Universidade Estadual de Santa Cruz - UESC

Relatórios de Implementações de Métodos da Disciplina Análise Numérica

Relatório de implementações realizadas por Igor Lima Rocha

Disciplina Análise Numérica.

Curso Ciência da Computação

Semestre 2023.2

Professor Gesil Sampaio Amarante II

Ilhéus - BA

2023

Índice

Lista de Figuras

Método da Bissecção

Método da Posição Falsa

Método de Newton-Raphson

Método da Secante

Método da Eliminação de Gauss

Método da Fatoração LU

Método de Jacobi

Imagem 7.1 - Retornos gerais das iterações do algoritmo

Método de Gauss-Seidel

Imagem 8.1 - Retornos gerais das iterações do algoritmo

Linguagem escolhida e justificativas

Nesse contexto, optei por utilizar a linguagem de programação Python, juntamente com as bibliotecas pandas e sympy, devido a várias razões que destacam suas vantagens significativas.

A escolha de Python como linguagem principal baseia-se na sua crescente popularidade na comunidade de desenvolvimento e na sua ampla adoção na área científica e de análise numérica. Python oferece uma sintaxe clara e legível, o que torna a implementação dos algoritmos mais intuitiva e facilita a colaboração entre membros da equipe. Além disso, sua vasta coleção de bibliotecas especializadas simplifica a execução de tarefas complexas, economizando tempo e esforço de desenvolvimento.

A inclusão da biblioteca pandas no projeto é justificada pela sua capacidade de manipular e analisar dados tabulares, como os presentes em arquivos CSV. Através do uso do pandas, conseguimos importar facilmente os dados do arquivo e manipulá-los em estruturas de dados flexíveis, tornando a exploração dos dados e a extração de informações relevantes uma tarefa mais eficiente.

A biblioteca sympy desempenha um papel fundamental na nossa abordagem, já que nos permite realizar cálculos simbólicos e manipulação algébrica, essenciais para a solução de equações e determinação de zeros de funções. Através do sympy, podemos definir variáveis simbólicas, criar equações e resolver algebricamente, garantindo uma abordagem precisa e confiável na determinação dos zeros de funções em intervalos específicos. Além disso, o sympy oferece integração com outras bibliotecas numéricas, permitindo-nos combinar os benefícios da manipulação simbólica com os métodos numéricos disponíveis.

Em resumo, a escolha de Python como linguagem principal, combinada com as bibliotecas pandas e sympy, oferece uma abordagem abrangente e eficiente para a implementação dos algoritmos de cálculo numérico. Essas ferramentas nos permitem aproveitar as vantagens da simplicidade de desenvolvimento, manipulação de dados tabulares e cálculos simbólicos, garantindo a precisão e eficiência na determinação dos zeros de funções, como requerido no escopo deste projeto.

Instruções gerais

Separei as partes principais do programa em componentes diferentes, para cada parte ficar com sua responsabilidade.

O programa dentro de “**main.py**” engloba todos os métodos. Para executar, você deve rodar o comando `python main.py` e selecionar o método desejado, assim como algumas configurações gerais.

Criei uma classe geral, que recebe a string da equação, e a transforma em uma equação do **SymPy** (biblioteca escolhida para tratar matematicamente as funções). **Essa classe vai ser utilizada em todos os métodos.**

Os arquivos de entrada devem estar na pasta **“entradas”**, da forma que é enunciado no tópico do método.

Já os arquivos de saída serão gerados em pastas separadas, dentro da pasta “**saidas**”.

Método 1 (Bissecção)

Estratégia de Implementação

Para tratar exclusivamente do método de bissecção, criei uma função que recebe a equação, o ponto A, o ponto B, o epsilon de tolerância, e a quantidade N máxima de iterações.

Nessa função, ocorre um loop N vezes, realizando o método, encontrando os valores da função nos pontos A, B e C, onde C é o ponto médio de cada iteração.

No final, a função retorna (caso exista) um ponto mais próximo do ótimo, e o histórico de iterações.

Estrutura dos Arquivos de Entrada/Saída

Um único arquivo de entrada é necessário, deve chamá-lo de "**bisseccao.csv**" e estar dentro da pasta **"entradas"**, o arquivo deve ser separado por vírgulas, e ter os seguintes cabeçalhos:

- **equation**: A string que contém a função a ser avaliada. Deve estar nos padrões do SymPy, então:
    - A única variável deve ser "x"
    - x^3 deve ser escrito como **x**3**
    - **cosseno de X** deve ser escrito como **cos(x)**
    - **seno de X** deve ser escrito como **sin(x)**
    - números reais devem ser escritos com ponto: cinco vírgula cinco = 5.5
- **a**: ponto mais à esquerda do intervalo de busca, se for número real, **deve ser escrito com ponto.**
- **b**: ponto mais à direita do intervalo de busca, se for número real, **deve ser escrito com ponto.**
- **tolerance**: valor de tolerância de erro, exemplo: 0.00001
- **max_iterations**: inteiro que define a quantidade máxima de iterações

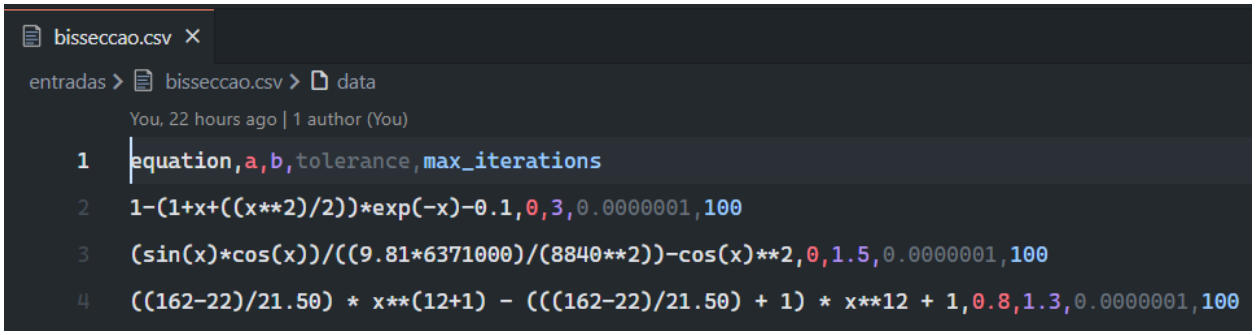

Imagem 1.1 - Exemplo de arquivo de entrada do método bissecção

Já os arquivos de saída estão na pasta "**bisseccao**" e dentro vai existir 1 arquivo para cada equação passada no arquivo de entrada, com o nome **"result_{index}.csv"**. Esse arquivo possui as seguintes colunas:

- **Iteration**: o índice da iteração
- **a**: o valor do intervalo mais a esquerda
- **b**: o valor do intervalo mais a direita
- **f(a)**: o valor da função no ponto a
- **f(b)**: o valor da função no ponto b
- **b-a**: a diferença entre b e a
- **c**: o valor do ponto médio escolhido
- **f(c)**: o valor da função no ponto médio

Aqui está um exemplo de arquivo de saída, omiti as linhas do meio para caber na captura.

Imagem 1.2 - Exemplo de arquivo de saída do método bissecção

Problemas teste

Foram realizados testes utilizando os princípios e técnicas descritos no livro **"Cálculo Numérico" de Neide Franco**. A implementação foi avaliada em relação a três problemas apresentados no livro. Os resultados obtidos foram consistentes com métodos amplamente reconhecidos, evidenciando a eficácia da implementação.

As questões escolhidas foram:

- **3.3** da página 97
- **3.6** da página 100
- **3.8** da página 100

```
---------- Execução por Bisseção ----------

Equação: 1-(1+x+((x**2)/2))*exp(-x)-0.1
Intervalo: (0.0, 3.0)
Tolerância: 1e-07
Zero encontrado: 1.1020653247833252 em 22 iterações
Tempo decorrido: 0.015415668487548828 segundos

Equação: (sin(x)*cos(x))/((9.81*6371000)/(8840**2))-cos(x)**2
Intervalo: (0.0, 1.5)
Tolerância: 1e-07
Zero encontrado: 0.67460846900939994 em 21 iterações
Tempo decorrido: 0.011505126953125 segundos

Equação: ((162-22)/21.50) * x**(12+1) - (((162-22)/21.50) + 1) * x**12 + 1
Intervalo: (0.8, 1.3)
Tolerância: 1e-07
Zero encontrado: 0.9999999880790711 em 23 iterações
Tempo decorrido: 0.011596441268920898 segundos
```

Imagem 1.3 - Retornos gerais das iterações do algoritmo

$$f(x) = 1 - \left(1 + x + \frac{x^2}{2}\right) e^{-x} - 0.1$$

$$\text{Raízes}(f, -0.3757402670049, 2.2078917208626)$$

$$= A = (1.1020652446445, 0)$$

Imagem 1.4 - Verificação pelo Geogebra da questão 3.3

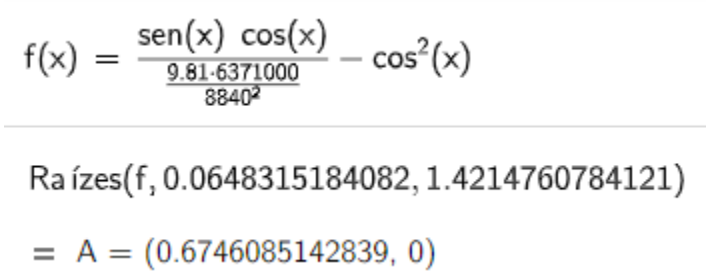

Imagem 1.5 - Verificação pelo Geogebra da questão 3.6

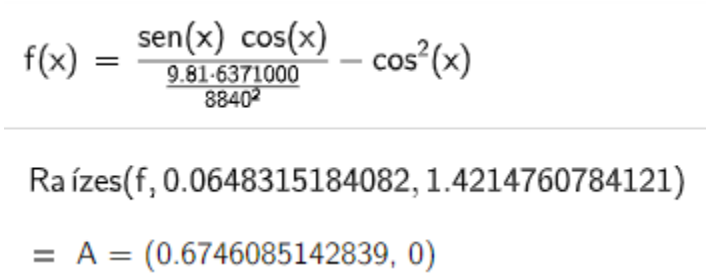

Imagem 1.6 - Verificação pelo Geogebra da questão 3.6

Dificuldades enfrentadas

Minha maior dificuldade foi conseguir organizar o código para reutilizar as funções e deixar mais fácil de entender, já que nesse momento, eu estava organizando todo o código para ser reutilizado nos métodos seguintes.

Método 2 (Posição Falsa)

Estratégia de Implementação

Para o método da Posição Falsa, o código do método Bissecção foi reutilizado por completo, exceto no cálculo do C, que agora não é mais o valor médio, e sim o ponto da reta AB que corta o eixo X.

Estrutura dos Arquivos de Entrada/Saída

Um único arquivo de entrada é necessário, deve chamá-lo de "**posicao_falsa.csv**" e estar dentro da pasta **"entradas"**, o arquivo deve ser separado por vírgulas, e ter os seguintes cabeçalhos:

- **equation**: A string que contém a função a ser avaliada. Deve estar nos padrões do SymPy, então:
  - A única variável deve ser "x"
  - x^3 deve ser escrito como **x**3**
  - **cosseno de X** deve ser escrito como **cos(x)**
  - **seno de X** deve ser escrito como **sin(x)**
  - números reais devem ser escritos com ponto: cinco vírgula cinco = 5.5
- **a**: ponto mais à esquerda do intervalo de busca, se for número real, **deve ser escrito com ponto.**
- **b**: ponto mais à direita do intervalo de busca, se for número real, **deve ser escrito com ponto.**
- **tolerance**: valor de tolerância de erro, exemplo: 0.00001
- **max_iterations**: inteiro que define a quantidade máxima de iterações

O arquivo de entrada pode ser o mesmo utilizado no método de bissecção.

Já os arquivos de saída estão na pasta "**posicao_falsa**" e dentro vai existir 1 arquivo para cada equação passada no arquivo de entrada, com o nome **"result_{index}.csv"**. Esse arquivo possui as seguintes colunas:

- **Iteration**: o índice da iteração
- **a**: o valor do intervalo mais a esquerda
- **b**: o valor do intervalo mais a direita

- **f(a)**: o valor da função no ponto a
- **f(b)**: o valor da função no ponto b
- **b-a**: a diferença entre b e a
- **c**: o valor do ponto médio escolhido
- **f(c)**: o valor da função no ponto médio

O arquivo de saída também vai ser igual ao do método de bissecção, porém o número de linhas pode ser diferente, devido à diferença de implementação.

Problemas teste

Foram realizados testes utilizando os princípios e técnicas descritos no livro **"Cálculo Numérico" de Neide Franco**. A implementação foi avaliada em relação a três problemas apresentados no livro. Os resultados obtidos foram consistentes com métodos amplamente reconhecidos, evidenciando a eficácia da implementação.

As questões escolhidas foram:

- **3.3** da página 97
- **3.6** da página 100
- **3.8** da página 100

```
---------- Execução por PosiçãoFalsa ----------

Equação: 1-(1+x+((x**2)/2))*exp(-x)-0.1
Intervalo: (0.0, 3.0)
Tolerância: 1e-07
Zero encontrado: 1.10206519809366 em 11 iterações
Tempo decorrido: 0.0064127445220947266 segundos

Equação: (sin(x)*cos(x))/((9.81*6371000)/(8840**2))-cos(x)**2
Intervalo: (0.0, 1.5)
Tolerância: 1e-07
Zero encontrado: 0.6746085406216 26 em 12 iterações
Tempo decorrido: 0.006006002426147461 segundos

Equação: ((162-22)/21.50) * x**(12+1) - (((162-22)/21.50) + 1) * x**12 + 1
Intervalo: (0.8, 1.3)
Tolerância: 1e-07
Zero encontrado: -0.808715305298452 em 38 iterações
Tempo decorrido: 0.011995553970336914 segundos
```

Imagem 2.1 - Retornos gerais das iterações do algoritmo

Como podemos perceber, utilizar o método de posição falsa fez com que encontrássemos o zero da função mais rápido nas 2 primeiras equações, porém mais lento na terceira.

Isso se deve, provavelmente, ao fato de a terceira equação possuir 2 zeros no intervalo definido, como podemos ver na imagem abaixo:

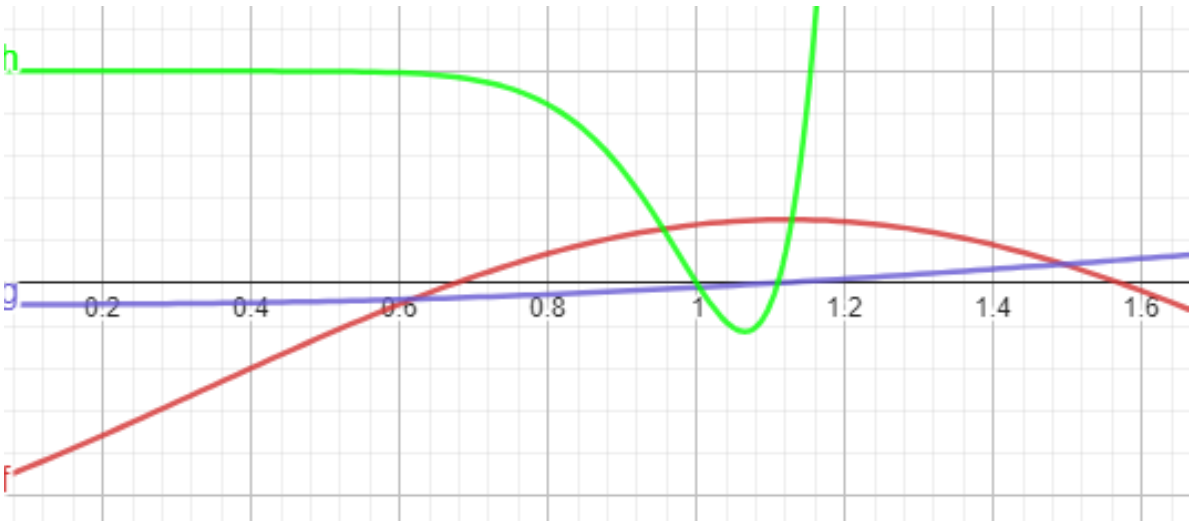

Imagem 2.2 - Gráfico das funções, pelo Geogebra

A função da questão 3.3 está em azul, a questão 3.6 está em vermelho, e a 3.8 está em verde.

Isso deve ter feito com que o algoritmo se confunda na hora de decidir para qual lado ele deve seguir.

Dificuldades enfrentadas

Não houveram dificuldades.

Método 3 (Newton-Raphson)

Estratégia de Implementação

Já para o método de Newton-Raphson, a função recebe apenas a equação, o ponto A, a tolerância e a quantidade N máxima de iterações.

Nessa função, ocorre um loop N vezes, onde a cada iteração, é calculado o valor da função no ponto STEP atual (o primeiro STEP é o ponto A), e a derivada nesse ponto. Verifica se o valor da função no ponto STEP satisfaz a tolerância. Se não satisfazer, será calculado o próximo STEP, a partir da equação de Newton-Raphson.

No final, a função retorna (caso exista) um ponto mais próximo do ótimo, e o histórico de iterações.

Estrutura dos Arquivos de Entrada/Saída

Um único arquivo de entrada é necessário, deve chamá-lo de "**newton_raphson.csv**" e estar dentro da pasta **"entradas"**, o arquivo deve ser separado por vírgulas, e ter os seguintes cabeçalhos:

- **equation**: A string que contém a função a ser avaliada. Deve estar nos padrões do SymPy, então:
    - A única variável deve ser "x"
    - x^3 deve ser escrito como **x**3**
    - **cosseno de X** deve ser escrito como **cos(x)**
    - **seno de X** deve ser escrito como **sin(x)**
    - números reais devem ser escritos com ponto: cinco vírgula cinco = 5.5
- **a**: ponto inicial de busca, se for número real, **deve ser escrito com ponto.**
- **tolerance**: valor de tolerância de erro, exemplo: 0.00001
- **max_iterations**: inteiro que define a quantidade máxima de iterações

O arquivo de entrada desse método é um pouco diferente dos demais, já que ele não tem a coluna "b". Como pode ser visto na imagem abaixo:

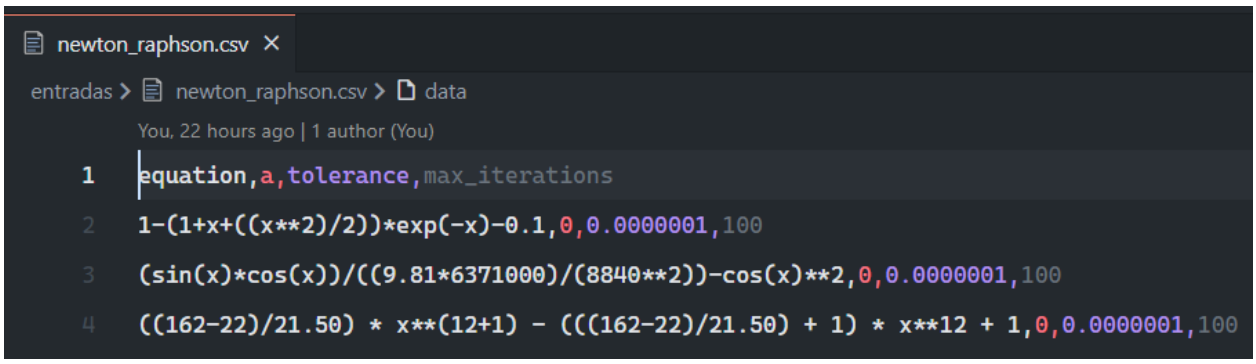

Imagem 3.1 - Exemplo de arquivo de entrada do método Newton-Raphson

Já os arquivos de saída estão na pasta "**newton_raphson**" e dentro vai existir 1 arquivo para cada equação passada no arquivo de entrada, com o nome **"result_{index}.csv"**. Esse arquivo possui as seguintes colunas:

- **Iteration**: o índice da iteração
- **step**: o valor do intervalo mais a esquerda
- **f(step)**: o valor da função no ponto step
- **f'(step)**: a derivada da função no ponto step

Aqui está um exemplo de arquivo de saída

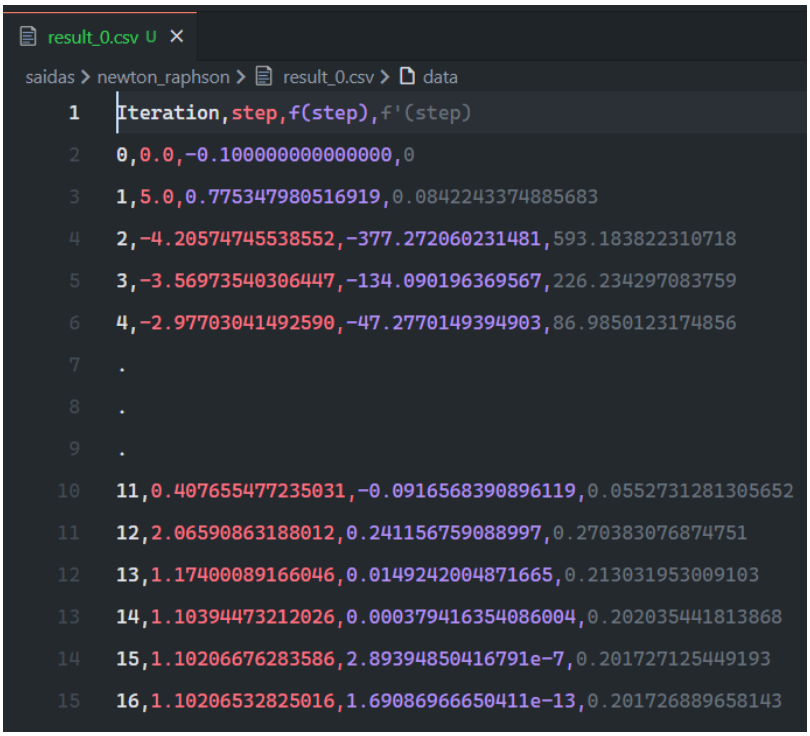

Imagem 3.2 - Exemplo de arquivo de saída do método Newton-Raphson

Problemas teste

Foram realizados testes utilizando os princípios e técnicas descritos no livro **"Cálculo Numérico" de Neide Franco**. A implementação foi avaliada em relação a três problemas apresentados no livro. Os resultados obtidos foram consistentes com métodos amplamente reconhecidos, evidenciando a eficácia da implementação.

As questões escolhidas foram:

- **3.3** da página 97
- **3.6** da página 100
- **3.8** da página 100

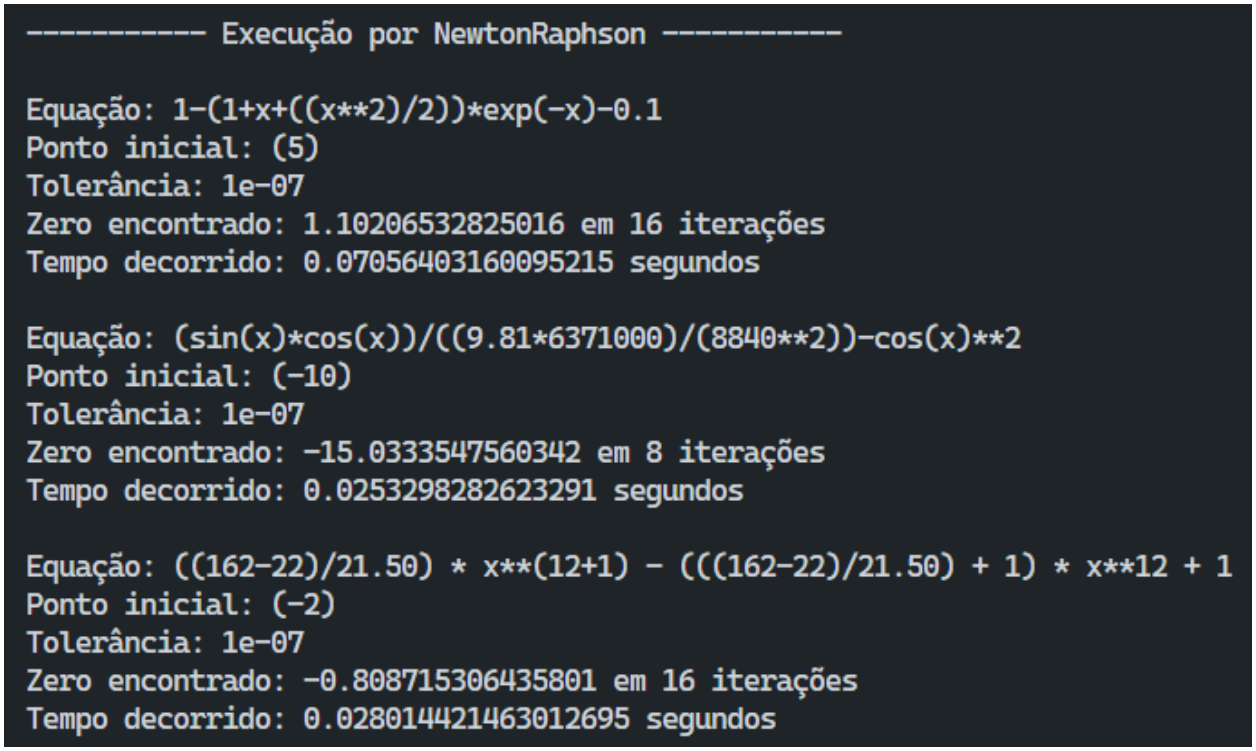

Imagem 3.3 - Retornos gerais das iterações do algoritmo

Como podemos perceber, o método de Newton-Raphson teve os melhores resultados até agora.

Porém existe um problema que pode acontecer durante a execução do algoritmo: Caso a taxa de variação no ponto STEP seja zero, ou seja, naquele ponto a função é quase uma reta horizontal, isso significa que a derivada naquele ponto é zero.

Dessa forma, quebra o código, já que a função de Newton-Raphson é

$$X_{k+1} = X - \frac{f(X_k)}{f'(X_k)}$$

```
---------- Execução por NewtonRaphson ----------

Equação: 1-(1+x+((x**2)/2))*exp(-x)-0.1
Ponto inicial: (0)
Tolerância: 1e-07
Derivada muito próxima de zero, o sistema não está preparado para isso.
Nenhum zero encontrado para a equação 1-(1+x+((x**2)/2))*exp(-x)-0.1 começan
do no ponto (0.0)
Tempo decorrido: 0.001996278762817383 segundos

Equação: (sin(x)*cos(x))/((9.81*6371000)/(8840**2))-cos(x)**2
Ponto inicial: (0)
Tolerância: 1e-07
Zero encontrado: 0.6746084507913 63 em 5 iterações
Tempo decorrido: 0.0049893856048583984 segundos

Equação: ((162-22)/21.50) * x**(12+1) - (((162-22)/21.50) + 1) * x**12 + 1
Ponto inicial: (0)
Tolerância: 1e-07
Derivada muito próxima de zero, o sistema não está preparado para isso.
Nenhum zero encontrado para a equação ((162-22)/21.50) * x**(12+1) - (((162-
22)/21.50) + 1) * x**12 + 1 começando no ponto (0.0)
Tempo decorrido: 0.002148866653442383 segundos
```

Imagem 3.4 - Código com problemas, pois tentou dividir por zero em 2 das equações

$$f(x) = 1 - \left(1 + x + \frac{x^2}{2}\right) e^{-x} - 0.1$$

$$g(x) = \frac{\text{sen}(x)\ \cos(x)}{\frac{9.81 \cdot 6371000}{8840^2}} - \cos^2(x)$$

$$h(x) = \frac{162 - 22}{21.5}\ x^{12+1} - \left(\frac{162 - 22}{21.5} + 1\right) x^{12} + 1$$

$$a = f'((0, f(0)))$$

$$= 0$$

$$b = g'((0, g(0)))$$

$$= 1.2503394026609$$

$$c = h'((0, h(0)))$$

$$= 0$$

Imagem 3.5 - Calculando pelo Geogebra as derivadas das funções no ponto 0

Dificuldades enfrentadas

Como citado acima, ocorre um erro quando encontramos uma derivada de função igual a zero.

Para corrigir esse problema, na iteração que a derivada é igual a zero, faço com que o ponto STEP seja acrescentado de um valor fixo (no caso, acrescentado em 5).

Fazendo isso, o algoritmo consegue sair da reta horizontal em alguns passos.

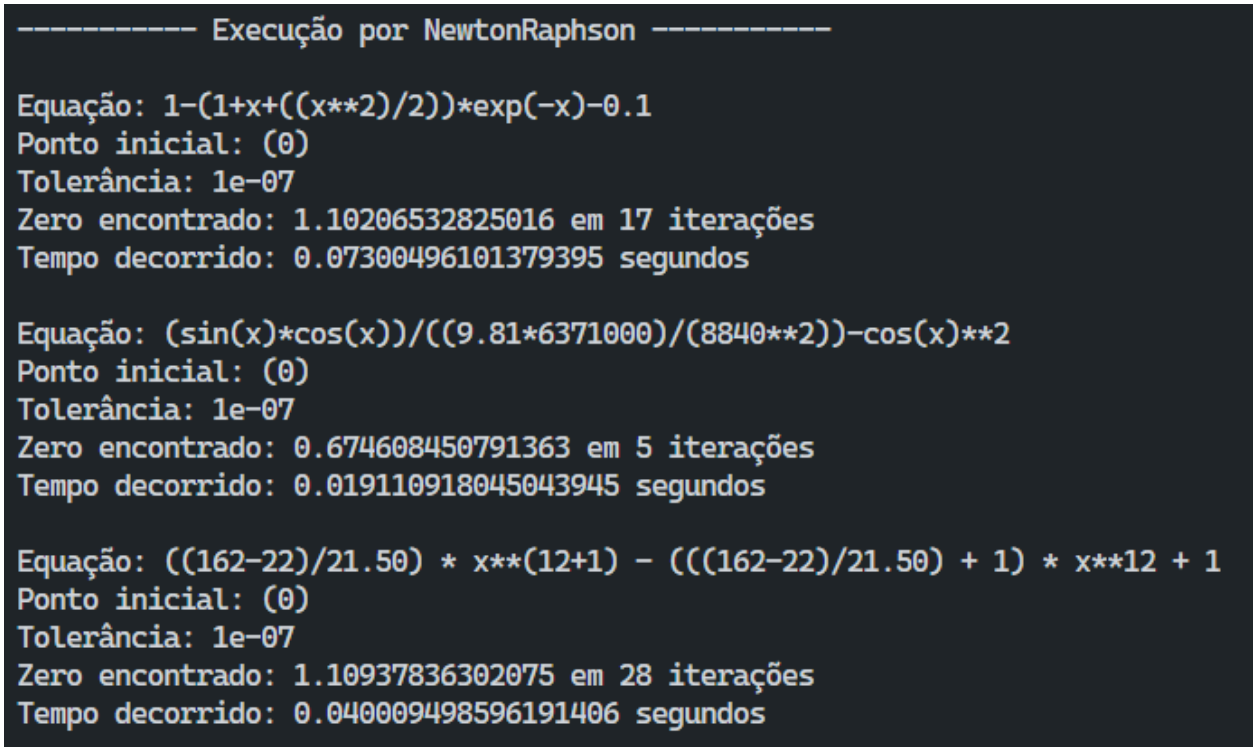

Imagem 3.6 - Resultado do algoritmo corrigido

Dessa forma o algoritmo conseguiu encontrar o zero da função até mais rapidamente do que outros algoritmos.

OBS: os zeros encontrados são diferentes pois as funções 2 e 3 tem mais de 2 pontos cortando o eixo X.

Método 4 (Secante)

Estratégia de Implementação

O algoritmo usando o método da Secante ficou bem parecido com o da Posição Falsa, ele recebe a equação, o ponto A, o ponto B, a tolerância e o número máximo N de iterações.

No loop, ele calcula o valor da função nos pontos A e B, e depois calcula o próximo ponto, a partir da equação da secante.

No final, também retorna o zero da função (caso exista) e o histórico de iterações.

Estrutura dos Arquivos de Entrada/Saída

Um único arquivo de entrada é necessário, deve chamá-lo de "**secante.csv**" e estar dentro da pasta **"entradas"**, o arquivo deve ser separado por vírgulas, e ter os seguintes cabeçalhos:

- **equation**: A string que contém a função a ser avaliada. Deve estar nos padrões do SymPy, então:
    - A única variável deve ser "x"
    - x^3 deve ser escrito como **x**3**
    - **cosseno de X** deve ser escrito como **cos(x)**
    - **seno de X** deve ser escrito como **sin(x)**
    - números reais devem ser escritos com ponto: cinco vírgula cinco = 5.5
- **a**: ponto mais à esquerda do intervalo de busca, se for número real, **deve ser escrito com ponto.**
- **b**: ponto mais à direita do intervalo de busca, se for número real, **deve ser escrito com ponto.**
- **tolerance**: valor de tolerância de erro, exemplo: 0.00001
- **max_iterations**: inteiro que define a quantidade máxima de iterações

O arquivo de entrada pode ser igual ao do método de bissecção.

Já o arquivo de saída está na pasta "**secante**" e dentro vai existir 1 arquivo para cada equação passada no arquivo de entrada, com o nome "equation_{index}.csv". Esse arquivo possui as seguintes colunas:

- **Iteration**: o índice da iteração
- **a**: o valor do intervalo mais a esquerda
- **f(a)**: o valor da função no ponto a
- **b**: o valor do intervalo mais a direita
- **f(b)**: o valor da função no ponto b

Aqui está um exemplo de arquivo de saída

Imagem 4.1 - Exemplo de arquivo de saída do método secante

Problemas teste

Foram realizados testes utilizando os princípios e técnicas descritos no livro **"Cálculo Numérico" de Neide Franco**. A implementação foi avaliada em relação a três problemas apresentados no livro. Os resultados obtidos foram consistentes com métodos amplamente reconhecidos, evidenciando a eficácia da implementação.

As questões escolhidas foram:

- **3.3** da página 97
- **3.6** da página 100
- **3.8** da página 100

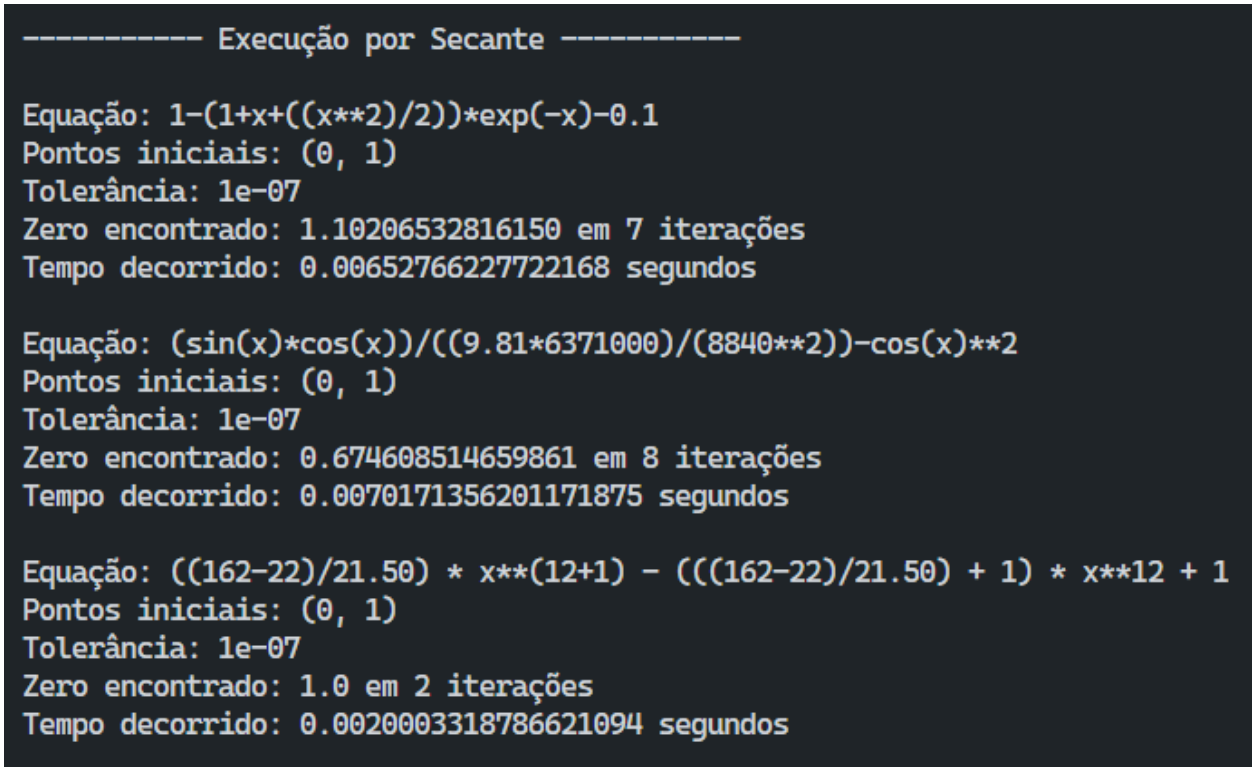

Imagem 4.2 - Retornos gerais das iterações do algoritmo

Até agora, o método da secante encontrou os resultados com o menor número de iterações.

Dificuldades enfrentadas

Não houveram dificuldades.

Método 5 (Eliminação de Gauss)

Estratégia de Implementação

O Método da Eliminação de Gauss é usado para resolver sistemas de equações lineares. O algoritmo inicia combinando a matriz de coeficientes A com o vetor de constantes b, formando uma matriz ampliada. Em seguida, realiza etapas de eliminação para simplificar o sistema:

1. **Normalização da Linha do Pivô:** O algoritmo encontra um elemento de referência, o "pivô", normalizando a linha do pivô.
2. **Eliminação dos Elementos Abaixo do Pivô:** Os elementos abaixo do pivô são zerados.
3. **Resolução das Equações:** A matriz resultante é triangular superior, facilitando a resolução das equações por substituição retroativa.

Após o processo, o algoritmo retorna os valores das variáveis desconhecidas X, que são as soluções do sistema.

Estrutura dos Arquivos de Entrada/Saída

Para cada sistema de equações, deve ser criado um arquivo de entrada, deve chamá-lo de “**system_{index}.csv**”, e deve colocá-lo na pasta “**entradas/eliminacao_gauss/**”. O arquivo deve ser separado por vírgulas, e ter os seguintes cabeçalhos:

- **X_1, X_2, X_3, ..., X_n:** Essas colunas representam os coeficientes das variáveis na matriz de coeficientes X. Se você estiver resolvendo um sistema com N variáveis, deve haver N colunas para os coeficientes. Os números reais devem ser escritos com ponto, por exemplo, "5.5".
- **b:** Esta coluna representa o vetor de constantes do sistema de equações. Deve estar alinhada com as equações correspondentes aos coeficientes na matriz X.

Aqui está um exemplo com uma matriz 3x3:

X_1	X_2	X_3	b
8	-4	-2	5
-4	6	-2	0
-2	-2	10	0

Aqui está um exemplo com uma matriz 4x4:

X_1	X_2	X_3	X_4	b
8	-4	-2	8	5
-4	6	-2	-3	0
-2	-2	10	-8	0
1	2	3	4	-1

Certifique-se de que seu arquivo de entrada corresponde à dimensão do sistema de equações que você deseja resolver, seja 3x3, 4x4 ou qualquer outra. Isso tornará a entrada mais dinâmica e adaptável às suas necessidades.

Os resultados obtidos ao resolver sistemas de equações lineares usando o Método da Eliminação de Gauss são registrados em um arquivo de saída na pasta correspondente "**saidas/eliminacao_gauss/**", como **"result_{index}.csv"**.

O arquivo de saída possui as seguintes informações:

- **X_1, X_2, X_3, ..., X_n:** Contém os valores calculados das variáveis desconhecidas do sistema. Se você estiver resolvendo um sistema com N variáveis, haverá N colunas para os valores.

Aqui está um exemplo com uma matriz 3x3:

X_1	X_2	X_3
2.7586206896551726	2.310344827586207	1.4137931034482758

Problemas teste

Foram realizados testes utilizando os princípios e técnicas descritos no livro **"Cálculo Numérico" de Neide Franco**. A implementação foi avaliada em relação a três problemas

apresentados no livro. Os resultados obtidos foram consistentes com métodos amplamente reconhecidos, evidenciando a eficácia da implementação.

As questões escolhidas foram:

- **4.1** da página 150
- **4.3** da página 151
- **4.6** da página 153

```
---------- Execução por EliminaçãoGauss ----------

Sistema de equações (1):
   X_1  X_2   X_3     b
0  8.0 -4.0  -2.0  10.0
1 -4.0  6.0  -2.0   0.0
2 -2.0 -2.0  10.0   4.0
Os valores de X são: [2.75862069, 2.31034483, 1.41379310]
Tempo decorrido: 0.005992174414855957 segundos

Sistema de equações (2):
   X_1  X_2  X_3  X_4  X_5  X_6  X_7  X_8  X_9    b
0  4.0 -1.0  0.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0
1 -1.0  4.0 -1.0  0.0 -1.0  0.0  0.0  0.0  0.0  0.0
2  0.0 -1.0  4.0  0.0  0.0 -1.0  0.0  0.0  0.0  0.0
3 -1.0  0.0  0.0  4.0 -1.0  0.0 -1.0  0.0  0.0  0.0
4  0.0 -1.0  0.0 -1.0  4.0 -1.0  0.0 -1.0  0.0  0.0
5  0.0  0.0 -1.0  0.0 -1.0  4.0  0.0  0.0 -1.0  0.0
6  0.0  0.0  0.0 -1.0  0.0  0.0  4.0 -1.0  0.0  1.0
7  0.0  0.0  0.0  0.0 -1.0  0.0 -1.0  4.0 -1.0  1.0
8  0.0  0.0  0.0  0.0  0.0 -1.0  0.0 -1.0  4.0  1.0
Os valores de X são: [0.07142857, 0.09821429, 0.07142857, 0.18750000, 0.25000000, 0.18750000,
 0.42857143, 0.52678571, 0.42857143]
Tempo decorrido: 0.009457588195800781 segundos

Sistema de equações (3):
    X_1  X_2   X_3      b
0  14.0  4.0   4.0  100.0
1   4.0  7.0  19.0  100.0
2   4.0  7.0  18.0  100.0
Os valores de X são: [3.65853659, 12.19512195, -0.00000000]
Tempo decorrido: 0.005110979080200195 segundos

Sistema de equações (4):
   X_1  X_2   X_3      b
0  3.0 -0.1  -0.2   7.85
1 -0.1  7.0  -0.3 -19.30
2  0.3 -0.2  10.0  71.40
Os valores de X são: [3.00296938, -2.41417355, 7.00162745]
Tempo decorrido: 0.0036306381225585938 segundos
```

Imagem 5.1 - Retornos gerais das iterações do algoritmo

Dificuldades enfrentadas

A maior dificuldade foi conseguir ler as colunas de forma dinamicamente, de acordo com a quantidade de coeficientes.

Método 6 (Fatoração LU)

Estratégia de Implementação

O algoritmo recebe como parâmetro uma matriz de coeficientes e um vetor de constantes b. Em seguida, realiza as seguintes etapas:

1. **Fatoração LU:** Primeiramente, a matriz de coeficientes ***A*** é decomposta em duas matrizes: uma matriz triangular inferior ***L*** e uma matriz triangular superior ***U***. A matriz ***L*** possui todos os elementos iguais a 1 na diagonal principal e os elementos não diagonais representam os multiplicadores usados para zerar os elementos abaixo da diagonal principal. A matriz ***U*** contém os elementos da diagonal principal de ***A*** e os elementos acima da diagonal principal.
2. **Resolução de Ly = b:** Para encontrar o vetor ***y***, resolvemos o sistema triangular inferior **Ly = b** usando substituição progressiva. Iniciamos com y_1 e calculamos os valores subsequentes usando as entradas de ***L*** e ***b***.
3. **Resolução de Ux = y**: Em seguida, resolvemos o sistema triangular superior **Ux = y** usando substituição regressiva. Começamos com ***xn*** e calculamos os valores anteriores usando as entradas de ***U*** e ***y***.
4. **Resultado Final:** O vetor ***x*** resultante contém as soluções do sistema de equações lineares original.

Após o processo, o algoritmo retorna os valores das variáveis desconhecidas X, que são as soluções do sistema.

Estrutura dos Arquivos de Entrada/Saída

Para cada sistema de equações, deve ser criado um arquivo de entrada, deve chamá-lo de “**system_{index}.csv**”, e deve colocá-lo na pasta “**entradas/lu/**”. O arquivo deve ser separado por vírgulas, e ter os seguintes cabeçalhos:

- **X_1, X_2, X_3, ..., X_n:** Essas colunas representam os coeficientes das variáveis na matriz de coeficientes X. Se você estiver resolvendo um sistema com N variáveis, deve haver N colunas para os coeficientes. Os números reais devem ser escritos com ponto, por exemplo, "5.5".
- **b:** Esta coluna representa o vetor de constantes do sistema de equações. Deve estar alinhada com as equações correspondentes aos coeficientes na matriz X.

Aqui está um exemplo com uma matriz 3x3:

X_1	X_2	X_3	b
8	-4	-2	5
-4	6	-2	0
-2	-2	10	0

Aqui está um exemplo com uma matriz 4x4:

X_1	X_2	X_3	X_4	b
8	-4	-2	8	5
-4	6	-2	-3	0
-2	-2	10	-8	0
1	2	3	4	-1

Certifique-se de que seu arquivo de entrada corresponde à dimensão do sistema de equações que você deseja resolver, seja 3x3, 4x4 ou qualquer outra. Isso tornará a entrada mais dinâmica e adaptável às suas necessidades.

Os resultados obtidos ao resolver sistemas de equações lineares usando o Método da Fatoração LU são registrados em um arquivo de saída na pasta correspondente “**saidas/lu/**”, como **"result_{index}.csv"**.

O arquivo de saída possui as seguintes informações:

- **X_1, X_2, X_3, ..., X_n:** Contém os valores calculados das variáveis desconhecidas do sistema. Se você estiver resolvendo um sistema com N variáveis, haverá N colunas para os valores.

Aqui está um exemplo com uma matriz 3x3:

X_1	X_2	X_3
2.7586206896551726	2.310344827586207	1.4137931034482758

Problemas teste

Foram realizados testes utilizando os princípios e técnicas descritos no livro **"Cálculo Numérico" de Neide Franco**. A implementação foi avaliada em relação a três problemas

apresentados no livro. Os resultados obtidos foram consistentes com métodos amplamente reconhecidos, evidenciando a eficácia da implementação.

As questões escolhidas foram:

- **4.1** da página 150
- **4.3** da página 151
- **4.6** da página 153

```
---------- Execução por LU ----------

Sistema de equações (1):
   X_1  X_2   X_3     b
0  8.0 -4.0 -2.00  10.0
1 -4.0  4.0 -3.00   0.0
2 -2.0 -3.0  7.25   4.0
Os valores de X são: [2.75862069, 2.31034483, 1.41379310]
Tempo decorrido: 0.004193067550659 18 segundos

Sistema de equações (2):
   X_1   X_2       X_3       X_4       X_5       X_6       X_7       X_8       X_9    b
0  4.0 -1.00  0.000000 -1.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.0
1 -1.0  3.75 -1.000000 -0.250000 -1.000000  0.000000  0.000000  0.000000  0.000000  0.0
2  0.0 -1.00  3.733333 -0.066667 -0.266667 -1.000000  0.000000  0.000000  0.000000  0.0
3 -1.0 -0.25 -0.066667  3.732143 -1.071429 -0.017857 -1.000000  0.000000  0.000000  0.0
4  0.0 -1.00 -0.266667 -1.071429  3.406699 -1.076555 -0.287081 -1.000000  0.000000  0.0
5  0.0  0.00 -1.000000 -0.017857 -1.076555  3.391854 -0.095506 -0.316011 -1.000000  0.0
6  0.0  0.00  0.000000 -1.000000 -0.287081 -0.095506  3.705176 -1.093168 -0.028157  1.0
7  0.0  0.00  0.000000  0.000000 -1.000000 -0.316011 -1.093168  3.354493 -1.101475  1.0
8  0.0  0.00  0.000000  0.000000  0.000000 -1.000000 -0.028157 -1.101475  3.343284  1.0
Os valores de X são: [0.07142857, 0.09821429, 0.07142857, 0.18750000, 0.25000000, 0.18750000,
 0.42857143, 0.52678571, 0.42857143]
Tempo decorrido: 0.009988546371459961 segundos

Sistema de equações (3):
    X_1       X_2       X_3      b
0  14.0  4.000000   4.000000  100.0
1   4.0  5.857143  17.857143  100.0
2   4.0  5.857143  -1.000000  100.0
Os valores de X são: [3.65853659, 12.19512195, -0.00000000]
Tempo decorrido: 0.0029964447021484375 segundos
```

Imagem 6.1 - Retornos gerais das iterações do algoritmo

Como podemos perceber, os resultados do método de LU foram os mesmos do de Gauss, porém ele demorou um pouco menos para entregar os resultados, sendo assim, mais performático.

Dificuldades enfrentadas

Não houveram dificuldades.

Método 7 (Jacobi)

Estratégia de Implementação

O algoritmo recebe como entrada a matriz de coeficientes A, o vetor de constantes b.

Após isso, segue para os seguintes passos:

1. **Inicialização:** O vetor ***X*** é inicializado com a aproximação inicial ***X(0)***. Também é criado um vetor ***X_new*** para armazenar os novos valores de ***X*** a cada iteração.
2. **Iteração Principal:** Um loop é iniciado com o contador k variando de 0 até o número máximo de iterações. Dentro deste loop é onde acontecem as estratégias do método.

Por fim, o vetor **X** resultante contém as soluções aproximadas do sistema de equações lineares original. O algoritmo também retorna o número de iterações realizadas.

Estrutura dos Arquivos de Entrada/Saída

Para cada sistema de equações, deve ser criado um arquivo de entrada, deve chamá-lo de "**system_{index}.csv**", e deve colocá-lo na pasta "**entradas/jacobi/**". O arquivo deve ser separado por vírgulas, e ter os seguintes cabeçalhos:

- **X_1, X_2, X_3, ..., X_n:** Essas colunas representam os coeficientes das variáveis na matriz de coeficientes X. Se você estiver resolvendo um sistema com N variáveis, deve haver N colunas para os coeficientes. Os números reais devem ser escritos com ponto, por exemplo, "5.5".
- **b:** Esta coluna representa o vetor de constantes do sistema de equações. Deve estar alinhada com as equações correspondentes aos coeficientes na matriz X.

Aqui está um exemplo com uma matriz 3x3:

X_1	X_2	X_3	b
8	-4	-2	5
-4	6	-2	0
-2	-2	10	0

Aqui está um exemplo com uma matriz 4x4:

X_1	X_2	X_3	X_4	b
8	-4	-2	8	5
-4	6	-2	-3	0
-2	-2	10	-8	0
1	2	3	4	-1

Certifique-se de que seu arquivo de entrada corresponde à dimensão do sistema de equações que você deseja resolver, seja 3x3, 4x4 ou qualquer outra. Isso tornará a entrada mais dinâmica e adaptável às suas necessidades.

Os resultados obtidos ao resolver sistemas de equações lineares usando o Método de Jacobi são registrados em um arquivo de saída na pasta correspondente "**saidas/jacobi/**", como **"result_{index}.csv"**.

O arquivo de saída possui as seguintes informações:

- **X_1, X_2, X_3, ..., X_n:** Contém os valores calculados das variáveis desconhecidas do sistema. Se você estiver resolvendo um sistema com N variáveis, haverá N colunas para os valores.

Aqui está um exemplo com uma matriz 3x3:

X_1	X_2	X_3
2.7586206896551726	2.310344827586207	1.4137931034482758

Problemas teste

Foram realizados testes utilizando os princípios e técnicas descritos no livro **"Cálculo Numérico" de Neide Franco**. A implementação foi avaliada em relação a três problemas apresentados no livro. Os resultados obtidos foram consistentes com métodos amplamente reconhecidos, evidenciando a eficácia da implementação.

As questões escolhidas foram:

- **5.1** da página 185
- **5.2** da página 185
- **5.5** da página 188

```
---------- Execução por Jacobi ----------

Sistema de equações (1):
   X_1  X_2  X_3  X_4  X_5  X_6      b
0  4.0 -1.0  0.0 -1.0  0.0  0.0  100.0
1 -1.0  4.0 -1.0  0.0 -1.0  0.0    0.0
2  0.0 -1.0  4.0  0.0  0.0 -1.0    0.0
3 -1.0  0.0  0.0  4.0 -1.0  0.0  100.0
4  0.0 -1.0  0.0 -1.0  4.0 -1.0    0.0
5  0.0  0.0 -1.0  0.0 -1.0  4.0    0.0
Os valores de X são: [38.09523810, 14.28571429, 4.76190476, 38.09523810, 14.28571429,
 4.76190476]
Tempo decorrido: 0.008265256881713867 segundos

Sistema de equações (2):
    X_1   X_2   X_3     b
0  20.0 -10.0  -4.0  26.0
1 -10.0  25.0  -5.0   0.0
2  -4.0  -5.0  20.0   7.0
Os valores de X são: [2.00000000, 1.00000000, 1.00000000]
Tempo decorrido: 0.005000591278076172 segundos

Sistema de equações (3):
   X_1  X_2  X_3  X_4  X_5  X_6  X_7  X_8  X_9      b
0 -4.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  -50.0
1  1.0 -4.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  -50.0
2  0.0  1.0 -4.0  0.0  0.0  1.0  0.0  0.0  0.0 -150.0
3  1.0  0.0  0.0 -4.0  1.0  0.0  1.0  0.0  0.0    0.0
4  0.0  1.0  0.0  1.0 -4.0  1.0  0.0  1.0  0.0    0.0
5  0.0  0.0  1.0  0.0  1.0 -4.0  0.0  0.0  1.0 -100.0
6  0.0  0.0  0.0  0.0  1.0  0.0 -4.0  1.0  0.0  -50.0
7  0.0  0.0  0.0  0.0  1.0  0.0  1.0 -4.0  1.0  -50.0
8  0.0  0.0  0.0  0.0  0.0  1.0  0.0  1.0 -4.0 -150.0
Os valores de X são: [32.78940887, 50.55418719, 68.13423645, 30.60344828, 51.29310345,
 71.98275862, 38.33128079, 52.03201970, 68.50369458]
Tempo decorrido: 0.017011642456054688 segundos
```

Imagem 7.1 - Retornos gerais das iterações do algoritmo

Dificuldades enfrentadas

Não houveram dificuldades.

Método 8 (Gauss-Seidel)

Estratégia de Implementação

O algoritmo recebe como entrada a matriz de coeficientes A, o vetor de constantes b.

Após isso, segue para os seguintes passos:

1. **Inicialização:** O vetor ***X*** é inicializado com a aproximação inicial ***X(0)***. Também é criado um vetor ***X_new*** para armazenar os novos valores de ***X*** a cada iteração.
2. **Iteração Principal:** Um loop é iniciado com o contador k variando de 0 até o número máximo de iterações. Dentro deste loop é onde acontecem as estratégias do método.
    a. **Aqui ocorre a diferença com o método de Jacobi**, onde no método Gauss-Seidel, o ***x_new*** é utilizado imediatamente após as variáveis serem atualizadas.

Por fim, o vetor **X** resultante contém as soluções aproximadas do sistema de equações lineares original. O algoritmo também retorna o número de iterações realizadas.

Estrutura dos Arquivos de Entrada/Saída

Para cada sistema de equações, deve ser criado um arquivo de entrada, deve chamá-lo de "**system_{index}.csv**", e deve colocá-lo na pasta "**entradas/gauss_seidel/**". O arquivo deve ser separado por vírgulas, e ter os seguintes cabeçalhos:

- **X_1, X_2, X_3, ..., X_n:** Essas colunas representam os coeficientes das variáveis na matriz de coeficientes X. Se você estiver resolvendo um sistema com N variáveis, deve haver N colunas para os coeficientes. Os números reais devem ser escritos com ponto, por exemplo, "5.5".
- **b:** Esta coluna representa o vetor de constantes do sistema de equações. Deve estar alinhada com as equações correspondentes aos coeficientes na matriz X.

Aqui está um exemplo com uma matriz 3x3:

X_1	X_2	X_3	b
8	-4	-2	5
-4	6	-2	0

-2	-2	10	0

Aqui está um exemplo com uma matriz 4x4:

X_1	X_2	X_3	X_4	b
8	-4	-2	8	5
-4	6	-2	-3	0
-2	-2	10	-8	0
1	2	3	4	-1

Certifique-se de que seu arquivo de entrada corresponde à dimensão do sistema de equações que você deseja resolver, seja 3x3, 4x4 ou qualquer outra. Isso tornará a entrada mais dinâmica e adaptável às suas necessidades.

Os resultados obtidos ao resolver sistemas de equações lineares usando o Método de Gauss-Seidel são registrados em um arquivo de saída na pasta correspondente "**saidas/gauss_seidel/**", como **"result_{index}.csv"**.

O arquivo de saída possui as seguintes informações:

- **X_1, X_2, X_3, ..., X_n:** Contém os valores calculados das variáveis desconhecidas do sistema. Se você estiver resolvendo um sistema com N variáveis, haverá N colunas para os valores.

Aqui está um exemplo com uma matriz 3x3:

X_1	X_2	X_3
2.7586206896551726	2.310344827586207	1.4137931034482758

Problemas teste

Foram realizados testes utilizando os princípios e técnicas descritos no livro **"Cálculo Numérico" de Neide Franco**. A implementação foi avaliada em relação a três problemas apresentados no livro. Os resultados obtidos foram consistentes com métodos amplamente reconhecidos, evidenciando a eficácia da implementação.

As questões escolhidas foram:

- **5.1** da página 185

- **5.2** da página 185
- **5.5** da página 188

```
---------- Execução por GaussSeidel ----------

Sistema de equações (1):
   X_1  X_2  X_3  X_4  X_5  X_6      b
0  4.0 -1.0  0.0 -1.0  0.0  0.0  100.0
1 -1.0  4.0 -1.0  0.0 -1.0  0.0    0.0
2  0.0 -1.0  4.0  0.0  0.0 -1.0    0.0
3 -1.0  0.0  0.0  4.0 -1.0  0.0  100.0
4  0.0 -1.0  0.0 -1.0  4.0 -1.0    0.0
5  0.0  0.0 -1.0  0.0 -1.0  4.0    0.0
Os valores de X são: [38.09523810, 14.28571429, 4.76190476, 38.09523810, 14.28571429,
 4.76190476]
Tempo decorrido: 0.008675813674926758 segundos

Sistema de equações (2):
    X_1   X_2   X_3     b
0  20.0 -10.0  -4.0  26.0
1 -10.0  25.0  -5.0   0.0
2  -4.0  -5.0  20.0   7.0
Os valores de X são: [2.00000000, 1.00000000, 1.00000000]
Tempo decorrido: 0.0039861202239990234 segundos

Sistema de equações (3):
   X_1  X_2  X_3  X_4  X_5  X_6  X_7  X_8  X_9      b
0 -4.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  -50.0
1  1.0 -4.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  -50.0
2  0.0  1.0 -4.0  0.0  0.0  1.0  0.0  0.0  0.0 -150.0
3  1.0  0.0  0.0 -4.0  1.0  0.0  1.0  0.0  0.0    0.0
4  0.0  1.0  0.0  1.0 -4.0  1.0  0.0  1.0  0.0    0.0
5  0.0  0.0  1.0  0.0  1.0 -4.0  0.0  0.0  1.0 -100.0
6  0.0  0.0  0.0  0.0  1.0  0.0 -4.0  1.0  0.0  -50.0
7  0.0  0.0  0.0  0.0  1.0  0.0  1.0 -4.0  1.0  -50.0
8  0.0  0.0  0.0  0.0  0.0  1.0  0.0  1.0 -4.0 -150.0
Os valores de X são: [32.78940887, 50.55418719, 68.13423645, 30.60344828, 51.29310345,
 71.98275862, 38.33128079, 52.03201970, 68.50369458]
Tempo decorrido: 0.012001752853393555 segundos
```

Imagem 8.1 - Retornos gerais das iterações do algoritmo

Como podemos perceber, o método de Gauss-Seidel foi mais rápido do que o de Jacobi.

Dificuldades enfrentadas

Não houveram dificuldades.