

2.1.6. Carregamento

Durante a execução do código, é necessário resolver sistemas lineares. Há diversas formas de se resolver um sistema linear. Como o método de resolução do sistema linear não é do escopo deste trabalho, foi utilizada uma biblioteca externa para realizar esta tarefa. O método linalg.solve() da biblioteca NumPy realiza a resolução do sistema linear utilizando a decomposição LU, conhecido método na álgebra linear

2.1.7. Vetores finais

Por fim, com base nos dados calculados, a última função realiza o cálculo dos vetores de deslocamento e forças em cada um dos nós do conjunto. A função retorna dois vetores: um para os deslocamentos e um para os carregamentos. Ainda é realizado outro cálculo para se obter as tensões, dividindo o carregamento pela área do elemento correspondente.

2.2. Pós-processamento

O pós-processamento dos dados é realizado em um programa separado, então o primeiro passo do programa é “importar” a função principal do programa de processamento para que este realize os cálculos que fornecem os dados a serem pós-processados.

2.2.1. Tratamento dos dados

São realizados filtros para o tratamento dos dados, anulando valores com módulo menor que 1E-10 (para reduzir ruídos indesejados nos gráficos) e realizando a inversão dos vetores para que a visualização corresponda à realidade e seja mais bem interpretada.

Os valores unidimensionais são bi-dimensionalizados de modo a possibilitar a utilização da função *pcolormesh*da biblioteca Matplotlib, para fornecer os gráficos. Assim, lista se tornam matrizes de duas colunas com valores iguais.

Os valores de deslocamento, carregamento e tensão para o eixo X foram desconsiderados para a análise da coluna de perfuração unidimensional.

2.2.2. Setup dos gráficos

Para a representação dos dados foi utilizada a função *pcolormesh* (matplotlib.pyplot.pcolormesh) pois é a principal função dentro da biblioteca *Matplotlib* para se produzir ”mapas de calor”, tipo de gráfico que será utilizado neste trabalho. A função colorbar (matplotlib.pyplot.colorbar) foi utilizado para fazer a legenda posicionada à esquerda de cada uma das grandezas representadas.

3. Resultados

3.1. Teste com 10 elementos

Para o primeiro exemplo testado, foi elaborado um arquivo que simulasse um objeto de 1000 metros de altura com uma força positiva de 10^4N no topo e dois carregamentos negativos de 5.10^3N distribuídos pelo objeto. O material considerado foi aço (Módulo de Young (E) = 210.10^9 N/m²) e a área da seção transversal considerada foi de $1{,}838.10^{-2}$ m² (tubo de 6⅝ in de diâmetro externo e 5,901 in de diâmetro interno) (LYONS e PLISGA, 2005). Uma representação dos dados pode ser visualizada na Figura 2.

Figura 2 - Exemplo de distribuição de forças no objeto

O primeiro pós-processamento dos resultados deste teste pode ser observado na Figura 3, onde os três gráficos representam os vetores de deslocamento, carregamento (força axial) e tensão obtidos para o eixo y, com suas respectivas legendas de cores ao lado esquerdo de cada grandeza, de forma a auxiliar a interpretação.

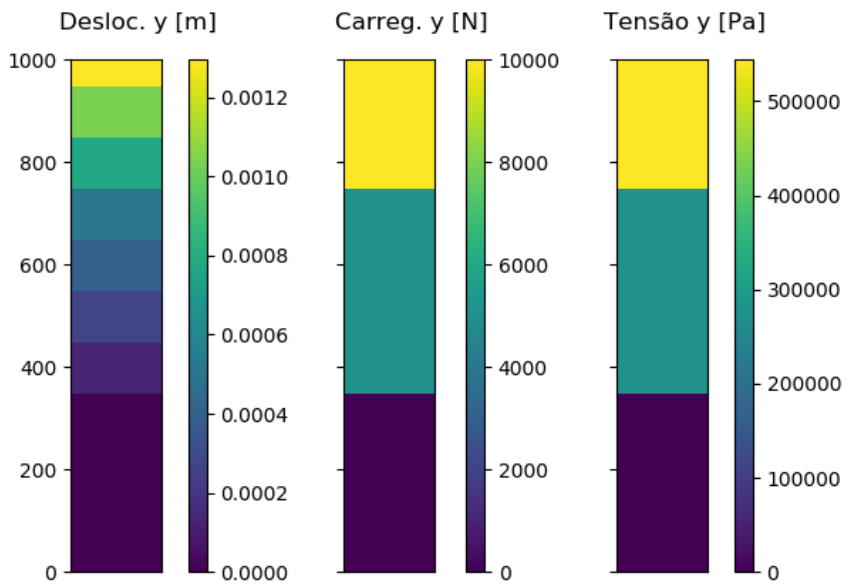


Figura 3 - Pós-processamento dos dados do primeiro teste com dez elementos.

3.2. Teste com 100 elementos

No segundo teste foi realizada uma análise com 100 elementos em condições similares à primeira. A diferença consta na disposição das forças: Ao invés de focos pontuais de forças aplicadas, serão distribuídas forças negativas em todos os elementos (além da força positiva no topo) de modo a simular a ação da "força peso" agindo sobre o objeto. As informações sobre as propriedades do objeto estão dispostas na Tabela 1.

Tabela 1– Propriedades do Segundo teste

Propriedade	**Medida**
Tamanho da seção (L)	1000 m
Módulo de Young (E)	210.10^9 Pa
Densidade (ρ)	7860 kg/m³
Diâmetro externo (OD)	6⅝ in
Diâmetro interno (ID)	5.901 in
Força de içamento (F)	1.10^6 N
Força peso nos nós (P)	-14172,9 N

O pós-processamento do segundo teste é representado pela Figura 4.

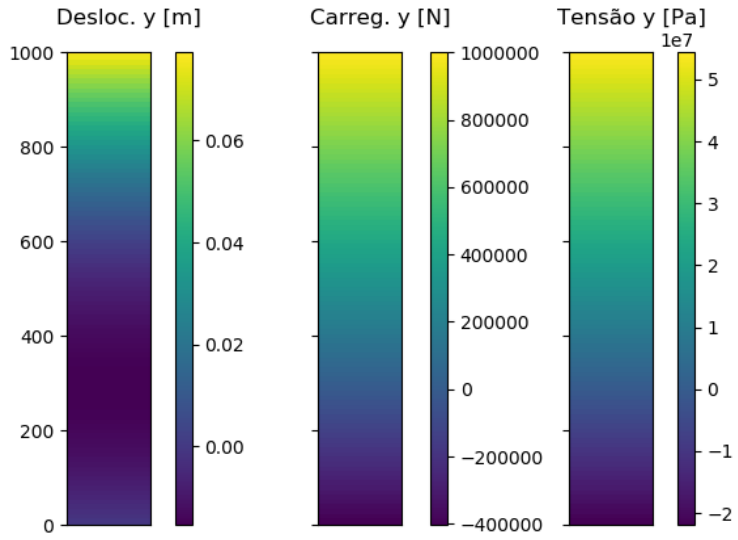


Figura 4 - Pós-processamento dos dados do segundo teste com cem elementos.

3.3. Teste com 1000 elementos

Para o teste com 1000 elementos, optou-se por realizar uma simulação mais próxima à realidade, com diferentes dimensões de diâmetros internos e externos para diferentes tamanhos. As propriedades são dispostas na Tabela 2.

Tabela 1– Propriedades do Terceiro teste

Propriedade	*Drillpipe*	*HW Drillpipe*	*Drillcollar*
Tamanho da seção (L)	800 m	100 m	100 m
Módulo de Young (E)	210.10^9 Pa	210.10^9 Pa	210.10^9 Pa
Densidade (ρ)	7860 kg/m³	7860 kg/m³	7860 kg/m³
Diâmetro externo (OD)	6⅝ in	4½ in	5.00 in
Diâmetro interno (ID)	5.90 in	2.25 in	2.25 in
Força de içamento (F)	$1,55.10^6$ N	-	-
Força peso nos nós (P)	-1417,3 N	-2373,5 N	-3115,9 N

O pós-processamento do terceiro teste é representado pela Figura 5.

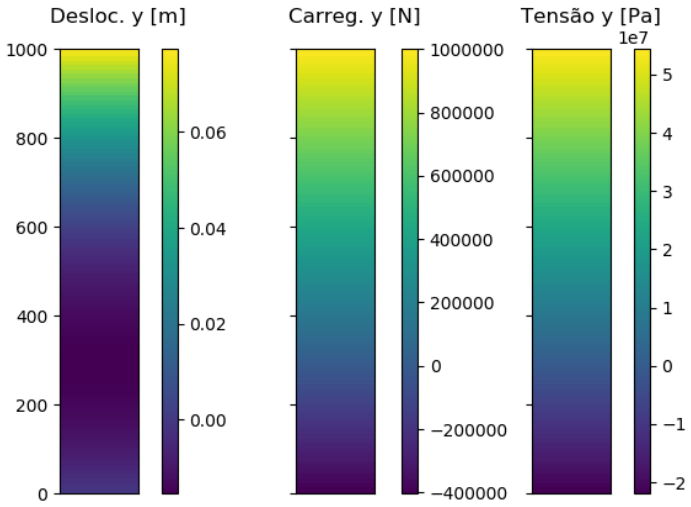


Figura 5 - Pós-processamento dos dados do terceiro teste com mil elementos.

4. Conclusão

Foi possível obter os resultados desejados, com os gráficos do pós-processamento do cálculo de tensões pelo MEF sendo representados por mapas de cores de forma a facilitar a visualização dos dados numéricos obtidos.

4.1. Contribuições do trabalho

Com este trabalho, esperou-se mostrar as ferramentas necessárias para realizar a produção de códigos simples que possam fornecer maneiras para visualizar dados obtidos de diversas formas. A utilização de colunas verticais da indústria do petróleo é um exemplo de aplicação dessas ferramentas.

4.2. Trabalhos futuros

Este trabalho realizou a análise do pós-processamento de exemplos estáticos, porém, com algumas mudanças seria possível realizar análises dinâmicas.
Seria necessário definir um conjunto de forças e realizar iterações no processamento para se obter uma imagem de pós-processamento para um momento diferente, com uma força diferente.

5. Conclusão

ABREU, F.; CATABRIGA, L. Problemas **de interação fluido-estrutura via método dos elementos finitos utilizando a Biblioteca FEniCS**. 2017.

LYONS, W. C.; PLISGA, G. J. Standard **Handbook of Petroleum & Natural Gas Engineering**. 2005.

PENNA, S. **Pós-processador para Modelos Bidimensionais não-lineares do Método dos Elementos Finitos**. 2007.

STEFANUTO, G., FILHO, S., DE LUCCA, J. E., ALVES, A. M. **O impacto do Software Livre e de Código Aberto (SL/CA) nas Condições de Apropriabilidade na Indústria de Software Brasileira**. 2005.

VAISBERG, O.; VINCKÉ, O.; PERRIN, O.; SARDA, J.P.; FAŸ, J.B**. Fatigue of Drillstring: State of the Art**. 2002.