PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Matias Maranhão

Desenvolvimento de um software de Simulação do processo de deslocamento de fluidos em anular de poços de petróleo

Projeto de Graduação

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

Orientador: Márcio Carvalho

Corientador: Frederico Gomes

Rio de Janeiro

Dezembro de 2019

AGRADECIMENTOS

Agradeço aos meus pais pela oportunidade de ter estudado aonde estudei, o caminho foi longo e preciso reconhecer que sem eles, eu jamais chegaria aqui neste dia.

Agradeço também ao meu querido irmão, companheiro de todos os momentos da vida.

Agradeço ao meu orientador, Márcio Carvalho e ao corientador, Frederico Gomes por todo o suporte fornecido durante o desenvolvimento deste projeto.

Agradeço aos meus amigos queridos do Nicho, por toda a convivência durante todos os momentos que compartilhamos juntos.

Agradeço por último a todas as pessoas que passaram na minha vida durante a faculdade e que de alguma forma contribuíram para a minha formação, seja profissional ou pessoal.

RESUMO

Desenvolvimento de um software de Simulação de processo de deslocamento de fluidos em anular de poços de petróleo

Durante a exploração de poços de petróleo, especialmente daqueles que se encontram em alto mar (Offshore), existem diversos processos de suma importância para a segurança operacional. Uma das operações mais importantes é a cimentação. A cimentação é o processo de bombeamento de cimento até o poço para que ele expulse o fluido de perfuração do anular e ali se estabeleça. O processo de cimentação tem diversas finalidades, entre elas destacam-se a expulsão do fluido de perfuração do anular do poço, sustentar e proteger o revestimento e ancorar os equipamentos na cabeça do poço. Uma das maiores dificuldades deste processo, é garantir a totalidade e homogeneidade do deslocamento do cimento pelo anular, a fim de expulsar o fluido de perfuração. Este projeto busca construir um software através da linguagem Python, que comunique com a modelagem matemática deste processo de deslocamento de fluidos, realizando o pré e pós-processamento dos dados, facilitando a análise deste procedimento, a fim de otimiza-lo e garantir a eficiência da operação.

Palavras chaves: Cimentação. Deslocamento de fluidos. Software. Petróleo. Poço de Petróleo. Python. Tkinter.

ABSTRACT

Development of a software for Simulation of fluid displacement in the oil well's annular space

Throughout the exploration and development of oil wells, especially those that are offshore, there are many important processes for the operational safety. Cementation is the process of cement displacement into the well, taking out the drilling fluids from the annular space, cementing the well annular. This process has different purposes, among them can be highlighted the displacement of the drilling fluids, support and protect the well covering and anchor the well head equipment. One of the biggest challenges of this process is to assure the totality and homogeneity of the cement displacement through the annular space, aiming to send off the drilling fluid. This project is intended to develop a python-based software, which will communicate with a displacement model, facilitating further analysis, in order to optimize and enhance efficiency of the process, assuring operational safety conditions.

Key-words: Completion. Fluid Displacement. Software. Petroleum. Oil Well. Python. Tkinter.

SUMÁRIO

1 Introdução	8
1.1. Contextualização	8
1.1.1. Cimentação de Poços de Petróleo	8
1.1.2. Modelagem do deslocamento de fluidos	10
1.2. Revisão Bibliográfica e Teórica	11
1.3. Escopo do Trabalho	12
2 Desenvolvimento	13
2.1. Especificações	13
2.1.1. Entradas	14
2.1.2. Saídas	15
2.1.3. Gerais	17
2.2. Implementação	17
2.2.1. Familiarização	17
2.2.2. Engenharia de Software	18
2.2.3. Programação	18
2.3. Exemplo de funcionamento	22
2.3.1. Pré-processamento	22
2.3.2. Pós-processamento	41
3 Conclusão	44
4 Bibliografia	45
ANEXO A	46

Lista de figuras

Figura 1: Exemplo de poço com diversas camadas anulares cimentadas 8
Figura 2: Foto aérea do acidente em Macondo, Golfo do México 9
Figura 3: Logo do Python 11
Figura 4: Ilustração esquemática do funcionamento do software.....13
Figura 5: Exemplo de arquivo de entrada do modelo matemático 14
Figura 6: Exemplo de arquivo de entrada do modelo matemático (2). 15
Figura 7: Exemplo de arquivo de entrada do modelo matemático 15
Figura 8: Exemplo de arquivo de texto da saída do modelo matemático. 16
Figura 9 Exemplo de arquivo de imagem da saída do modelo matemático. 16
Figura 10: Parâmetros do arquivo de entrada do modelo 20
Figura 11: Esquema ilustrativo da 1ª janela do pré-processamento 20
Figura 12 : Janela do software desenvolvido. 23
Figura 13: Exemplo de arquivo de imagem da saída do modelo matemático. 23
Figura 14: Janela do software desenvolvido. 24
Figura 15: Janela do software desenvolvido. 25
Figura 16 : Janela do software desenvolvido. 26
Figura 17: Janela do software desenvolvido. 27
Figura 18 : Janela do software desenvolvido. 28
Figura 19: Janela do software desenvolvido. 29
Figura 20: Janela do software desenvolvido. 30
Figura 21 Janela do software desenvolvido. 31
Figura 22 Janela do software desenvolvido. 32
Figura 23: Janela do software desenvolvido. 33
Figura 24 Janela do software desenvolvido. 34
Figura 25: Janela do software desenvolvido. 35
Figura 26: Janela do software desenvolvido. 36
Figura 27: Janela do software desenvolvido. 37
Figura 28: Janela do software desenvolvido. 37
Figura 29: Janela do software desenvolvido. 38
Figura 30: Janela do software desenvolvido. 38
Figura 31: Janela do software desenvolvido. 39

Figura 32: Janela do software desenvolvido. 39
Figura 33: Janela do software desenvolvido. 40
Figura 34: Janela do software desenvolvido. 41
Figura 35: Janela do software desenvolvido. 41
Figura 36: Janela do software desenvolvido. 42
Figura 37: Janela do software desenvolvido. 42
Figura 38: Janela do software desenvolvido. 43

1 Introdução

1.1. Contextualização

1.1.1. Cimentação de Poços de Petróleo

Durante o processo de desenvolvimento de um campo de petróleo, muitas atividades precisam ocorrer para que o objetivo final do projeto seja alcançado. Inicialmente, após muito estudo, análises e a confirmação de que o petróleo realmente está presente naquele local (campo), a empresa com os direitos exploratórios dessa área começa a campanha de perfuração de poços de petróleo.

A primeira fase dessa operação é realizada através da perfuração com brocas. Uma vez perfurado, seções de revestimento, usualmente tubos de aço levemente menores em diâmetro que a perfuração, são colocados no poço. Com esse revestimento no local, pode-se completar o espaço anular com cimento. Com esta seção cimentada, o poço pode continuar a ser perfurado, seção por seção.

Poços modernos possuem de dois a cinco conjuntos de tamanhos de perfuração subsequentemente menores, com cada espaço anular preenchido de cimento, como exemplificado abaixo:

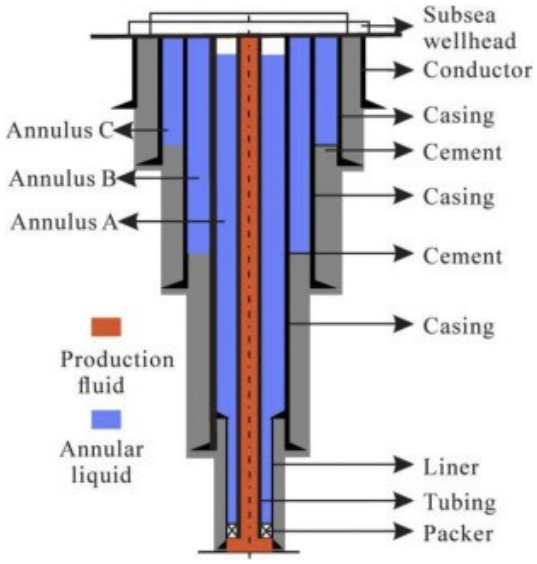


Figura 1: Exemplo de poço com diversas camadas anulares cimentadas

Fonte: Artigo no ScienceDIrect sobre estratégias de mitigação de riscos em poços de petróleo: https://www.sciencedirect.com/science/article/abs/pii/S0920410518307964

A operação de cimentação consiste em um trabalho de extrema importância para a fases de perfuração de poços de petróleo e tem um grande impacto sobre a produtividade do poço.

A cimentação é o processo de bombeamento de cimento até o poço para que ele expulse o fluido de perfuração do anular e ali se estabeleça. O processo de cimentação tem diversas finalidades, entre elas destacam-se a expulsão do fluido de perfuração do anular do poço, sustentar e proteger o revestimento do poço e ancorar os equipamentos na cabeça do poço. Uma das maiores dificuldades deste processo, é garantir a totalidade e homogeneidade do deslocamento do cimento pelo anular, a fim de expulsar o fluido de perfuração de forma homogênea e assim ter sucesso na operação de cimentação.

Um dos acidentes mais conhecidos da indústria de óleo e gás, que ocorreu em Macondo no Golfo do México, evidencia a importância desse processo. A sonda Deepwater Horizon sofreu uma explosão devido a um “blowout” repentino. Após algumas investigações, foi constatado que uma das principais causas deste acidente, foi justamente o fato de durante a cimentação, o espaço anular não ter sido totalmente preenchido com o cimento.

Figura 2: Foto aérea do acidente em Macondo, Golfo do México

Fonte: Matéria do jornal The Telegraph

1.1.2. Modelagem do deslocamento de fluidos

O software que será desenvolvido neste trabalho terá como base o modelo já existente, desenvolvido na tese de doutorado de Frederico Carvalho Gomes [1]. Na literatura, é um dos modelos mais completos e atualizados devido a sua capacidade de abranger diversas condições e situações no processo de deslocamento de fluidos.

Como o processo de deslocamento de fluidos durante a perfuração de poços de petróleo pode ser bastante complexa, envolvendo o bombeamento de diferentes fluidos com diferentes vazões e volumes, o desenvolvimento de um modelo que considera todas as variáveis durante o escoamento no anular é complexo e com custo computacional elevadíssimo. Soma-se à essa complexidade o fato de possuir a presença de líquidos que apresentam comportamento não-newtoniano em um universo 3D e que podem ter seu escoamento transiente em regime laminar e turbulento.

Como já foi adiantado, na literatura existem alguns modelos mais simplificados que usualmente são utilizados em simuladores comerciais para a cimentação. Dentre essas simplificações, principalmente o uso de coordenadas cartesianas para parametrizar o espaço anular, restringe a utilização do modelo por deixar os resultados menos precisos e consequentemente menos confiáveis.

O modelo desenvolvido por Frederico Carvalho Gomes, visa atacar essas fragilidades dos modelos mais simples e com aplicações mais limitadas. Assim, este modelo considera coordenadas cilíndricas para parametrizar o espaço anular, fazendo uso da Teoria de lubrificação. Além disso, ele considera também o comportamento não Newtoniano dos fluidos bombeados, a rotação da coluna de perfuração e tanto regime laminar como turbulento para o escoamento dos fluidos. Dessa forma, permite-se a análise de uma larga faixa de parâmetros de operação, abrangendo o estudo deste processo de forma mais precisa.

1.2. Revisão Bibliográfica e Teórica

Neste projeto, a linguagem que será utilizada para o desenvolvimento do software será Python, uma das linguagens mais utilizadas no planeta quando o assunto é criação de software.

Figura 3: Logo do Python

Fonte: Foto retirada da matéria do site Cafeína codificada

https://cafeinacodificada.com.br/retirar-caracteres-especiais-python/

Python foi criado por Guido Van Rossum no Centro de Matemática e Tecnologia da Informação (CWI, Centrum Wiskunde e informática) em 1989, na Holanda. Foi o sucessor da linguagem de programação ABC, sendo capaz de lidar com exceções e interagir com o sistema operacional usado na época, o Amoeba.

A linguagem foi pensada com a filosofia que enfatiza a importância do programador em relação ao esforço computacional. Prioriza a facilidade de entendimento da interface programador-máquina sobre a eficiência de processamento.

A versão 1.0 foi lançada em janeiro de 1994, com funcionalidades inovadoras para programação funcional, como lamda, map filter e reduce, o que facilitava a vida dos programadores. A última versão enquanto Guido estava na CWI foi o Python 1.2. O Python 3.0 foi lançado em dezembro de 2008, um momento marcante pois houve quebra de compatibilidade com as versões 2.x, para corrigir falhas e aprimorar de vez a linguagem.

Atualmente, Python é componente padrão de diversos sistemas operacionais, entre eles estão a maioria das distribuições do Linux, OpenBSD e OS X. A linguagem se tornou a padrão no curso de ciências da computação do MIT em 2009. Além disso, muitas empresas de grande sucesso utilizam Python em seus projetos. Alguns exemplos que usam muito Python: Google, Youtube, Nasa, Disney, e no Brasil: Magazine Luiza, Locaweb, globo.com.

Python possui uma sintaxe concisa e clara com recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros. Para fins de utilização neste projeto, podemos citar a biblioteca Tkinter.

Tkinter é uma biblioteca de linguagem Python que acompanha a instalação padrão e permite desenvolver interfaces gráficas. Isso significa que qualquer

computador que tenha o interpretador Python instalado é capaz de criar interfaces gráficas usando o Tkinter, com exceção de algumas distribuições Linux, exigindo que seja feita o download do módulo separadamente.

1.3. Escopo do Trabalho

A partir do conteúdo já aqui introduzido, pode-se descrever este projeto de forma mais aprofundada, apresentando a problematização e os objetivos do projeto.

Tendo posse do modelo matemático desenvolvido por Frederico Carvalho Gomes em sua tese de doutorado [1], é possível realizar simulações complexas utilizando o modelo em sua própria linguagem de desenvolvimento, no MATLab. Para realizar essa simulação, é preciso alterar o código fonte do modelo, imputando de forma manual todos os parâmetros necessários.

Dessa forma, vê-se um fator limitante quanto a utilização deste modelo, uma vez que para realizar a simulação, o usuário necessita ter o mínimo de conhecimento de programação da linguagem do modelo, MatLab.

A proposta deste projeto é desenvolver um software que auxilie o modelo aqui citado, a fim de facilitar a interface entre o usuário e entrada de dados/saída dos resultados. Para o desenvolvimento deste software, será utilizada a linguagem Python acompanhada da biblioteca Tkinter, responsável pela criação de interfaces gráficas do usuário (GUI, Graphic User Interface).

O Principal objetivo é disponibilizar um software multiplaforma, através de interfaces intuitivas e interativas a fim de oferecer ao usuário uma ferramenta rápida e eficaz. Também pretende-se estimular os usuários sem conhecimento específico de programação, a utilizar a ferramenta a fim de estudar o processo de cimentação de poços de petróleo de forma integrada.

2 Desenvolvimento

Neste capítulo, serão apresentadas todas as etapas que foram realizadas durante o desenvolvimento do software, através da linguagem selecionada desde a idealização até o real funcionamento do software.

2.1.Especificações

Para o desenvolvimento de um software, independentemente da plataforma, ou ferramenta de programação a ser utilizada, primeiramente é preciso definir todas as características do software em questão, de forma clara, simplificada e que contenha suas funcionalidades, interação com o usuário e o processamento de informações.

No caso deste projeto, como estamos utilizando o modelo assintótico de deslocamento de fluidos, desenvolvido por Frederico Gomes Carvalho [1], certas especificações já estão pré-definidas. Da tese de Doutorado, retira-se todos os parâmetros necessários para a realização da simulação, assim como os parâmetros gerados pelo modelo. Serão esses parâmetros que o software terá que processar, sejam de entradas ou saídas.

Dessa forma, a parte do desenvolvimento, que será direcionada às entradas, será chamada de pré-processamento, enquanto que a parte dedicada a lidar com as saídas do modelo será denominada de pós- processamento. Segue abaixo figura ilustrativa desta abordagem.

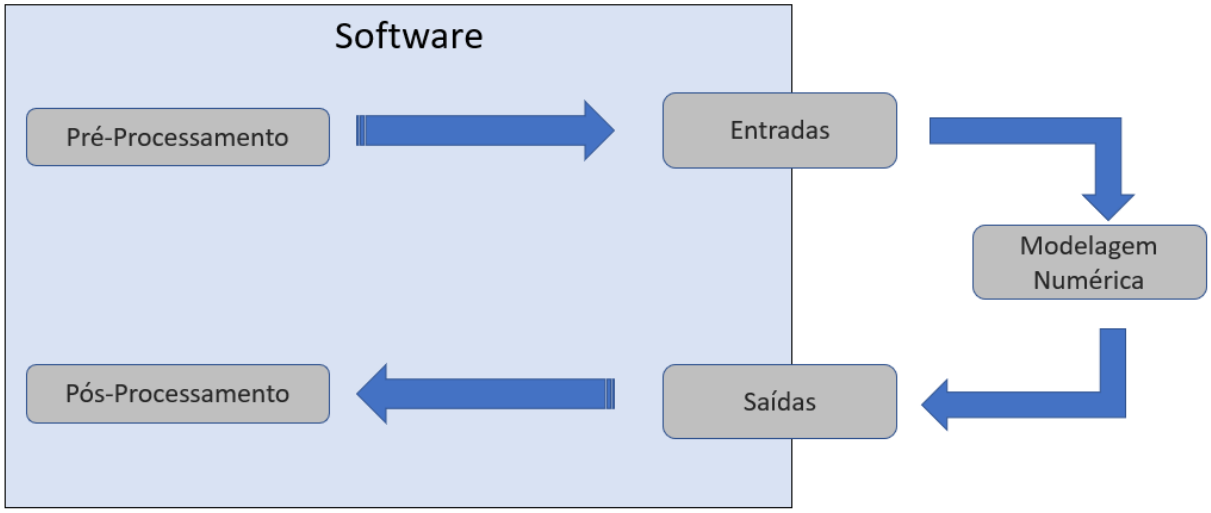


Figura 4: Ilustração esquemática do funcionamento do software

Como em qualquer software, uma das etapas mais importantes é a especificação de suas entradas e saídas. Como o software será construído a fim de utilizar o modelo da tese de Doutorado, as entradas do software a ser construído serão exatamente as mesmas variáveis de entrada para o modelo matemático e terão que estar no mesmo formato no qual as entradas do modelo matemático são definidas. O mesmo acontecerá para as saídas.

2.1.1. Entradas

Como adiantado, as especificações de entrada terão que acompanhar as entradas do modelo matemático. Por isso, analisaremos agora exatamente o que este modelo recebe como entradas.

Estes dados são dispostos em um arquivo no formato PCO, que podem ser acessados por leitores de arquivo texto, como por exemplo o Bloco de Notas. É neste arquivo onde se encontra todas as informações necessárias para a simulação ser realizada. Assim, a especificação a respeito das entradas é explicitada como sendo o fato do software a ser construído ser capaz de gerar este arquivo, no mesmo formato e com todas as informações dispostas da mesma maneira.

Dito isso, este arquivo será visitado com um maior detalhamento. Os parâmetros ali dispostos são divididos em três “categorias”:

- Informações do **modelo**, dispostas no arquivo texto com o prefixo “MODEL”

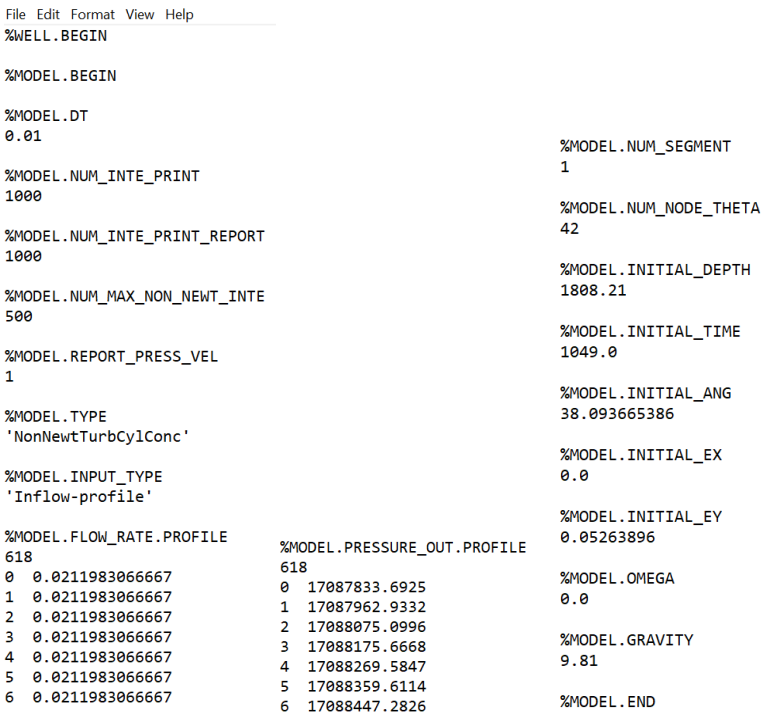


Figura 5: Exemplo de arquivo de entrada do modelo matemático.
Fonte: Fornecido por Frederico Gomes, autor do Modelo

- Informações da geometria do **segmento** a ser cimentado, dispostas no arquivo texto com o prefixo “SEGMENT”

```
%SEGMENT.BEGIN
%SEGMENT.EXTERNAL_DIAMETER
0.220497
%SEGMENT.INTERNAL_DIAMETER
0.0889
%SEGMENT.ANG
32.7689
%SEGMENT.NUM_NODE
17
%SEGMENT.LENGTH
161.79
%SEGMENT.EY
0.052639
%SEGMENT.EX
0
%SEGMENT.ID
1
%SEGMENT.END
```

Figura 6: Exemplo de arquivo de entrada do modelo matemático (2).

Fonte: Fornecido por Frederico Gomes, autor do Modelo

- Informações do(s) **fluido**(s) a ser(em) injetado(s), dispostas no arquivo texto com o prefixo “FLUID”

```
%FLUID.BEGIN
%FLUID.TYPE
'Herschel-Bulkley'
%FLUID.ID
1
%FLUID.RHO
1030.50704
%FLUID.CONSISTENCY
0.001
%FLUID.POWER_LAY_INDEX
1.0
%FLUID.YIELD_STRESS
0.0
%FLUID.VOLUME
0.0
%FLUID.VISCOSITY
0.001
%FLUID.FLOW_RATE
0.01
%FLUID.END
```

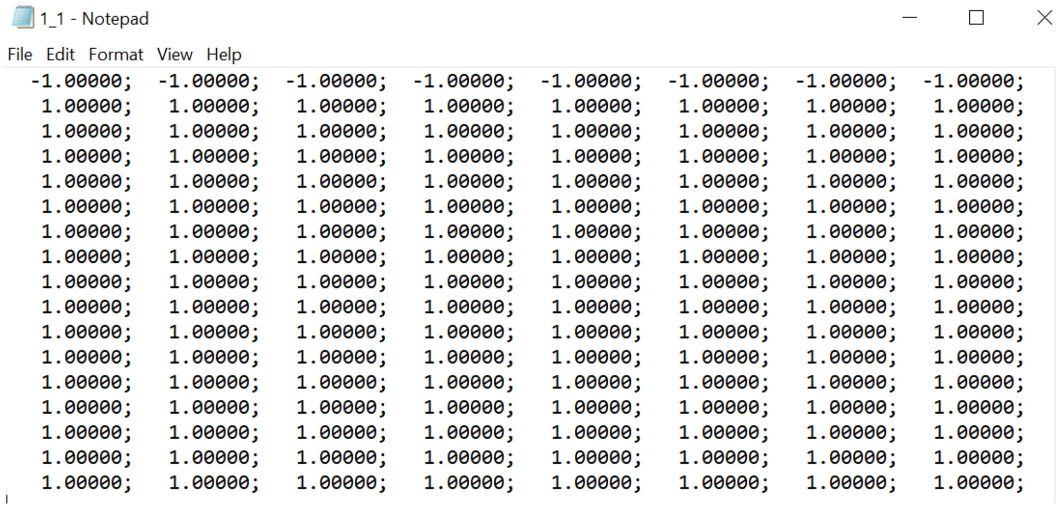
Figura 7: Exemplo de arquivo de entrada do modelo matemático.
Fonte: Fornecido por Frederico Gomes, autor do Modelo

O software terá que obter todas estas informações do usuário e gerar um arquivo idêntico, para ser utilizado pelo modelo matemático.

2.1.2. Saídas

De forma análoga às entradas, as saídas do software também devem estar em total sintonia com as saídas do modelo matemático.

As saídas do modelo matemático são dispostas de maneira diferente às entradas, devido a sua natureza de significado. São arquivos de textos (.PCO) juntamente com um arquivo de imagem (formato .JPG) para cada arquivo de texto. O arquivo de texto possui uma série de valores, separados por vírgula, sem nomes descritos, como pode ser visto na figura X:




```
-1.00000; -1.00000; -1.00000; -1.00000; -1.00000; -1.00000; -1.00000; -1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
 1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;  1.00000;
```

Figura 8: Exemplo de arquivo de texto da saída do modelo matemático.

Fonte: Fornecido por Frederico Gomes, autor do Modelo.

Este arquivo texto é acompanhado de uma imagem de um gráfico descrevendo o deslocamento do fluido no anular do poço em coordenadas cilíndricas, naquele momento específico da simulação, indicado pelo nome do arquivo. Abaixo segue o gráfico correspondente ao arquivo de texto da figura 5, como exemplo, onde cada cor representa o fluido que ocupa cada célula da geometria

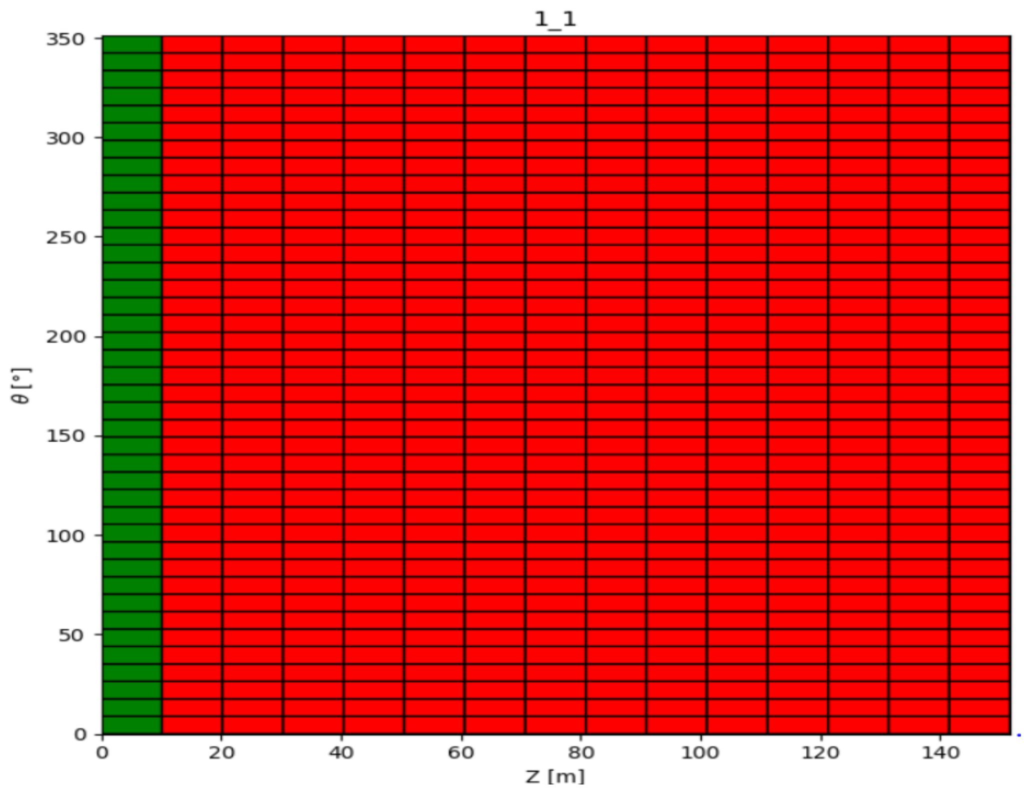


Figura 9 Exemplo de arquivo de imagem da saída do modelo matemático.

Fonte: Fornecido por Frederico Gomes, autor do Modelo.

Assim, define-se as especificações quanto as saídas do software. Ele terá que ser capaz de receber um ou vários arquivos de texto, interpretá-los e conseguir gerar

os gráficos, como demonstrado aqui acima e depois criar um arquivo de imagem para cada gráfico plotado.

2.1.3. Gerais

Após a definição do funcionamento dos parâmetros de entrada e saída do programa, mais alguns requisitos serão detalhados neste capítulo.

Como já citado anteriormente, a ferramenta de programação que será utilizada na programação do software já é um requerimento deste projeto e será a linguagem Python. Dentro das bibliotecas presentes neste ambiente, foi especificado também uma biblioteca do tipo GUI (Graphic User Interface) a ser utilizada, que é o Tkinter.

2.2. Implementação

Com as especificações do projeto definidas, passa-se à parte de implementação do software. Como este projeto requer um entendimento razoável da linguagem Python a fim de atender os requisitos necessários, a primeira etapa foi familiarizar-se com este ambiente.

2.2.1. Familiarização

Para este efeito, foram estudadas todas as esferas básicas e médias dessa linguagem. A documentação disponível para consulta, seja na internet ou em livros é muito vasta. Como esta ferramenta é gratuita, a quantidade de estudos e projetos disponíveis também é relativamente grande.

Começando pela sintaxe do Python, que é uma de suas mais fundamentais características, vê-se alguns atributos importantes como o fato de um bloco de código ser feito através de indentações. Assim, hierarquiza-se o fluxo de execução, fazendo com que estruturas condicionais, funções e loops respeitem a uma execução que é comandada pela indentação. Somado a isso, podemos citar outras características como o baixo uso de caracteres especiais e quase nenhum uso de palavra-chave voltadas para a compilação.

Além da linguagem python, este projeto envolve diretamente a biblioteca Tkinter, já aqui apresentada. Possui a função de criar janelas, abas e botões para lidar com a interação entre o usuário e o modelo/software, o que justamente é o core deste projeto. Também foi preciso um processo de familiarização para assimilar o funcionamento desta ferramenta, antes dela ser utilizada. Foram utilizados diversos sites e alguns cursos, como o da eXcript [3].

2.2.2. Engenharia de Software

A fim de melhorar o gerenciamento e continuidade de um projeto, além de diminuir a possibilidade de erros, a Engenharia de Software entra em cena. Ela está ligada ao desenvolvimento de sistemas complexos dentro do prazo, com qualidade alta e custos viáveis. Aspectos importantíssimos para o sucesso de um projeto. Este desenvolvimento de software não é diferente, mesmo que alguns conceitos não sejam viáveis analisar aqui.

"Engenharia de Software é o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter software de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais", foram as palavras de Fritz Bauer, citado no livro de (PRESSMAN,2011) [4] para definir o conceito de Engenharia de Software.

Trazendo para este projeto, vê-se a necessidade de assimilar esse conceito antes do desenvolvimento prático, uma vez que o objetivo principal é desenvolver um software que seja o mais próximo de um comercial, disponível no mercado. Somado a isso, para termos continuidade no desenvolvimento de outras ferramentas, baseadas no código deste projeto, é essencial que seja bem estruturado, o mais eficiente possível e que tenha uma abordagem sistemática.

2.2.3. Programação

Este capítulo abordará com detalhes todo o processo de desenvolvimento do código de programação base que foi construído para atender as especificações do projeto, aqui já citadas.

Como explicado, o software deste projeto foi dividido em dois componentes, o pré e o pós-processamento. O Primeiro, responsável pelo processamento dos dados de entrada e o segundo responsável pelas saídas. Como o contexto dessas duas partes é muito diferente, a programação será dividida efetivamente em duas partes.

2.2.3.1. Pré-Processamento

Antes de começar a pensar em como seria o arranjo da interface entre usuário e máquina para a entrada de dados, é valido identificar o significado de todas as linhas do arquivo de entrada e saber exatamente de onde vem e o que significam. Para isso, foi criada uma tabela que enumera todos os parâmetros do arquivo, a fim de obter um melhor entendimento das informações.

NOME	SIGNIFICADO	UNIDADE
%WELL.BEGIN	Indica o início dos parâmetros de um Poço para a simulação ser realizada	-

%MODEL.BEGIN	Indica o início dos parâmetros relativos ao modelo (MODEL)	-
%MODEL.DT	Passo de Tempo	Segundos
%MODEL.NUM_INTE_PRINT	Número de Interações a cada armazenamento dos dados de simulação	-
%MODEL.NUM_INTE_PRINT_REPORT	Número de Interações a cada armazenamento dos dados de simulação	-
%MODEL.NUM_MAX_NON_NEWT_INTE	Número máximo de Interações para fluidos não newtonianos	-
%MODEL.REPORT_PRESS_VEL	Intervalo em que cada Report do modelo é printado	Segundos
%MODEL.TYPE	Tipo do Modelo	-
%MODEL.INPUT_TYPE	Tipo das entradas do modelo	-
%MODEL.FLOW_RATE.PROFILE	Perfil de Vazão	m^3
%MODEL.PRESSURE_OUT.PROFILE	Perfil da Pressão de saída	Pa
%MODEL.NUM_SEGMENT	Número de Segmentos da coluna	-
%MODEL.NUM_NODE_THETA	Número de Nós	-
%MODEL.INITIAL_DEPTH	Profundidade Inicial	m
%MODEL.INITIAL_TIME	Tempo Inicial	Segundos
%MODEL.INITIAL_ANG	Ângulo Inicial	Rad
%MODEL.INITIAL_EX	Excentricidade Inicial no eixo x	m
%MODEL.INITIAL_EY	Excentricidade Inicial no eixo y	m
%MODEL.OMEGA	Ômega	°
%MODEL.GRAVITY	Gravidade	m/s^2
%MODEL.END	Indica o fim dos parâmetros relativos ao modelo (MODEL)	-
%SEGMENT.BEGIN	Indica o início dos parâmetros relativos a geometria de um segmento (SEGMENT)	-
%SEGMENT.EXTERNAL_DIAMETER	Diâmetro Externo da coluna	m
%SEGMENT.INTERNAL_DIAMETER	Diâmetro Interno da Coluna	m
%SEGMENT.ANG	Ângulo do Segmento	°
%SEGMENT.NUM_NODE	Número de Nós	-
%SEGMENT.LENGTH	Comprimento do Segmento	m
%SEGMENT.EY	Excentrencidade Final do Segmento no eixo Y	m
%SEGMENT.EX	Excentrencidade Final do Segmento no eixo x	m
%SEGMENT.ID	Número de Identificação do Segmento	-
%SEGMENT.END	Indica o fim dos parâmetros relativos a geometria de um segmento (SEGMENT)	-

%FLUID.BEGIN	Indica o início dos parâmetros relativos a um fluido (FLUID)	-
%FLUID.TYPE	Tipo de Fluido	-
%FLUID.ID	Número de identificação do fluido	-
%FLUID.RHO	Densidade	Kg/m³
%FLUID.CONSISTENCY	Consistência do Fluido	-
%FLUID.POWER_LAY_INDEX	Índice de Potência	-
%FLUID.YIELD_STRESS	Tensão de Escoamento	Pa
%FLUID.VOLUME	Volume	m³
%FLUID.VISCOSITY	Viscosidade	P
%FLUID.FLOW_RATE	Vazão	m³/s
%FLUID.END	Indica o fim dos parâmetros relativos a um fluido (FLUID)	-
%WELL.END	Indica o fim dos parâmetros de um Poço para a simulação ser realizada	-

Figura 10: Parâmetros do arquivo de entrada do modelo

Tendo conhecimento de todas as informações de entrada, pode-se estruturar a forma com que as janelas para a entrada do usuário serão organizadas utilizando todos os recursos que o TKinter possui. Esta biblioteca, tem uma estrutura específica, a qual é baseada em criação de janelas, frames e objetos. Esses objetos, também chamados de widgets, podem ser colocados dentro de janelas e frames. É dessa forma que o TKinter realiza a criação de janelas, abas, botões, entradas de texto, dentre outros diversos elementos.

Sabendo disso, precisava-se de uma estrutura a ser criada, que fosse de fácil entendimento para o usuário e pudesse teoricamente ser construída através dos recursos disponíveis no Tkinter. Assim, chegou-se à seguinte estrutura:

- A primeira janela do pré-processamento a ser aberta teria 4 Botões, como ilustrado abaixo:

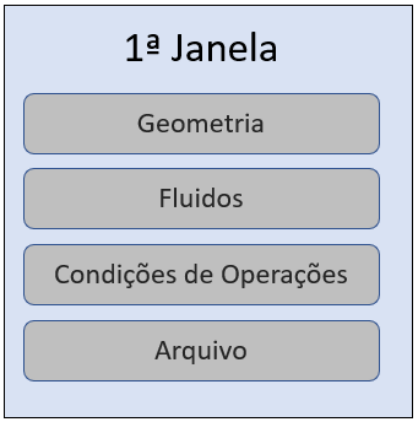


Figura 11: Esquema ilustrativo da 1ª janela do pré-processamento

- O Botão “**Geometria”**, quando pressionado, iria abrir uma nova janela, onde seria lido do usuário todas as informações genéricas dos Segmentos. Com posse dessas informações, incluindo o número de segmentos, ao apertar outro botão nesta segunda janela, uma nova janela iria se abrir para o usuário fornecer os parâmetros específicos para cada Segmento.

- O botão “**Fluidos**”, quando pressionado, iria abrir uma nova janela, onde seria lido do usuário a quantidade de fluidos da simulação. Após o pressionamento de um botão nesta segunda janela, outra seria aberta para receber as informações de cada fluido da simulação.

- O botão “**Condições de Operações**”, quando pressionado, iria abrir uma nova janela, onde seria lido informações gerais do modelo, incluindo o tipo de dados para entrada do modelo. Quando pressionado um botão nessa segunda janela, uma nova se abriria para o perfil dos dados de entrada (pressão de saída e vazão) serem fornecidos pelo usuário.

- O botão “**Arquivo**”, quando pressionado, seria o responsável por armazenar e dispor todos os parâmetros recebidos pelo usuário em um arquivo de texto, para ser usado pelo modelo matemático para realizar a simulação.

Definida a lógica de abertura de janelas, começou o desenvolvimento do programa, a programação em si. Tendo a “engenharia de software” em mente, foi-se criando funções e lógicas, de forma cronológica, visando estar sempre pensando na correta ordem das ações do software. Destaco esse fator porque a linguagem Python e em especial os recursos do Tkinter, são programações orientadas a objetos, sendo vital programar sabendo exatamente a ordem em que as funções/ações precisam ser ativadas. Para detalhes específicos da programação, o código desenvolvido, pode ser consultado no anexo A.

2.2.3.2. Pós-Processamento

No contexto do pós-processamento, como o objetivo é criar gráficos com um “mapa de fluidos” do anular do poço, baseado em um arquivo de texto que será gerado pelo modelo matemático [1], precisa-se em um primeiro momento analisar o arquivo de texto que este software terá que ler a fim de gerar o gráfico mapa de fluido.

O modelo matemático pode gerar vários arquivos de texto para uma mesma modelagem. Os arquivos são gerados de acordo com configurações do usuário, através do parâmetro “NUM_INTE_PRINT_REPORT", que indica o intervalo de passos de tempo para os dados serem gravados, ou seja, indica a cada quantos passos do modelo que será gerado o arquivo de texto. Além disso, cada arquivo possui em seu nome, o número da iteração a qual aqueles dados pertencem. Válido

notar que a primeira e última iterações sempre irão gerar seus respectivos arquivos de texto, independente do valor desse parâmetro.

Entrando nos detalhes de cada arquivo de texto, o número de colunas é relacionado com o número de nós na direção Theta e o número de linhas com o número de nós na direção Z, na direção longitudinal da coluna de perfuração a ser simulada. Além disso, os valores em si representam a pseudo concentração de cada fluido do sistema, seguindo a seguinte regra:

- Fluido 1: +1
- Fluido 2: -1
- Fluido 3: -3
- Fluido 4: -5
- Fluido 5: -7

Devido a difusão, os arquivos podem conter valores intermediários e por isso, a lógica de interpretação para esses arquivos será guiada através dessas regras. Como cada posição no campo de fluidos só pode ser preenchida com uma cor, o valor do arquivo será comparado com os valores da regra, a fim de determinar o mais próximo e assim preencher este ponto com a cor respectiva do fluido com o valor mais próximo.

Assim, a construção do pós-processamento começa abordando a leitura dos arquivos, assumindo que todos estão armazenados em uma mesma pasta. O software abre uma janela para o usuário escolher os arquivos a serem utilizados. Uma vez escolhidos, o software interpreta e armazena os dados do arquivo. Com posse dos dados, o software plota os gráficos para cada conjunto de dados, armazenando-os em arquivos de imagem, localizados na mesma pasta que os outros arquivos, de texto.

Nesta etapa foram utilizadas algumas funções já experimentadas no pré-processamento, além também da biblioteca “matplotlib”, responsável por produzir gráficos do estilo ideal a fim de representar um campo de fluido, como pode ser visto no Anexo A.

2.3. Exemplo de funcionamento

Após o desenvolvimento da programação, tem-se em mãos o software totalmente funcional, sendo responsável pelo pré e pós-processamento dos dados do modelo matemático desenvolvido por Frederico Gomes em sua tese de doutorado [1]. Agora, será demonstrado o funcionamento dele, através de um estudo de caso, fornecido pelo próprio Frederico.

2.3.1. Pré-processamento

Primeiro passo a ser realizado é rodar o software. Assim que executado, a primeira janela a aparecer será a escolha entre o pré-processamento e o pós-processamento, como mostra a figura X1

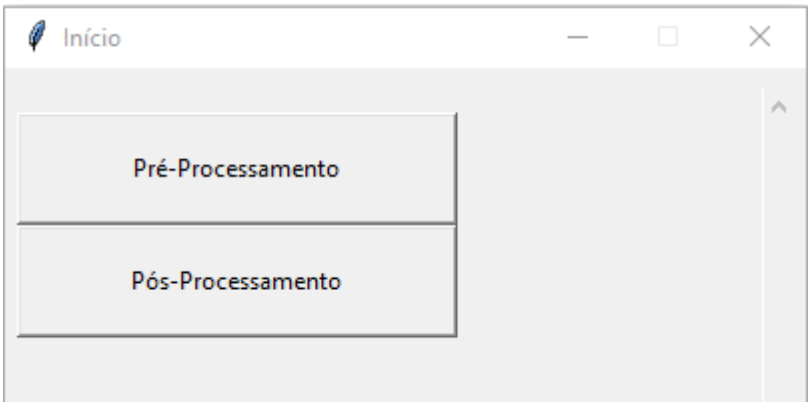


Figura 12 : Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Como estamos na etapa relacionada às entradas, iremos clicar no botão “Pré-Processamento”. Assim que clicado, uma nova janela será aberta com 5 novos botões:

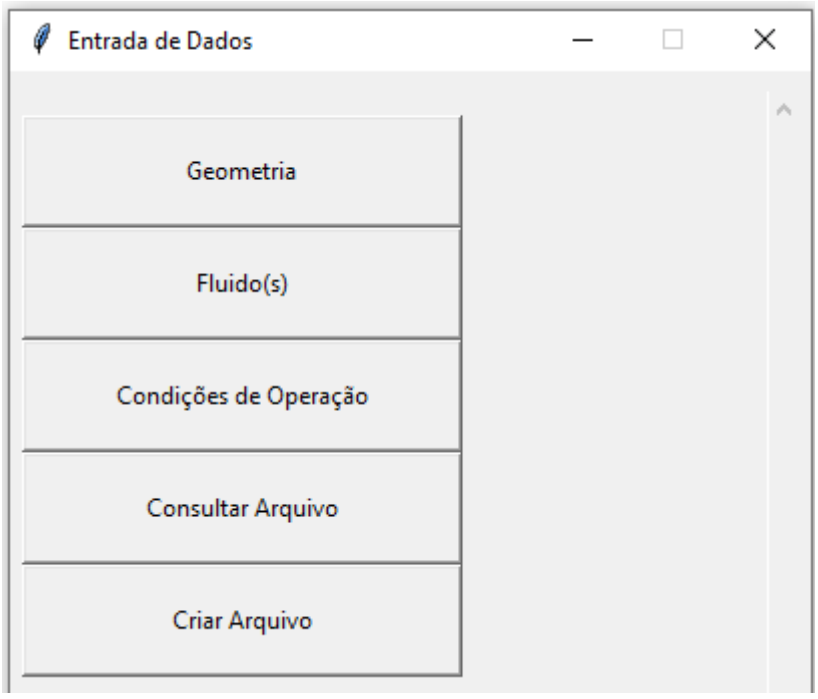


Figura 13: Exemplo de arquivo de imagem da saída do modelo matemático.

Fonte: Software Desenvolvido neste projeto

Esta janela possui os cinco botões, anteriormente descritos, com suas cinco respectivas funções. Como ainda não registramos nenhum dado de entrada, começaremos pela ordem em que estão dispostos. Pressiona-se então, o botão “Geometria”.

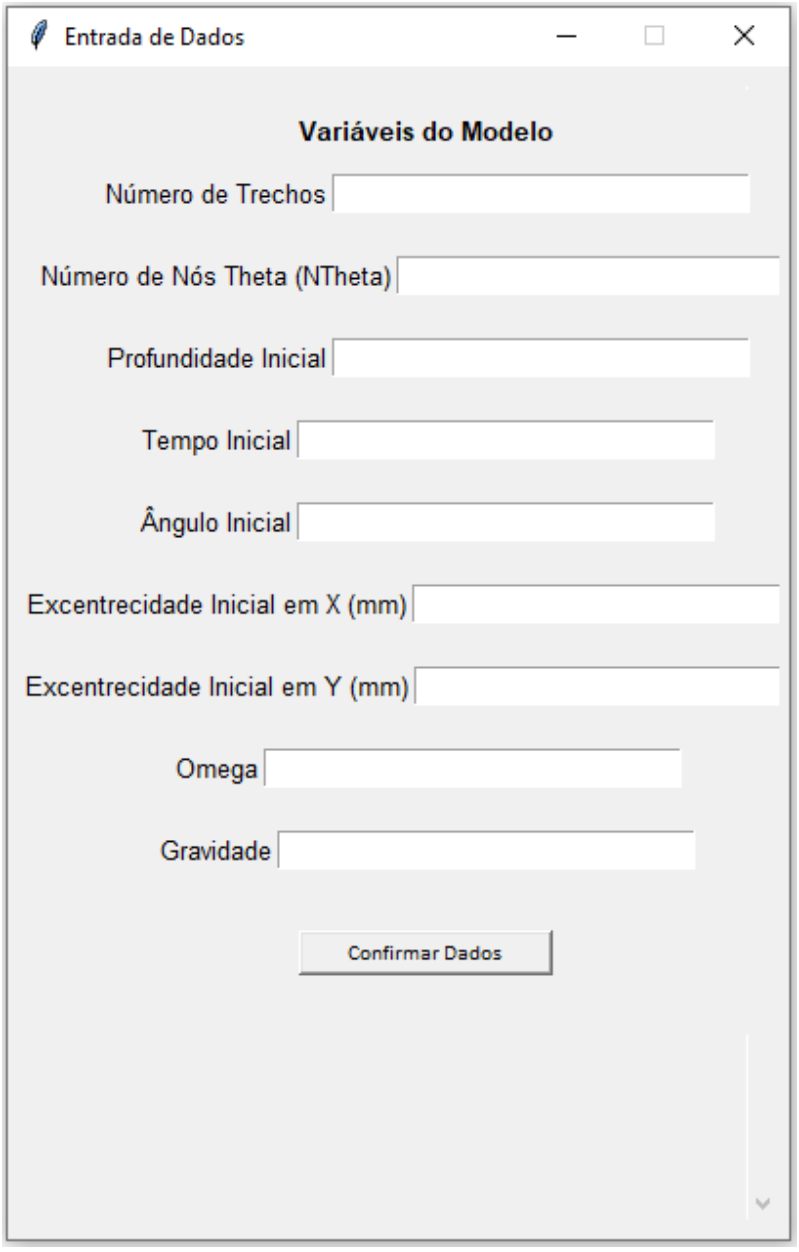


Figura 14: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Ao pressionar o botão geometria, é iniciada a etapa de detalhamento dos parâmetros geométricos da simulação, onde o usuário necessita inserir todas essas características geométricas. A primeira janela ao abrir, inclui parâmetros gerais da geometria além de requisitar o número de trechos que a coluna do poço de petróleo possuirá. Todas essas características são preenchidas:

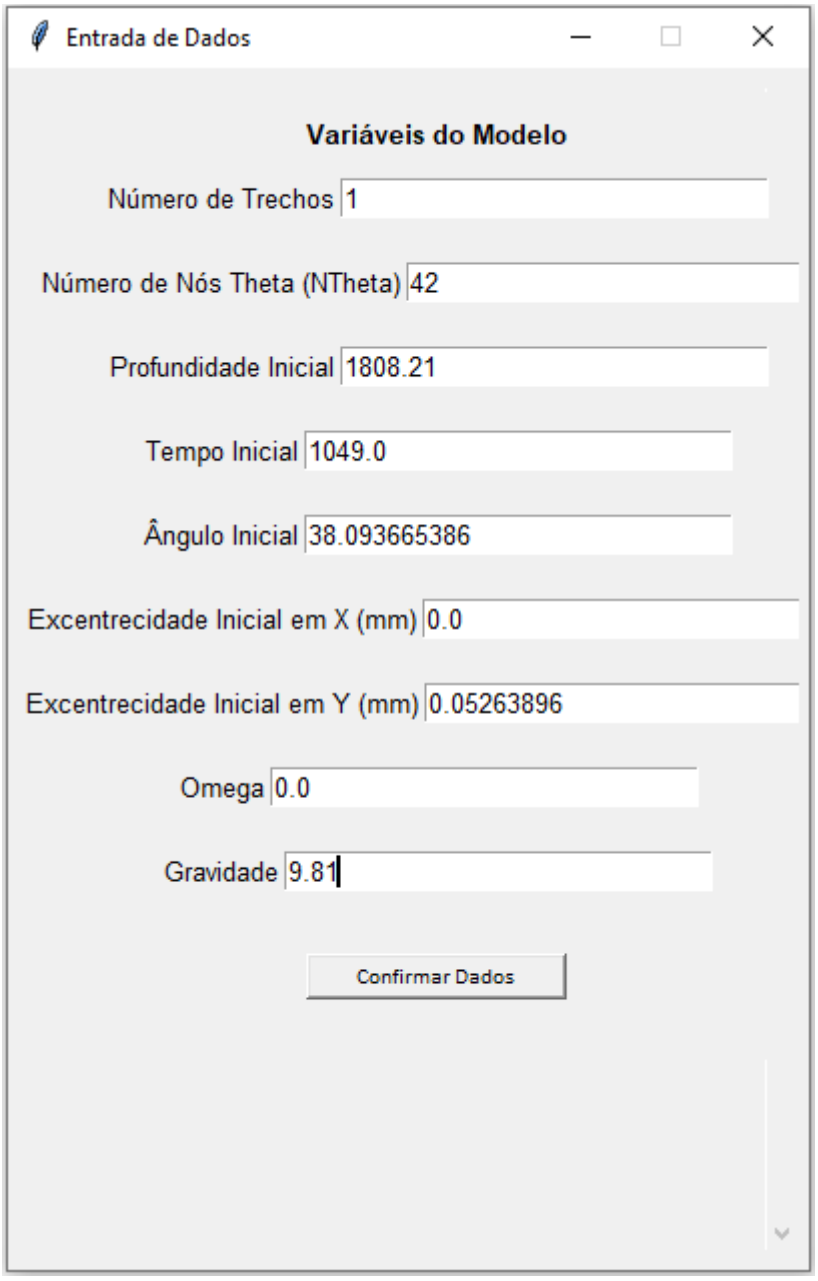


Figura 15: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Após inseridas, as informações são confirmadas ao pressionar o botão “Confirma Dados”. Além disso, abre-se uma nova janela para detalharmos cada trecho da coluna do poço. Como na janela anterior foi informado que seria apenas 1 segmento, as características detalhadas de cada segmento aparecem somente 1 vez:

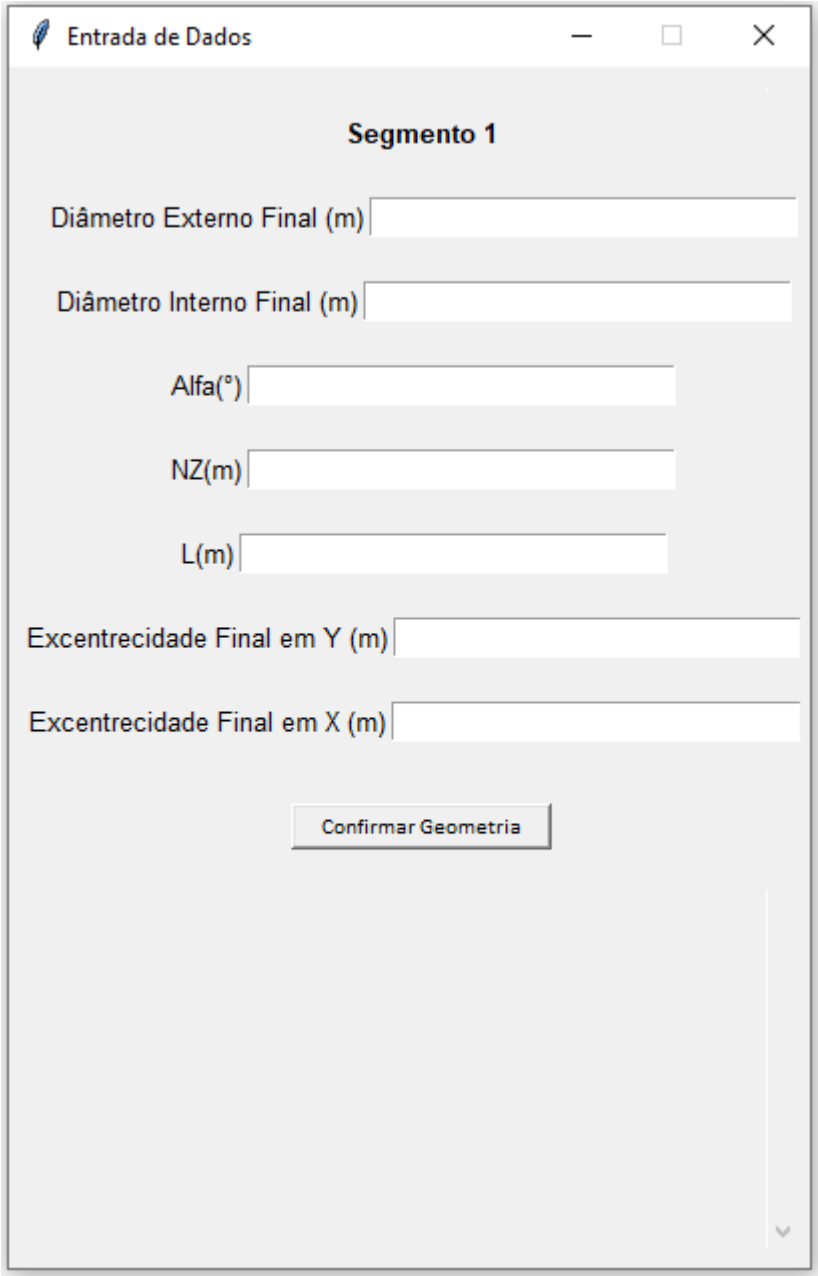


Figura 16 : Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

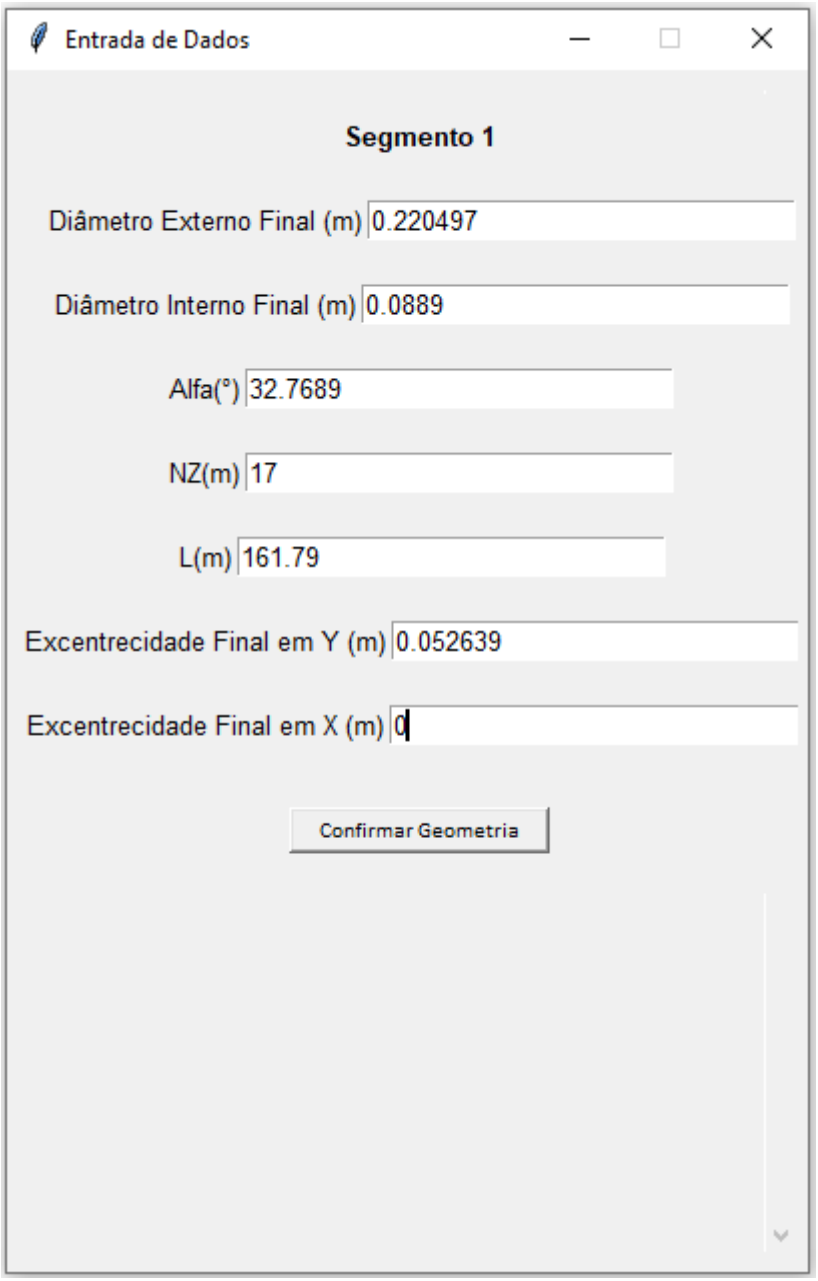


Figura 17: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Ao confirmar a geometria apertando o botão desta janela, uma nova janela ganha vida, confirmando o sucesso do registro de todas as informações sobre a geometria:

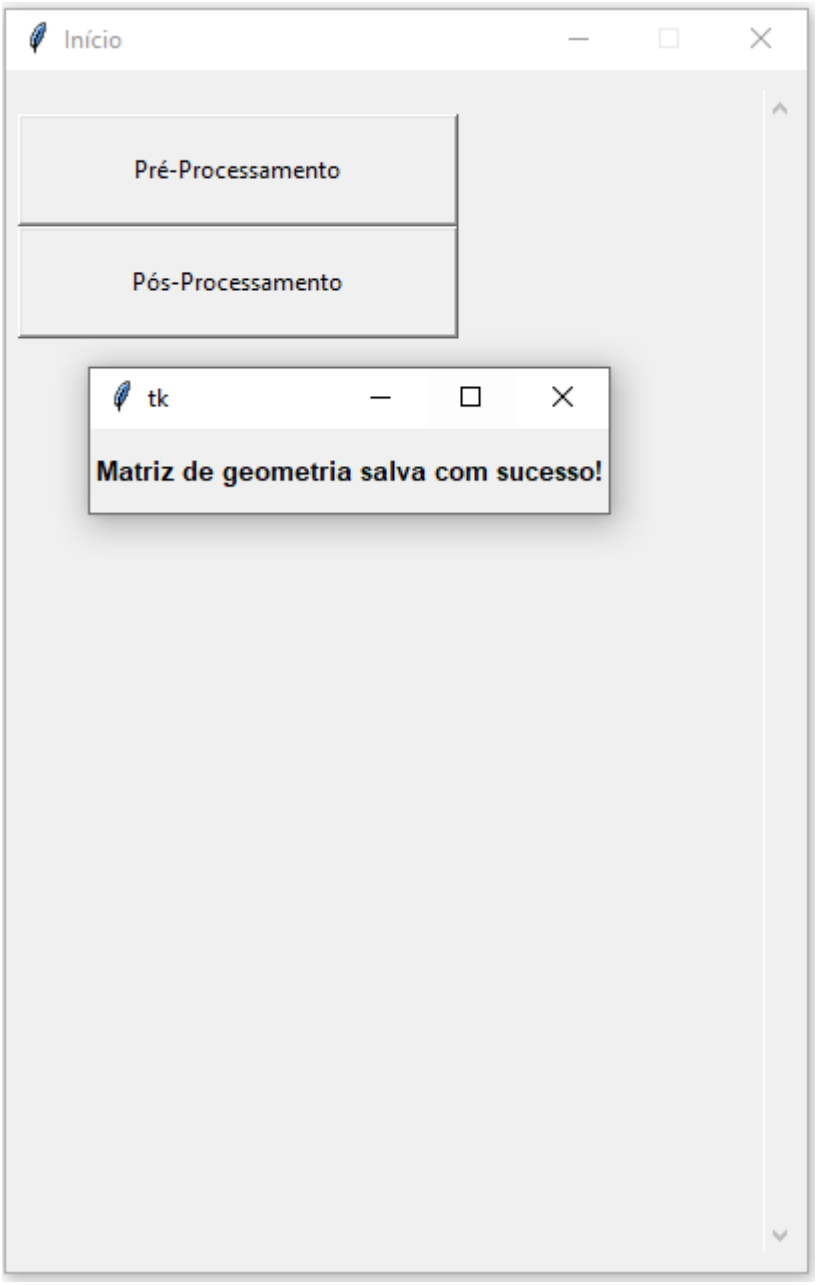


Figura 18 : Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Com isso, volta-se a janela principal. Agora, clicando novamente no pré-processamento, a segunda etapa é detalhar as características do fluido, e para isso, o botão “Fluidos” é pressionado. Pressionando o botão “Fluidos” uma janela é aberta para definir o número de fluidos da simulação. Neste caso definiremos da seguinte forma:

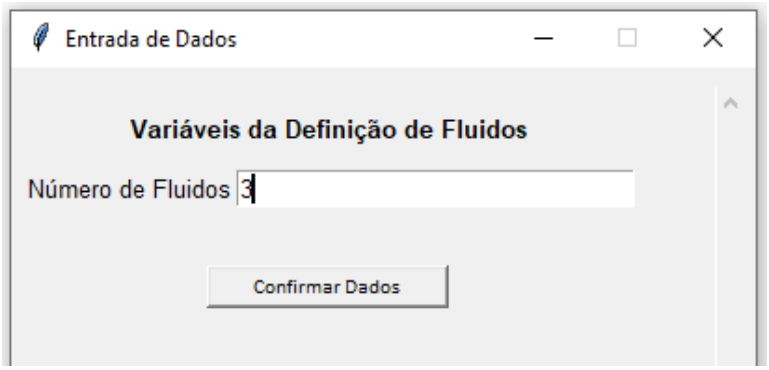


Figura 19: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Com essa informação preenchida, aperta-se o botão “Confirmar Dados “ . E assim, abre-se uma nova janela:

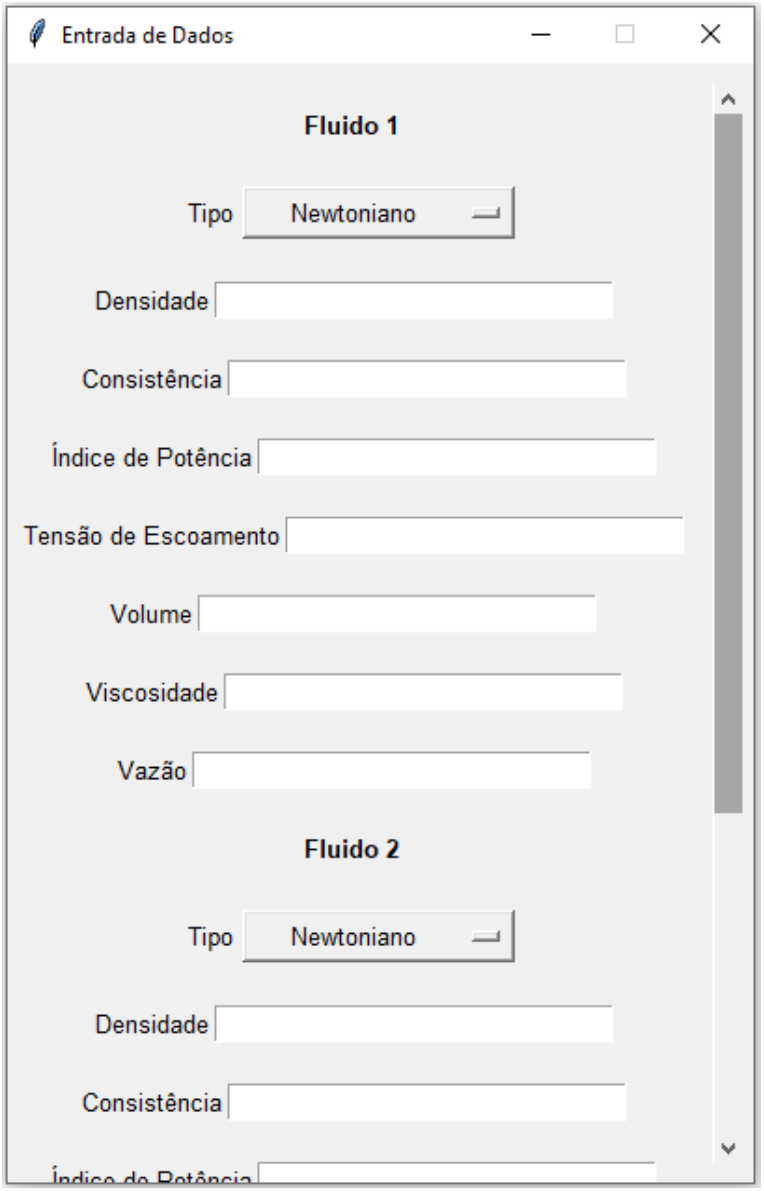


Figura 20: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Esta janela possui os parâmetros a serem definidos para cada fluido da simulação. Neste exemplo, como foi informado que seriam 3 fluidos de perfuração, serão definidos os valores a seguir:

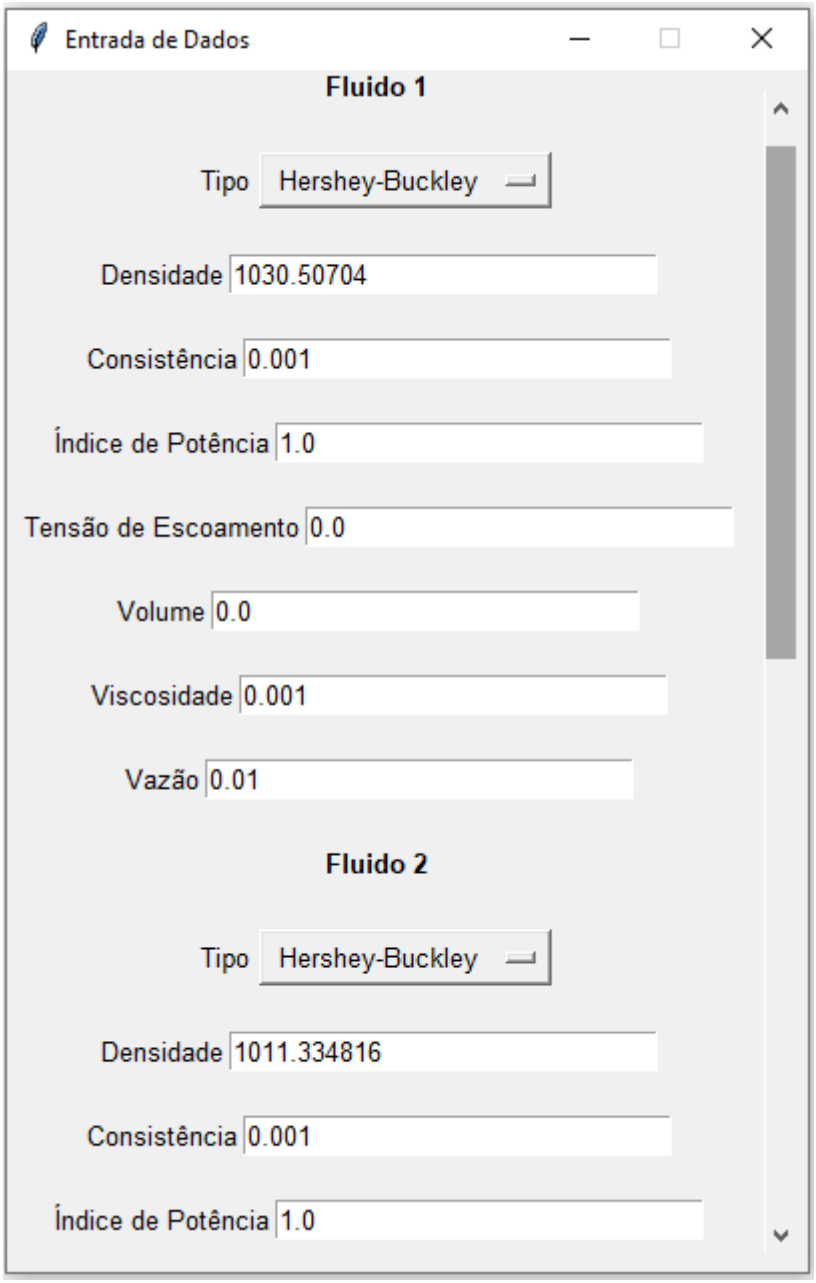


Figura 21 Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Figura 22 Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Uma vez definido todos os parâmetros a respeito dos fluidos, aperta-se no botão desta última janela, e uma nova janela é aberta, confirmando o registro e voltando mais uma vez à janela principal:

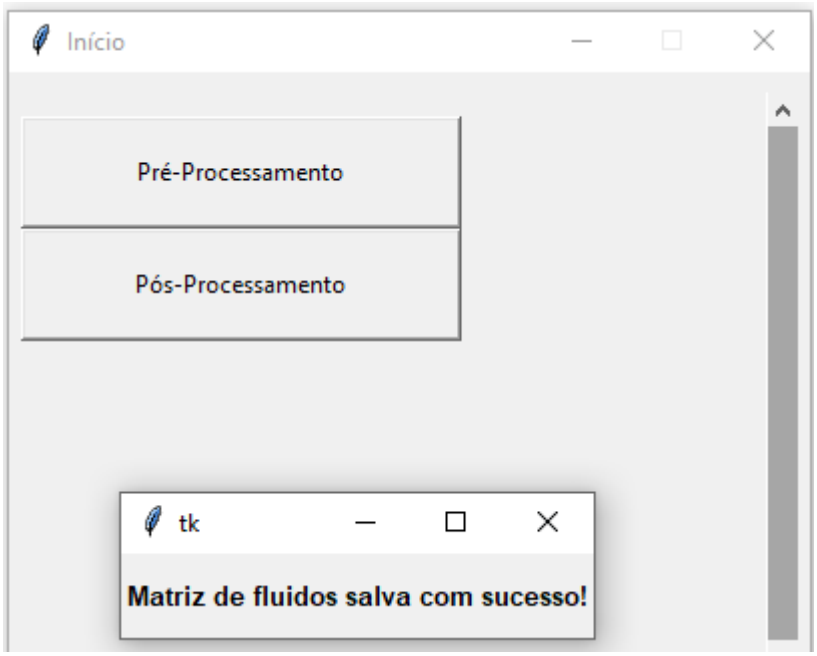


Figura 23: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

A terceira etapa, será iniciada pressionando o botão “Condições de Operação”, onde serão inseridos valores relativos ao modelo.

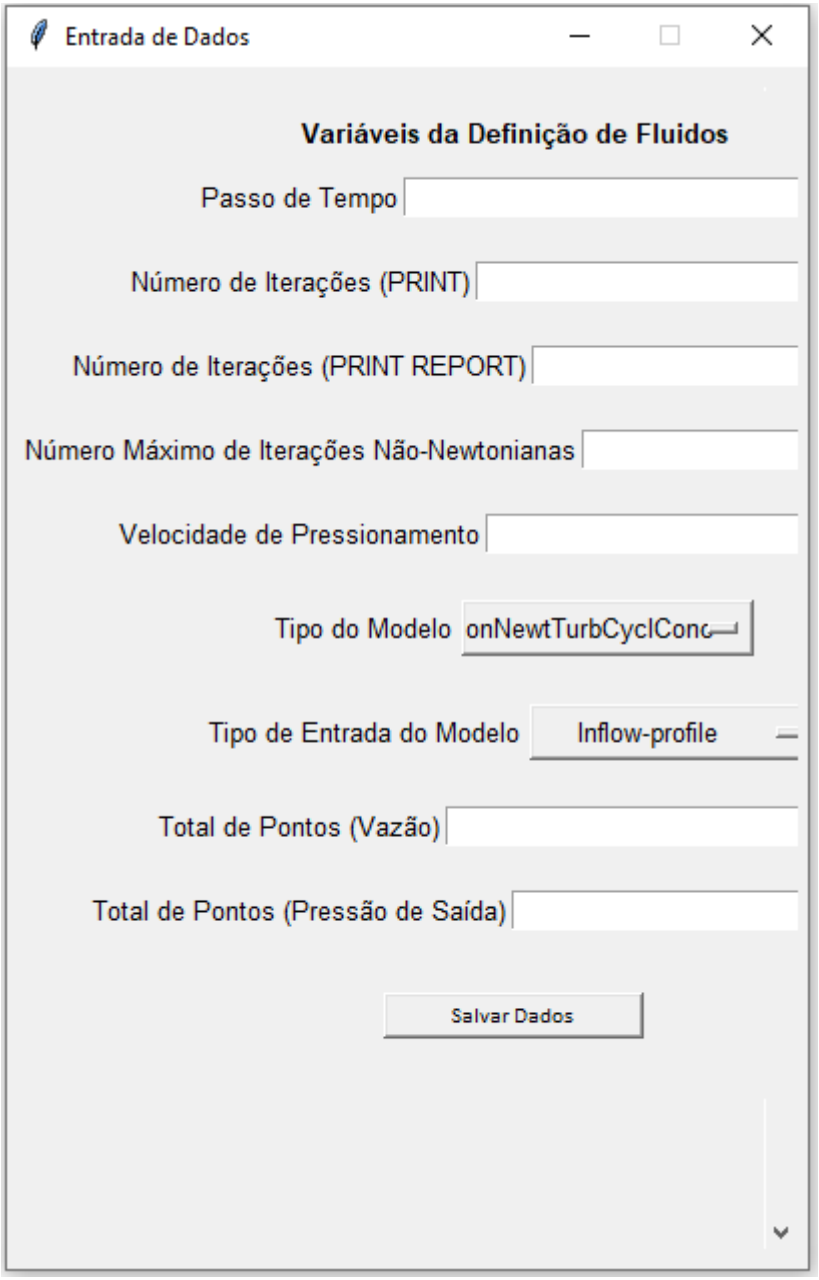


Figura 24 Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

A janela que foi aberta requisita a definição dos parâmetros gerais do modelo. Nessa parte, destaca-se os últimos 4 parâmetros, pois eles serão de suma importância para a abertura da próxima janela, como também para a simulação. Eles que vão definir o perfil e valores das condições de vazão e pressão de saída da simulação, afim de definir condições de contorno para a simulação. Neste caso, será inserido da seguinte forma:

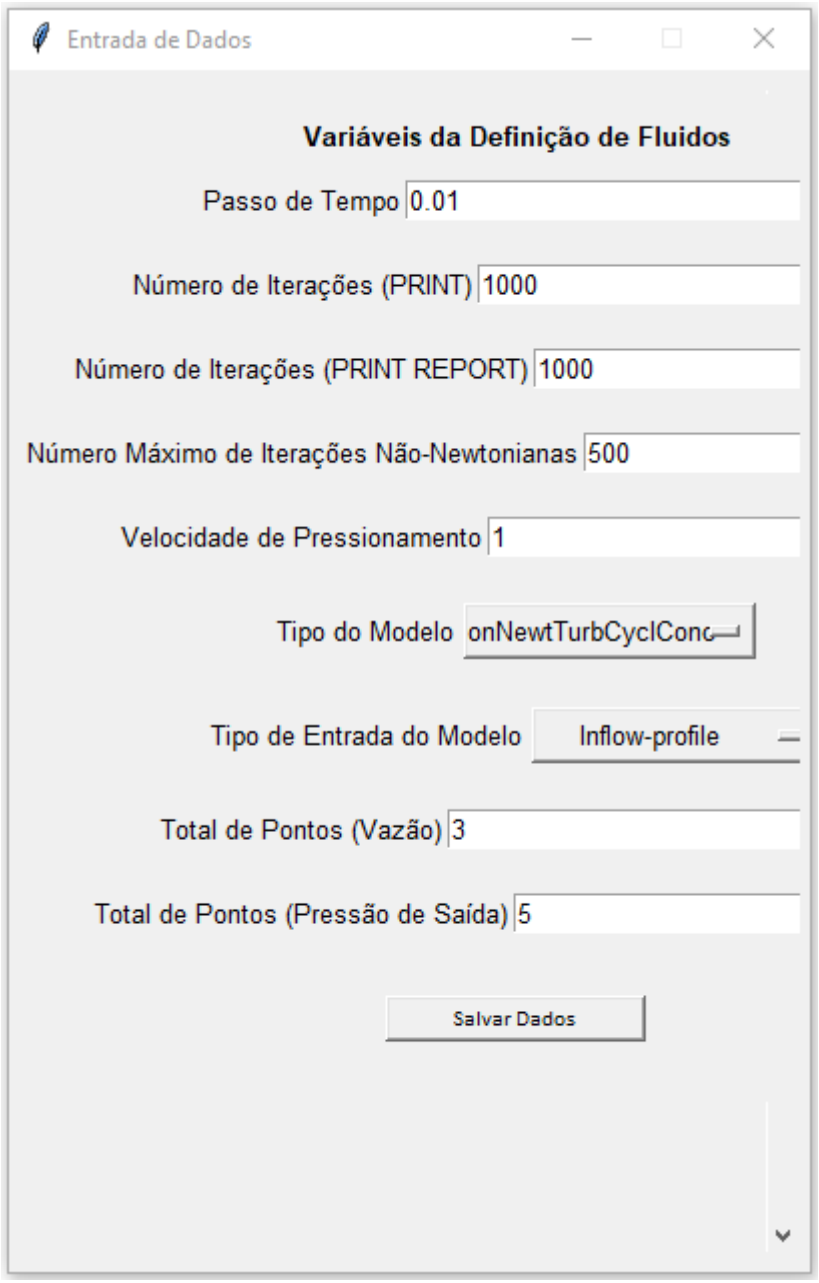


Figura 25: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Ao pressionar o botão da janela acima, confirmando os valores inseridos, uma nova janela é aberta para definir os pontos de mudança de função de cada parâmetro que servirá de entrada para o modelo matemático. Como o software considera as funções como sendo sempre retas, os pontos informados servirão de base para a interpolação dos pontos intermediários. Por isso, pede-se que informe os pontos em que hajam mudança da função que descreve o parâmetro, como apresentado na figura abaixo:

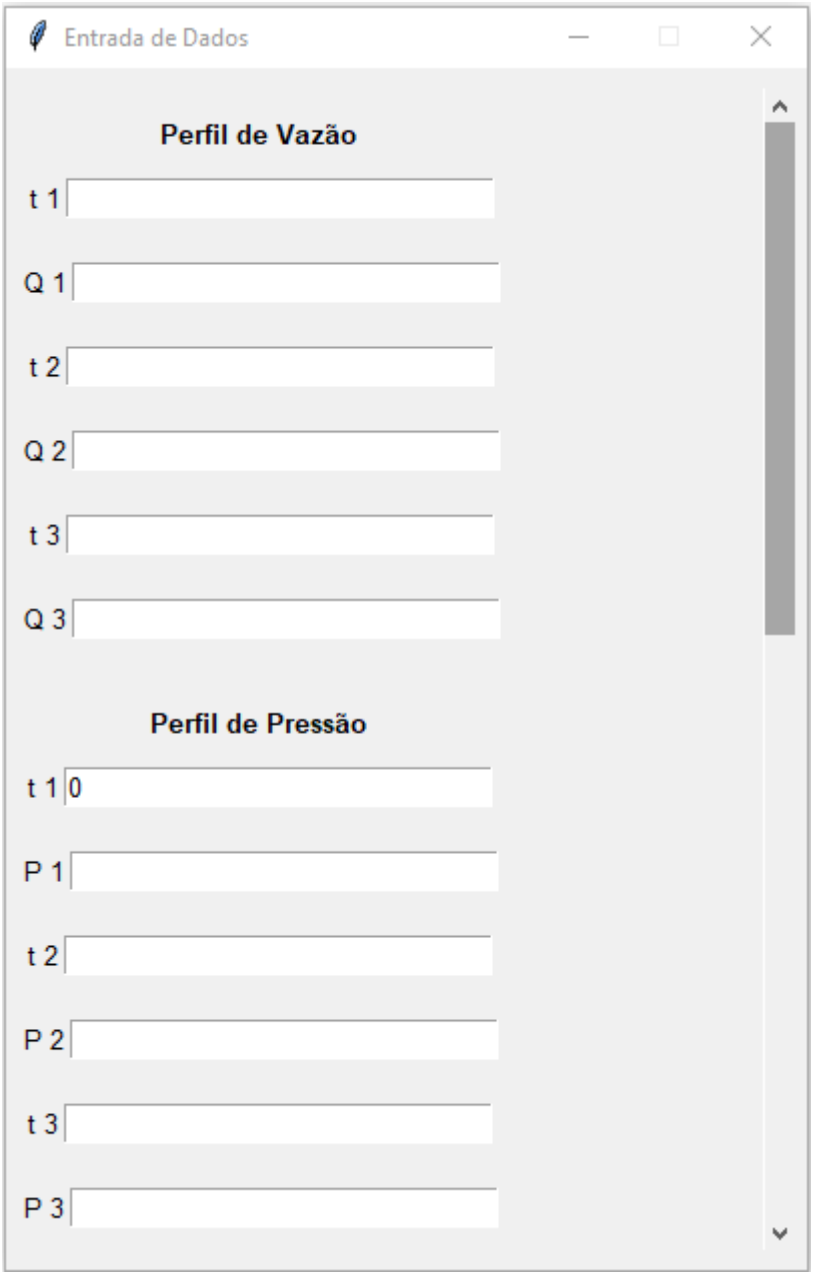


Figura 26: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Figura 27: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Nota: No caso deste exemplo, como o arquivo compartilhado pelo Frederico já é o arquivo final com todos os pontos interpolados, não possuímos os valores exatos destes "pontos de inversão" para este caso. Apesar disso, essa lógica foi testada e verificada, se comportando satisfatoriamente bem em definir perfis de vazão e pressão de saída.

Preenchida a janela, como a figura anterior, pressiona-se o botão "Confirmar dados de Vazão e Pressão" para confirmar o registro de todos parâmetros do modelo e voltar ao menu principal do pré-processamento:

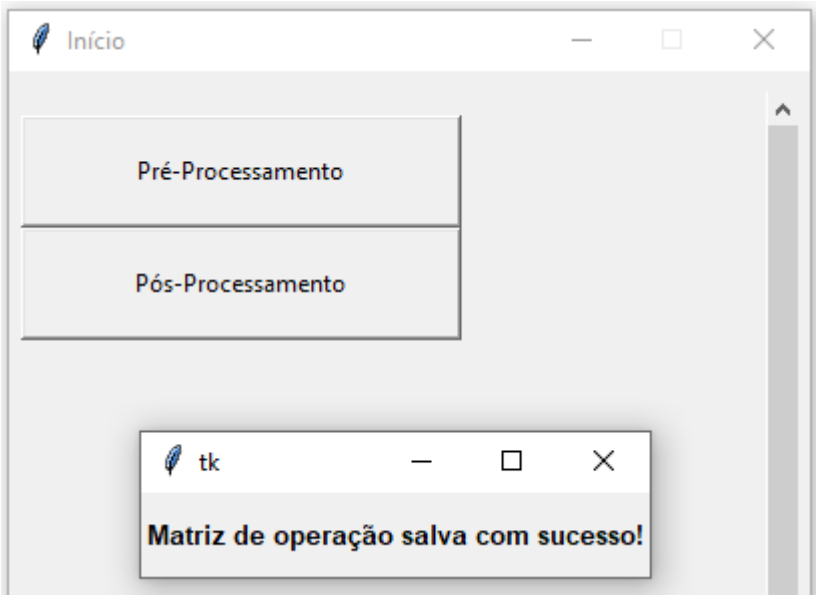


Figura 28: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Agora, que todos as características do modelo foram definidas, para o modelo matemático interpretar da forma correta, os dados precisam estar dispostos de forma específica, como já informado. Para isso, basta apertarmos o 5º e último botão, a fim de gerar um arquivo TXT, que estará pronto para ser lido pelo modelo matemático:

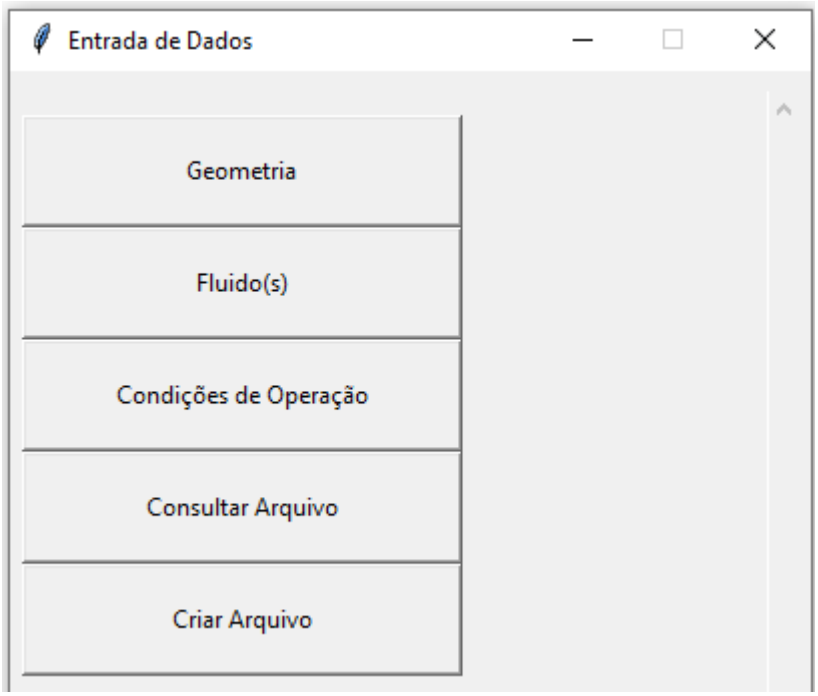


Figura 29: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Ao apertar o botão, abre-se uma janela do explorador de arquivos, para salvarmos este arquivo de entrada.

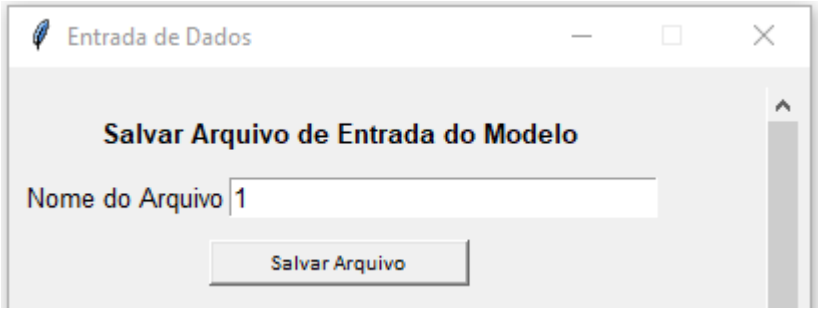


Figura 30: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Assim, encerra-se a parte do pré-processamento do software, quando precisamos gerar um arquivo do zero. Agora, e se em algum caso específico for conveniente acessarmos um arquivo já preenchido, pois precisa-se mudar apenas alguns poucos parâmetros?

Para isso, existe o botão “consultar Arquivo” da janela de pré-processamento:

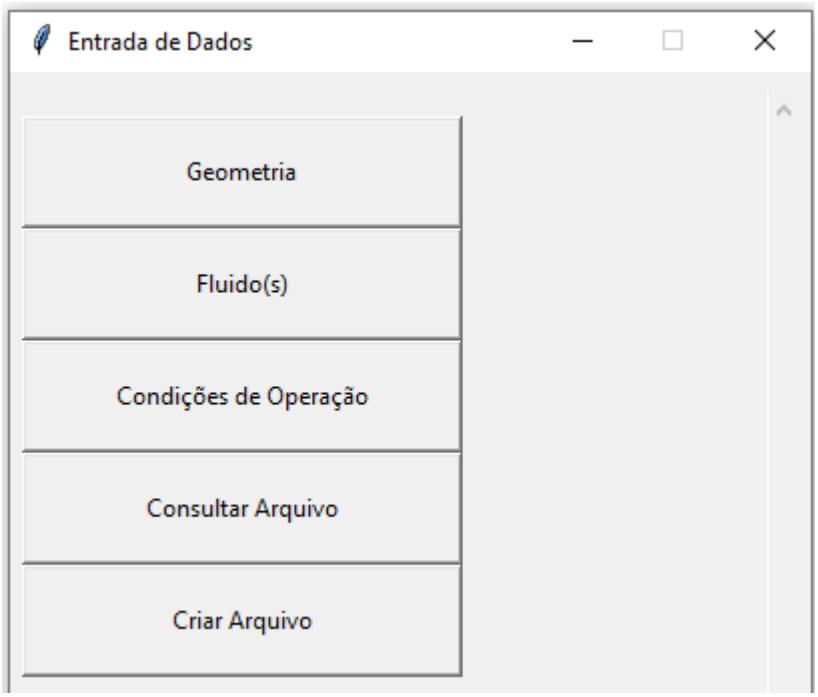


Figura 31: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Ao pressioná-lo, abre-se o explorador de arquivos, para o usuário escolher o arquivo a ser aberto e alterado.

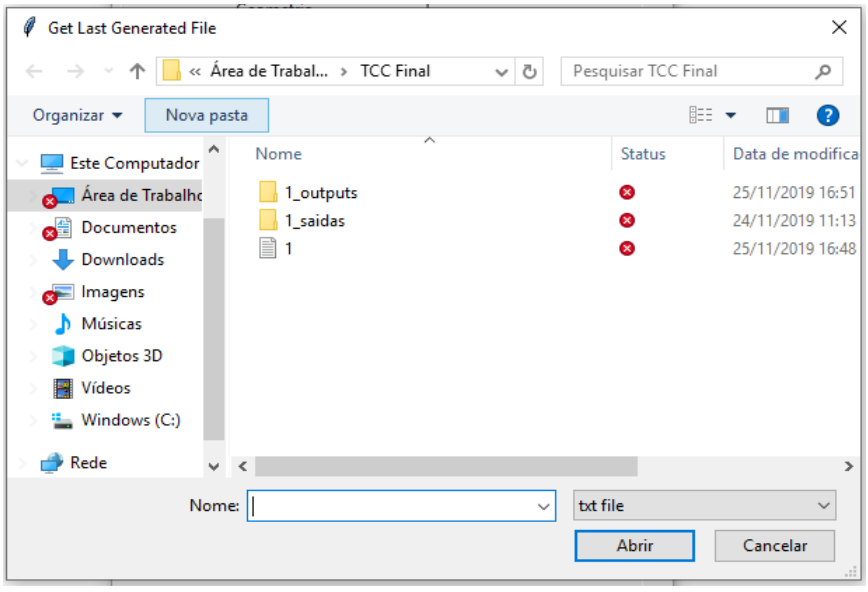


Figura 32: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Ao escolher um arquivo, o programa abre o mesmo, para que manualmente o usuário altere diretamente no arquivo de texto, como é mostrado abaixo:

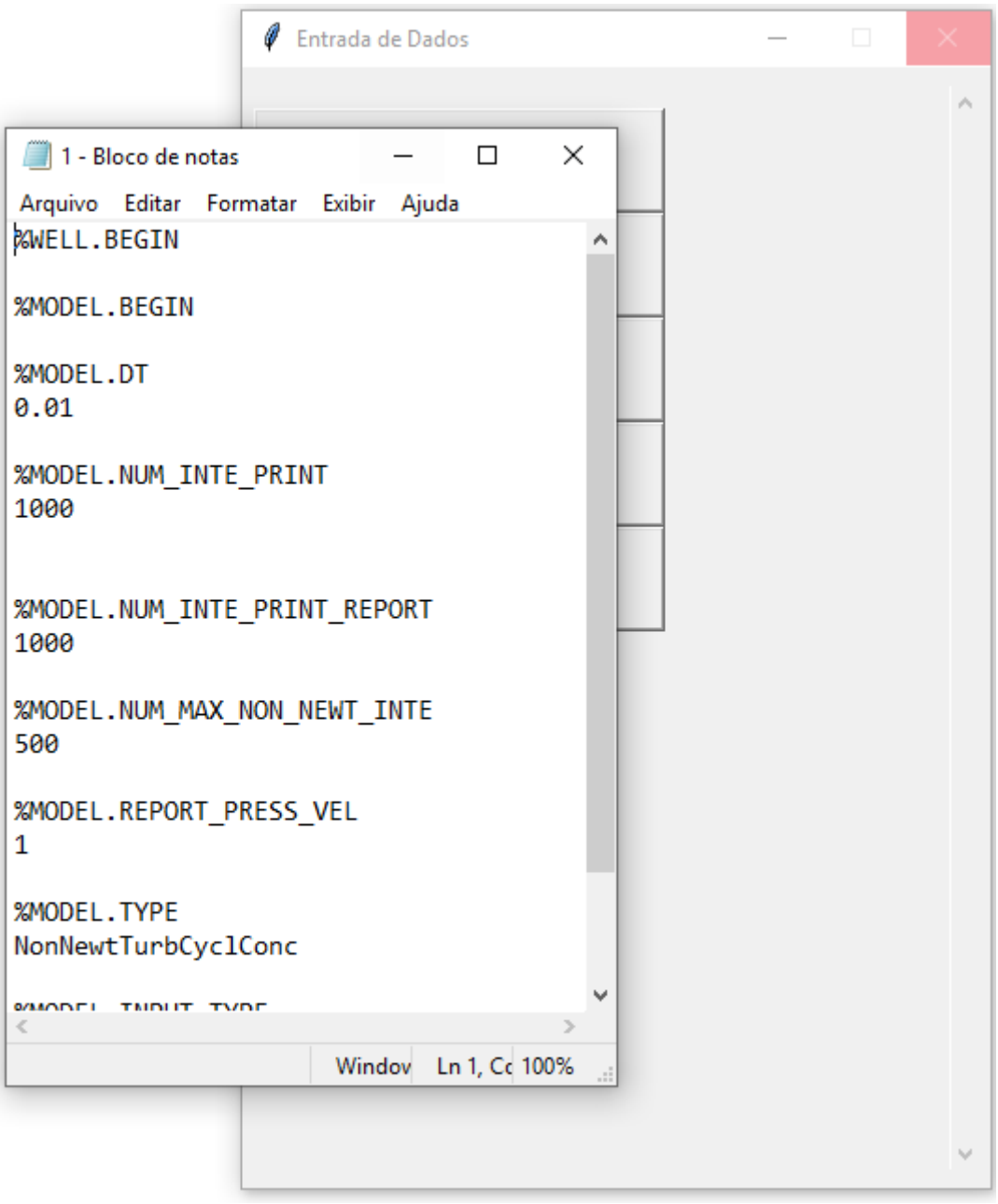


Figura 33: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Assim, o usuário não necessita completar todos os parâmetros novamente, e já possui em mãos o arquivo de entrada para o modelo matemático para realizar a simulação.

2.3.2. Pós-processamento

Para o pós-processamento, o sistema possui algumas peculiaridades. Para o funcionamento completo do sistema, deve-se antes de tudo criar uma pasta e um arquivo na mesma pasta em que se encontra o arquivo do software. São eles:

- Pasta “1_saidas” contendo todos os arquivos do tipo TXT a serem lidos
- Arquivo “1” do tipo PCO, referente ao arquivo de entrada, para o modelo conseguir acessar os parâmetros de entrada

Da mesma forma que o pré-processamento, o primeiro passo é abrirmos o software.

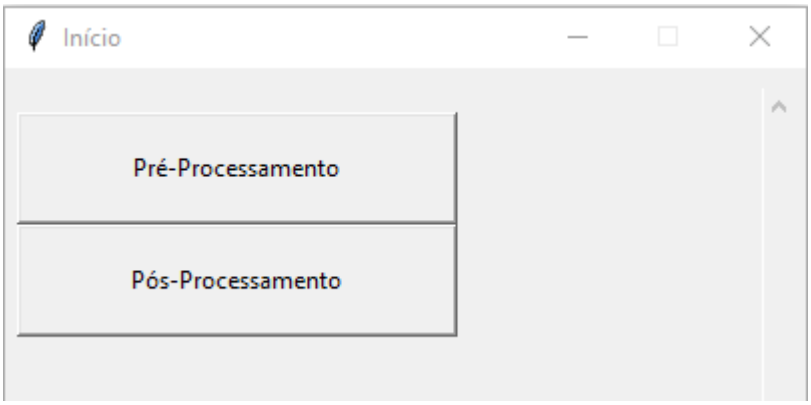


Figura 34: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Como estamos abordando as saídas do modelo, apertaremos no botão “Pós-processamento”:

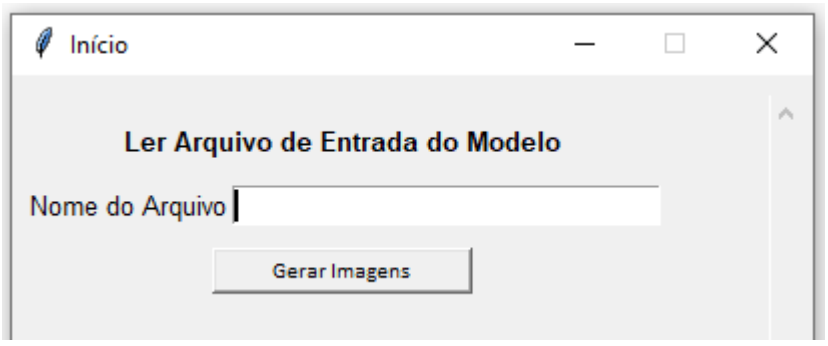


Figura 35: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Agora, temos que colocar o nome do arquivo de entrada referente a simulação que foi feita.

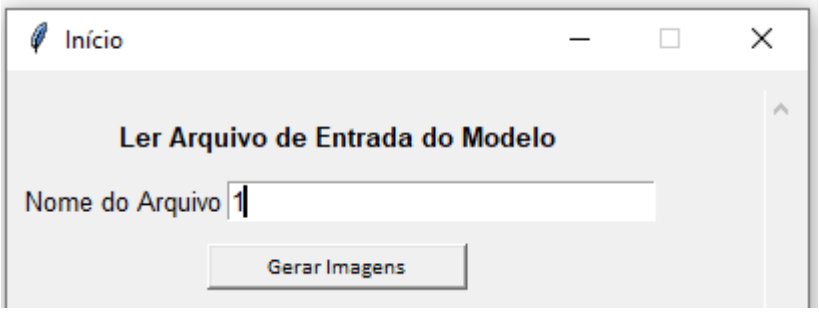


Figura 36: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Neste caso, o arquivo de entradas de chama “1”. Após essa etapa, como todos os arquivos de texto referente às saídas do modelo matemático esão na pasta”1_saidas”, este software automaticamente irá ler todos esses arquivos e armazenar os respectivos gráficos com o mesmo nome, na mesma pasta:

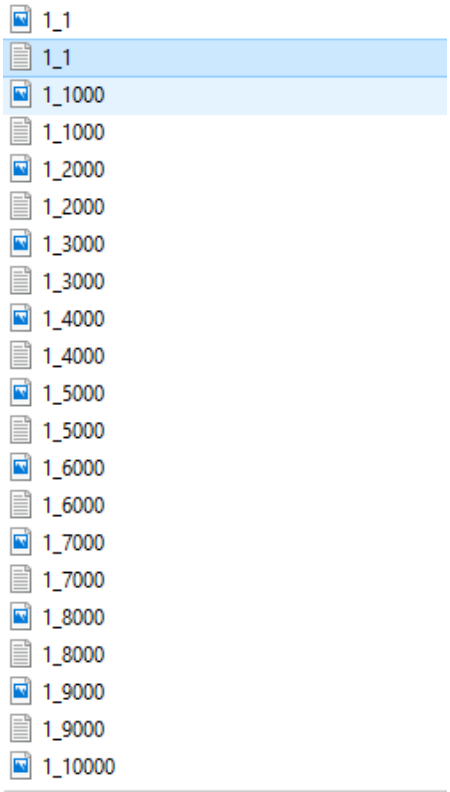


Figura 37: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

Assim, obtemos todos os mapas de fluidos para os respectivos instantes da simulação, vide exemplo do arquivo “1_3000”:

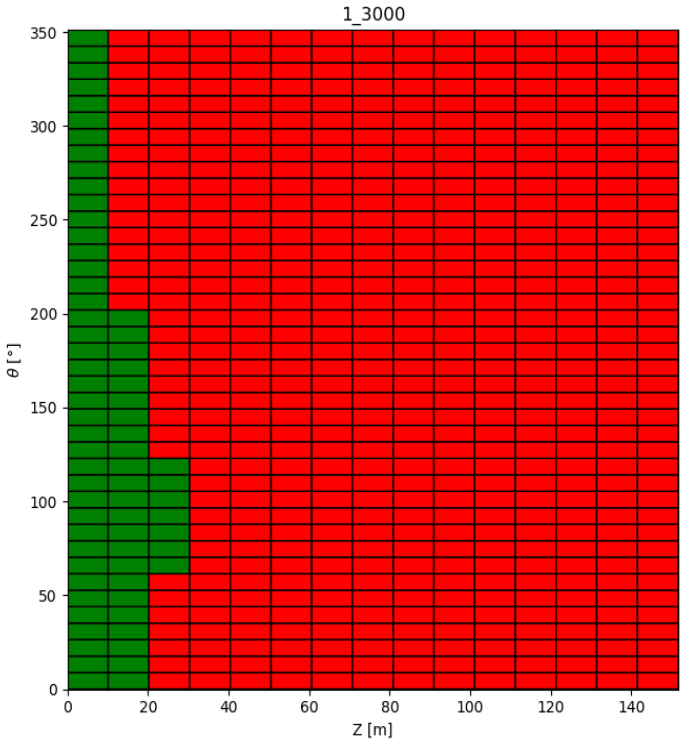


Figura 38: Janela do software desenvolvido.

Fonte: Software Desenvolvido neste projeto

3 Conclusão

O resultado do projeto foi bastante satisfatório, principalmente no pré-processamento, uma vez que o objetivo principal de criar um software para auxiliar o modelo matemático foi alcançado. Foi construído do zero um software para processar os dados de entrada e saída relativos a uma simulação computacional, do deslocamento de fluidos durante o processo de cimentação de um poço de petróleo.

O software desenvolvido inclui diferentes abordagens e funções da linguagem Python, com código organizado e desenvolvido de forma lógica. Fazendo o uso da biblioteca de interface de usuários, Tkinter, foi dado o primeiro e largo passo para transformar um modelo de simulação puramente ligado ao código fonte, em uma interface para o usuário, retirando a necessidade de trabalhar com possíveis alterações no código fonte.

O presente software pode ser melhorado, rescrevendo o código de solução do modelo matemático em Python, não sendo necessário o uso de Matlab para obtenção dos resultados.

Válido ressaltar que a parte do pós-processamento possui ainda alguns pontos a serem otimizados que não foram solucionados. Devido à complexidade e à alta demanda de esforço que o pré-processamento possuía, o pós-processamento não foi completamente otimizado, possuindo alguns recursos faltando, o que compromete o funcionamento em alguns casos.

Durante este projeto muitos desafios foram enfrentados, particularmente durante o processo de escrita do código, a fim de atender inúmeros requisitos da forma mais clara e objetiva possível. Ao final, conseguiu-se construir um software o mais próximo de um comercial, mas que ainda não possui todos os requisitos para tal. Dentre os pontos que evidenciam essa diferença, podemos citar o layout das janelas, uma vez que o software utiliza janelas com aparência básica sem design específico para este software.

O presente software pode ser melhorado, rescrevendo o código de solução do modelo matemático em Python, não sendo necessário o uso de Matlab para obtenção dos resultados.

Por fim, destaco que este foi um importante projeto para conseguir criar um software comercial e que as próximas versões poderão considerar as lições aprendidas deste projeto, além de aproveitar a filosofia dos códigos aqui desenvolvidos.

4 Bibliografia

[1] Gomes,F.C. e Carvalho, M. S. Modelo Assintótico de processo de deslocamento laminar e turbulento de líquidos não newtonianos em anulares. Dissertação de Doutourado, PUC-Rio, 2015.

[2] Fórum Devmedia; Disponível em: <https://www.devmedia.com.br/>

[3] Curso de Pyhton – Módulo Tkinter, eXcript. Disponível em: <https://www.youtube.com/watch?v=kSQDp20eeqE&list=PLesCEcYj003ShHnUT83gQEH6KtG8uysUE&index=1>

[4] PRESSMAN, R. S. Engenharia De Software. [S.l.]: Bookman, 2011.

[5] Raspeberrypi Disponível em:<https://www.raspberrypi.org/>

[6] FÓRUMS Stackoverflow; Disponível em: <https://stackoverflow.com/>

[7] O PETROLEO; Disponível em:< https://www.opetroleo.com.br/>

[8] PyCharm; Computer Program Disponível em: <https://www.jetbrains.com/pycharm/>

[9] Site oficial da linguagem Python. Disponível em <https://www.python.org/>

ANEXO A

```
from tkinter.tix import *
from tkinter.filedialog import askopenfilename
from tkinter import *
from os import listdir, mkdir, system
import matplotlib.pyplot as plt
from matplotlib import colors
import numpy as np
import sys
import platform

# Global variables
curr_page = None
geometry_matrix = []
model_segment_dict = {}

fluid_matrix = []
fluid_dict = {}

operation_matrix = {}
operation_dict = {}

windows_to_close = []

# List to define the labels of each segment variable
segment_labels = [
    "EXTERNAL_DIAMETER",
    "INTERNAL_DIAMETER",
    "ANG",
    "NUM_NODE",
    "LENGTH",
    "EY",
    "EX",
    "ID",
]

# List to define the labels of each model-segment variable
model_segment_labels = [
    "NUM_SEGMENTS",
    "NUM_NODE_THETA",
    "INITIAL_DEPTH",
    "INITIAL_TIME",
    "INITIAL_ANG",
    "INITIAL_EX",
    "INITIAL_EY",
    "OMEGA",
    "GRAVITY",
]
```

```
    # List to define the labels of each model-segment variable
    fluid_labels = [
        "TYPE",
        "ID",
        "RHO",
        "CONSISTENCY",
        "POWER_LAY_INDEX",
        "YIELD_STRESS",
        "VOLUME",
        "VISCOSITY",
        "FLOW_RATE",
    ]

    # List to define the labels of each model-operation variable
    operation_labels = [
        "DT",
        "NUM_INTE_PRINT",
        "NUM_INTE_PRINT_REPORT",
        "NUM_MAX_NON_NEWT_INTE",
        "REPORT_PRESS_VEL",
        "TYPE",
        "INPUT_TYPE",
        "TOTAL_FLOW_RATE",
        "TOTAL_PRESSURE_OUT",
    ]


    model_fluid_labels = [
        "NUM_FLUIDS",
    ]

    ############################################################
    ################### FLUID FUNCTIONS ########################
    ############################################################

    def Definicao_Fluidos (n_fluidos):
        global geometry_matrix

        total = int(n_fluidos.get())
        labels_list = [
            "Tipo",
            "Id",
            "Densidade",
            "Consistência",
            "Índice de Potência",
            "Tensão de Escoamento",
            "Volume",
            "Viscosidade",
```

```
            "Vazão",
        ]

        fluid_types = [
            "Newtoniano",
            "Power-law",
            "Hershey-Buckley",
        ]

        page = Frame(main_frame)

        # Vai ter todas as infos de geometria;
        # Cada linha da matriz representa um trecho,
        # e cada coluna uma info deste trecho
        for index in range(total):
            fontePadrao = ("Arial", "10")
            container = Frame(page)
            container["pady"] = 10
            container.pack()

            titulo_por_fluido = Label(container, text="Fluido %d"%(index+1))
            titulo_por_fluido["font"] = ("Arial", "10", "bold")
            titulo_por_fluido.pack()
            fluid_matrix.append({})

            index2 = 0
            while index2 < len(labels_list):
                label = labels_list[index2]
                fluid_label = fluid_labels[index2]
                if fluid_label == "ID":
                    index2 += 1
                    continue

                entry_container = Frame(page)
                entry_container["pady"] = 10
                entry_container.pack()

                curr_label     =     Label(entry_container,     text=label,
font=fontePadrao)
                curr_label.pack(side=LEFT)
                if fluid_label != "TYPE":
                    fluid_matrix[index][fluid_label]                        =
Entry(entry_container)
                    fluid_matrix[index][fluid_label]["width"] = 30
                    fluid_matrix[index][fluid_label]["font"] = fontePadrao
                    fluid_matrix[index][fluid_label].pack(side=RIGHT)
                else:
                    selected = StringVar(entry_container)
                    selected.set(fluid_types[0])
```

```
                    fluid_matrix[index][fluid_label] = selected

                    dropdown    =    OptionMenu(entry_container,    selected,
*fluid_types)
                    dropdown["width"] = 15
                    dropdown["font"] = fontePadrao
                    dropdown.pack(side=RIGHT)
                index2 += 1


        container_button = Frame(page)
        container_button["pady"] = 20
        container_button.pack()

        button = Button(container_button)
        button["text"] = "Confirmar Fluidos"
        button["font"] = ("Calibri", "8")
        button["width"] = 20
        button["command"] = lambda: Fim_Fluido(fluid_matrix)
        button.pack()

        ChangePageTo(page)
        CanvasReconfigure()

    def Fim_Fluido(matrix):
        # Converting matrix content from Entry to integer (int)
        for i in range(len(matrix)):
            line = matrix[i]
            for key, elem in line.items():
                matrix[i][key] = elem.get()

        OpenNewMessageWindow("Matriz de fluidos salva com sucesso!")

        page1 = FirstPage()
        ChangePageTo(page1)
        CanvasReconfigure()

        new_window.mainloop()

    ############################################################
    ################## GEOMETRY FUNCTIONS ######################
    ############################################################

    def Definicao_Trechos_Geometria (n_Teta):
        global geometry_matrix
        global windows_to_close
        global model_segment_dict

        # Update dict before frame is destroyed
```

```
            for key, elem in model_segment_dict.items():
                model_segment_dict[key] = elem.get()

            total = int(n_Teta.get())
            labels_list = [
                "Diâmetro Externo Final (mm)",
                "Diâmetro Interno Final (m)",
                "Alfa(°)",
                "NZ(mm)",
                "L(mm)",
                "Excentrecidade Final em Y (mm)",
                "Excentrecidade Final em X (mm)",
            ]

            container = Frame(main_frame)

            # Vai ter todas as infos de geometria;
            # Cada linha da matriz representa um trecho,
            # e cada coluna uma info deste trecho
            for index in range(total):
                fontePadrao = ("Arial", "10")
                Container_Trecho_x = Frame(container)
                Container_Trecho_x["pady"] = 10
                Container_Trecho_x.pack()

                titulo_por_trecho   =   Label(Container_Trecho_x,   text="Segmento
%d"%(index+1))
                titulo_por_trecho["font"] = ("Arial", "10", "bold")
                titulo_por_trecho.pack()
                geometry_matrix.append({})

                for index2 in range(len(labels_list)):
                    label = labels_list[index2]
                    segment_label = segment_labels[index2]
                    entry_container = Frame(container)
                    entry_container["pady"] = 10
                    entry_container.pack()

                    Label_Trecho_x    =    Label(entry_container,    text=label,
font=fontePadrao)
                    Label_Trecho_x.pack(side=LEFT)
                    geometry_matrix[index][segment_label]                      =
Entry(entry_container)
                    geometry_matrix[index][segment_label]["width"] = 30
                    geometry_matrix[index][segment_label]["font"] = fontePadrao
                    geometry_matrix[index][segment_label].pack(side=RIGHT)

            Container_button = Frame(container)
            Container_button["pady"] = 20
```

```
            Container_button.pack()

            Butao_Geometria = Button(Container_button)
            Butao_Geometria["text"] = "Confirmar Geometria"
            Butao_Geometria["font"] = ("Calibri", "8")
            Butao_Geometria["width"] = 20
            Butao_Geometria["command"] = lambda: Fim_Geometria(geometry_matrix)
            Butao_Geometria.pack()

            ChangePageTo(container)
            CanvasReconfigure()

        def Fim_Geometria(matrix):

            # Converting matrix content from Entry to integer (int)
            for i in range(len(matrix)):
                line = matrix[i]
                for key, elem in line.items():
                    matrix[i][key] = elem.get()

            OpenNewMessageWindow("Matriz de geometria salva com sucesso!")

            page1 = FirstPage()
            ChangePageTo(page1)
            CanvasReconfigure()

            new_window.mainloop()


        ############################################################
        ################ OPERATION FUNCTIONS ######################
        ############################################################

        def Definicao_Perfil_Vazao_Pressao(num_points_flow, num_points_press):
            global operation_matrix
            global operation_dict

            # Dict which contains num_points_flow and num_points_press,
respectively
            num_points_flow = int(num_points_flow.get())
            num_points_press = int(num_points_press.get())
            profiles_dict = {
                "Perfil de Vazão": num_points_flow,
                "Perfil de Pressão": num_points_press,
            }
            coord_labels = {
                "FLOW_RATE": ["t", "Q"],
                "PRESSURE_OUT": ["t", "P"],
            }
```

```
        page = Frame(main_frame)

        for key, elem in operation_dict.items():
            operation_dict[key] = elem.get()

        count = 0
        for title, total in profiles_dict.items():
            fontePadrao = ("Arial", "10")
            container = Frame(page)
            container["pady"] = 10
            container.pack()

            profile_title = Label(container, text=title)
            profile_title["font"] = ("Arial", "10", "bold")
            profile_title.pack()

            dict_key = list(coord_labels.keys())[count]
            labels_list = coord_labels[dict_key]
            matrix_aux = operation_matrix.get(dict_key, [])

            for i in range(total):
                matrix_aux.append({})
                for label in labels_list:
                    dict_elem = matrix_aux[i].get(label, None)
                    entry_container = Frame(container)
                    entry_container["pady"] = 10
                    entry_container.pack()

                    curr_label    =    Label(entry_container,    text=label+"
"+str(i+1), font=fontePadrao)
                    curr_label.pack(side=LEFT)
                    dict_elem = Entry(entry_container)
                    dict_elem["width"] = 30
                    dict_elem["font"] = fontePadrao
                    dict_elem.pack(side=RIGHT)
                    matrix_aux[i][label] = dict_elem
            operation_matrix[dict_key] = matrix_aux
            count += 1
        #print(operation_matrix)

        container_button = Frame(page)
        container_button["pady"] = 20
        container_button.pack()

        button = Button(container_button)
        button["text"] = "Salvar Dados de Vazão e Pressão"
        button["font"] = ("Calibri", "8")
        button["width"] = 20
```

```
            button["command"] = lambda: Fim_Operacao(operation_matrix)
            button.pack()

            ChangePageTo(page)
            CanvasReconfigure()

        def Fim_Operacao(op_matrix):
            # Converting dict content from Entry to integer (int)
            for profile_type, profile_list in op_matrix.items():
                for i in range(len(profile_list)):
                    curr_dict = profile_list[i]
                    for axis_label, entry in curr_dict.items():
                        op_matrix[profile_type][i][axis_label]                  =
int(entry.get())
            #print(op_matrix)

            OpenNewMessageWindow("Matriz de operação salva com sucesso!")

            BackToFirstPage()

        ############################################################
        ################## BUTTON FUNCTIONS ########################
        ############################################################

        def Entrada_Geometria():
            global model_segment_dict
            global windows_to_close

            labels_list = [
                "Número de Trechos",
                "Número de Nós Theta (NTheta)",
                "Profundidade Inicial",
                "Tempo Inicial",
                "Ângulo Inicial",
                "Excentrecidade Inicial em X (mm)",
                "Excentrecidade Inicial em Y (mm)",
                "Omega",
                "Gravidade",
            ]

            fontePadrao = ("Arial", "10")
            container = Frame(main_frame)
            container["pady"] = 10

            title = Label(container, text="Variáveis do Modelo")
            title["font"] = ("Arial", "10", "bold")
            title.pack()

            for index in range(len(labels_list)):
```

```
            label = labels_list[index]
            segment_label = model_segment_labels[index]
            entry_container = Frame(container)
            entry_container["pady"] = 10
            entry_container.pack()

            curr_label = Label(entry_container, text=label, font=fontePadrao)
            curr_label.pack(side=LEFT)
            model_segment_dict[segment_label] = Entry(entry_container)
            model_segment_dict[segment_label]["width"] = 30
            model_segment_dict[segment_label]["font"] = fontePadrao
            model_segment_dict[segment_label].pack(side=RIGHT)

        container_button = Frame(container)
        container_button["pady"] = 20
        container_button.pack()

        button = Button(container_button)
        button["text"] = "Confirmar Dados"
        button["font"] = ("Calibri", "8")
        button["width"] = 20

        num_segments = model_segment_dict["NUM_SEGMENTS"]
        button["command"] = lambda: Definicao_Trechos_Geometria(num_segments)
        button.pack()

        ChangePageTo(container)
        CanvasReconfigure()

    def Entrada_Fluidos():
        global model_segment_dict

        labels_list = [
            "Número de Fluidos",
        ]

        fontePadrao = ("Arial", "10")
        container = Frame(main_frame)
        container["pady"] = 10
        container.pack()

        title = Label(container, text="Variáveis da Definição de Fluidos")
        title["font"] = ("Arial", "10", "bold")
        title.pack()

        for index in range(len(labels_list)):
            label = labels_list[index]
            fl_label = model_fluid_labels[index]
            entry_container = Frame(container)
```

```
            entry_container["pady"] = 10
            entry_container.pack()

            curr_label = Label(entry_container, text=label, font=fontePadrao)
            curr_label.pack(side=LEFT)
            fluid_dict[fl_label] = Entry(entry_container)
            fluid_dict[fl_label]["width"] = 30
            fluid_dict[fl_label]["font"] = fontePadrao
            fluid_dict[fl_label].pack(side=RIGHT)

        container_button = Frame(container)
        container_button["pady"] = 20
        container_button.pack()

        button = Button(container_button)
        button["text"] = "Confirmar Dados"
        button["font"] = ("Calibri", "8")
        button["width"] = 20

        num_fluidos = fluid_dict["NUM_FLUIDS"]
        button["command"] = lambda: Definicao_Fluidos(num_fluidos)
        button.pack()

        ChangePageTo(container)
        CanvasReconfigure()

    def Entrada_Operacao():
        global operation_dict

        labels_list = [
            "Passo de Tempo",
            "Número de Iterações (PRINT)",
            "Número de Iterações (PRINT REPORT)",
            "Número Máximo de Iterações Não-Newtonianas",
            "Velocidade de Pressionamento",
            "Tipo do Modelo",
            "Tipo de Entrada do Modelo",
            "Total de Pontos (Vazão)",
            "Total de Pontos (Pressão de Saída)",
        ]

        # PREENCHA AQUI MATIAS!!!!
        op_types = [
            "NonNewtTurbCyclConc",
        ]
        # PREENCHA AQUI MATIAS!!!!
        op_input_types = [
            "Inflow-profile",
        ]
```

```
        dict_types = {
            "TYPE": op_types,
            "INPUT_TYPE": op_input_types,
        }

        fontePadrao = ("Arial", "10")
        container = Frame(main_frame)
        container["pady"] = 10
        container.pack()

        title = Label(container, text="Variáveis da Definição de Fluidos")
        title["font"] = ("Arial", "10", "bold")
        title.pack()

        for index in range(len(labels_list)):
            label = labels_list[index]
            fl_label = operation_labels[index]

            entry_container = Frame(container)
            entry_container["pady"] = 10
            entry_container.pack()

            curr_label = Label(entry_container, text=label, font=fontePadrao)
            curr_label.pack(side=LEFT)
            if "TYPE" not in fl_label:
                operation_dict[fl_label] = Entry(entry_container)
                operation_dict[fl_label]["width"] = 30
                operation_dict[fl_label]["font"] = fontePadrao
                operation_dict[fl_label].pack(side=RIGHT)
            else:
                types_list = dict_types[fl_label]
                selected = StringVar(entry_container)
                selected.set(types_list[0])
                operation_dict[fl_label] = selected

                dropdown = OptionMenu(entry_container, selected, *types_list)
                dropdown["width"] = 15
                dropdown["font"] = fontePadrao
                dropdown.pack(side=RIGHT)

        container_button = Frame(container)
        container_button["pady"] = 20
        container_button.pack()

        button = Button(container_button)
        button["text"] = "Salvar Dados"
        button["font"] = ("Calibri", "8")
        button["width"] = 20
```

```
            num_points_flow = operation_dict["TOTAL_FLOW_RATE"]
            num_points_press = operation_dict["TOTAL_PRESSURE_OUT"]
            # button["command"] = lambda: Fim_Operacao(operation_dict)
            button["command"]                       =                       lambda:
Definicao_Perfil_Vazao_Pressao(num_points_flow, num_points_press)
            button.pack()

            ChangePageTo(container)
            CanvasReconfigure()

        ############################################################
        ############ FILE GENERATION FUNCTIONS ###################
        ############################################################

        def ModelSegmentDictToFile(f):
            default_label = "%MODEL."
            for label in model_segment_labels:
                if model_segment_dict:
                    variable = model_segment_dict[label]
                    f.write(default_label + label + "\n")
                    f.write(variable + "\n\n")
            f.write(default_label + "END\n\n")

        def ModelOperationDictToFile(f):
            default_label = "%MODEL."
            f.write(default_label + "BEGIN\n\n")
            for i in range(len(operation_labels)):
                label = operation_labels[i]
                if ("TOTAL" not in label) and operation_dict:
                    variable = operation_dict[label]
                    f.write(default_label + label + "\n")
                    f.write(variable + "\n\n")

        def defineFunction(curr_x1, curr_y1, curr_x2, curr_y2):
            coef_a = (curr_y1 - curr_y2)/(curr_x1 - curr_x2)
            coef_b = curr_y1 - coef_a*curr_x1

            return coef_a, coef_b

        def calculateFunctionImage(t, a, b):
            return a*t + b

        def getNewXY(profile_list, currInt, x_label, y_label):
            x = profile_list[currInt][x_label]
            y = profile_list[currInt][y_label]

            return x, y
```

```
      def ModelPointsDictToFile(f):
          if operation_dict:
              dt = float(operation_dict["DT"])
          for profile_type, profile_list in operation_matrix.items():
              currTime=0
              currInt = 0
              currStep = 0
              totalInt = len(profile_list) - 1
              x_label = list(profile_list[0].keys())[0]
              y_label = list(profile_list[0].keys())[1]
              curr_x1,  curr_y1  =  getNewXY(profile_list,  currInt,  x_label,
y_label)
              curr_x2,  curr_y2  =  getNewXY(profile_list,  currInt+1,  x_label,
y_label)
              a, b = defineFunction(curr_x1, curr_y1, curr_x2, curr_y2)
              last_x = profile_list[-1][x_label]
              totalSteps = int((last_x - curr_x1)/dt)

              f.write("%MODEL."+profile_type+".PROFILE\n")
              f.write("%d\n"%(totalSteps))

              while currInt < totalInt:
                  # Se houve mudança de intervalo,
                  # incrementa o contador de intervalos
                  # e define a nova função
                  if currTime > curr_x2 :
                      currInt += 1
                      if currInt >= totalInt:
                          continue
                      curr_x1,   curr_y1   =   getNewXY(profile_list,   currInt,
x_label, y_label)
                      curr_x2,   curr_y2   =   getNewXY(profile_list,   currInt+1,
x_label, y_label)
                      a, b = defineFunction(curr_x1, curr_y1, curr_x2, curr_y2)

                  image = calculateFunctionImage(currTime, a, b)

                  f.write("%d  %f\n"%(currStep, image))

                  currTime += dt
                  currStep += 1
              f.write("%d\n\n")

      def FluidMatrixToFile(f):
          default_label = "%FLUID."
          for i in range(len(fluid_matrix)):
              f.write(default_label + "BEGIN\n\n")
              if len(fluid_matrix) > 0:
                  line = fluid_matrix[i]
```

```
                for label in fluid_labels:
                    if label != "ID":
                        f.write(default_label + label + "\n")
                        f.write(line[label] + "\n\n")
                    else:
                        f.write(default_label + label + "\n")
                        f.write(str(i+1) + "\n\n")
            f.write(default_label + "END\n\n")

    def GeometryMatrixToFile(f):
        default_label = "%SEGMENT."
        for i in range(len(geometry_matrix)):
            f.write(default_label + "BEGIN\n\n")
            line = geometry_matrix[i]
            if len(geometry_matrix) > 0:
                for label in segment_labels:
                    if label != "ID":
                        f.write(default_label + label + "\n")
                        f.write(line[label] + "\n\n")
                    else:
                        f.write(default_label + label + "\n")
                        f.write(str(i+1) + "\n\n")
            f.write(default_label + "END\n\n")

    def GerarArquivo(entry):
        fileName = entry.get()
        f = open(fileName+".txt", "w")
        default_label = "%WELL."
        f.write(default_label + "BEGIN\n\n")
        ModelOperationDictToFile(f)
        ModelPointsDictToFile(f)
        ModelSegmentDictToFile(f)
        GeometryMatrixToFile(f)
        FluidMatrixToFile(f)
        f.write(default_label + "END\n\n")
        f.close()

        OpenNewMessageWindow("Arquivo salvo com sucesso!")
        BackToFirstPage()

    def CriarArquivo():
        container = Frame(main_frame)
        container["pady"] = 10
        container.pack()
        fontePadrao = ("Arial", "10")

        title = Label(container, text="Salvar Arquivo de Entrada do Modelo")
        title["font"] = ("Arial", "10", "bold")
        title.pack()
```

```
          entry_container = Frame(container)
          entry_container["pady"] = 10
          entry_container.pack()

          label    =    Label(entry_container,    text="Nome    do    Arquivo",
font=fontePadrao)
          label.pack(side=LEFT)

          entry_input = Entry(entry_container)
          entry_input["width"] = 30
          entry_input["font"] = fontePadrao
          entry_input.pack(side=RIGHT)

          submit_button = Button(container)
          submit_button["text"] = "Salvar Arquivo"
          submit_button["font"] = ("Calibri", "8")
          submit_button["width"] = 20

          submit_button["command"] = lambda: GerarArquivo(entry_input)
          submit_button.pack(anchor=CENTER)

          ChangePageTo(container)
          CanvasReconfigure()

      def PostProcessing():
          container = Frame(main_frame)
          container["pady"] = 10
          container.pack()
          fontePadrao = ("Arial", "10")

          title = Label(container, text="Ler Arquivo de Entrada do Modelo")
          title["font"] = ("Arial", "10", "bold")
          title.pack()

          entry_container = Frame(container)
          entry_container["pady"] = 10
          entry_container.pack()

          label    =    Label(entry_container,    text="Nome    do    Arquivo",
font=fontePadrao)
          label.pack(side=LEFT)

          entry_input = Entry(entry_container)
          entry_input["width"] = 30
          entry_input["font"] = fontePadrao
          entry_input.pack(side=RIGHT)

          submit_button = Button(container)
```

```
            submit_button["text"] = "Gerar Imagens"
            submit_button["font"] = ("Calibri", "8")
            submit_button["width"] = 20

            submit_button["command"] = lambda: CreateImages(entry_input)
            submit_button.pack(anchor=CENTER)

            ChangePageTo(container)
            CanvasReconfigure()

        def ConsultarArquivo():
            ftypes = [('txt file',"*.txt")]
            ttl  = "Get Last Generated File"
            dir1 = ''
            fileName = askopenfilename(filetypes = ftypes, initialdir = dir1,
title = ttl)

            if sys.platform != "darwin":
                system('notepad.exe ' + fileName)
            else:
                system('open -a TextEdit ' + fileName)

        ############################################################
        ################ MAIN PROGRAM WINDOW ######################
        ############################################################

        def PreProcessing():
            window.title("Entrada de Dados")

            container_test=Frame(main_frame)
            container_test["pady"] = 10

            bt_Geometria    =    Button(container_test,    width=30,    height=3,
text="Geometria")
            bt_Geometria["command"] = Entrada_Geometria
            bt_Geometria.pack(anchor=CENTER)

            bt_Fluido     =     Button(container_test,     width=30,     height=3,
text="Fluido(s)")
            bt_Fluido["command"] = Entrada_Fluidos
            bt_Fluido.pack(anchor=CENTER)

            bt_Operação     =     Button(container_test,     width=30,     height=3,
text="Condições de Operação")
            bt_Operação["command"] = Entrada_Operacao
            bt_Operação.pack(anchor=CENTER)

            bt_Consulta_Arquivo  =  Button(container_test,  width=30,  height=3,
text="Consultar Arquivo")
```

```
          bt_Consulta_Arquivo["command"] = ConsultarArquivo
          bt_Consulta_Arquivo.pack(anchor=CENTER)

          bt_Gera_Arquivo   =   Button(container_test,   width=30,   height=3,
text="Criar Arquivo")
          bt_Gera_Arquivo["command"] = CriarArquivo
          bt_Gera_Arquivo.pack(anchor=CENTER)

          ChangePageTo(container_test)
          CanvasReconfigure()

      def CreateImages(entry):
          import matplotlib.pyplot as plt
          from matplotlib.table import Table
          import numpy as np
          from os import listdir, mkdir, system
          from os.path import isfile, join, exists

          inputFileName = entry.get()

          def findInterval(intvList, val):
              for i in range(len(intvList)-1):
                  if intvList[i] <= val < intvList[i+1]:
                      return i
              return -1

          def getFilesList(path):
                filesList = []
                for file in listdir(path):
                  newPath = join(path,file)
                  if isfile(newPath) and ".txt" in file:
                    filesList.append(newPath)
                return filesList

          inputFile = open(inputFileName + ".pco", "r")

          for line in inputFile:
              line = line.strip()
              sep = "NUM_INTE_PRINT_REPORT"
              if sep in line:
                  step = inputFile.readline()
                  step = int(step)
                  break

          colors = ['blue', 'green', 'red', 'blue', 'green', 'red']
          bounds = []
          bounds.append(1)
          rate = 4/3
          first = bounds[0]
```

```
        for index in range(1, 7):
            bounds.append(first - rate*index)
        bounds = bounds[::-1]

        folderPath = "saidas"
        #prefix = "1_"
        prefix = inputFileName+"_"
        imgFolderPath = "outputs"

        if not exists(prefix+imgFolderPath):
            mkdir(prefix+imgFolderPath)

        filesList = getFilesList(prefix+"saidas")

        for fileName in filesList:
            currFile = open(fileName)
            currMatrix = []
            fig, ax = plt.subplots()

            for line in currFile:
                line = line.strip()
                listAux = line.split(";")
                for i in range(len(listAux)):
                    currNum = listAux[i]
                    if currNum != "":
                        listAux[i] = float(currNum)
                    else:
                        listAux.pop(i)
                currMatrix.append(listAux[::-1])

                data = np.array(currMatrix)
                data = data.T

                nRows = len(data)
                nCols = len(data[0])

                tb = Table(ax, bbox=[0,0,1,1])

                width, height = 1.0 / nCols, 1.0 / nRows

                for i in range(nRows):
                    for j in range(nCols):
                        currValue = data[i][j]
                        idx = findInterval(bounds, currValue)
                        color = colors[idx]
                        tb.add_cell(i, j, width, height, facecolor=color)

                ax.add_table(tb)
```

```
                noFolderName = fileName.split("/")[-1]
                imgName = noFolderName.split(".")[0]
                print("Creating image plot for file: "+noFolderName)
                plt.savefig(imgFolderPath+"/"+imgName+'.png')
                plt.close(fig)

            BackToFirstPage()
            CanvasReconfigure()

        def BackToFirstPage():
            page1 = FirstPage()
            ChangePageTo(page1)
            CanvasReconfigure()

        def OpenNewMessageWindow(message):
            # Opening new window
            new_window = Tk()
            new_container = Frame(new_window)
            new_container["pady"] = 10
            new_container.pack()

            success_message = Label(new_container, text=message)
            success_message["font"] = ("Arial", "10", "bold")
            success_message.pack()

        def ChangePageTo(page):
            global curr_page

            if curr_page:
                curr_page.destroy()
            curr_page = page
            curr_page.pack()

        def FirstPage():
            window.title("Início")

            container_test=Frame(main_frame)
            container_test["pady"] = 10

            bt_Pre  =  Button(container_test,  width=30,  height=3,  text="Pré-
Processamento")
            bt_Pre["command"] = PreProcessing
            bt_Pre.pack(anchor=CENTER)

            bt_Pos  =  Button(container_test,  width=30,  height=3,  text="Pós-
Processamento")
            bt_Pos["command"] = PostProcessing
            bt_Pos.pack(anchor=CENTER)
```

```
            return container_test

        def CanvasConf(event):
curr_canvas.configure(scrollregion=curr_canvas.bbox("all"),width=370,height=60
0)

        def CanvasReconfigure():
            curr_canvas.configure(yscrollcommand=myscrollbar.set)
            curr_canvas.create_window((0,0), window=curr_page, anchor='nw')
            curr_page.bind("<Configure>", CanvasConf)

        window = Tk()
        sizex = 400
        sizey = 600
        posx  = 100
        posy  = 100
        window.resizable(False, False)
        window.wm_geometry("%dx%d+%d+%d" % (sizex, sizey, posx, posy))

        main_frame = Frame(window,width=50,height=100)
        main_frame["pady"] = 10
        main_frame.pack()

        curr_canvas=Canvas(main_frame)
        page1 = FirstPage()
        ChangePageTo(page1)

        myscrollbar=Scrollbar(main_frame,orient="vertical",command=curr_canvas.y
view)
        curr_canvas.configure(yscrollcommand=myscrollbar.set)

        myscrollbar.pack(side="right",fill="y")
        curr_canvas.pack()

        curr_canvas.create_window((0,0), window=curr_page, anchor='nw')
        curr_page.bind("<Configure>", CanvasConf)

        window.mainloop()

        ############################################################
        ############################################################
```