Universidade Estadual de Santa Cruz – UESC

Relatórios de Implementações de Métodos da Disciplina Análise Numérica

Relatório de implementações realizadas por Luiz Felipe do Rosário Alves Silva

Disciplina Análise Numérica.

Curso Ciência da Computação

Semestre 2025.1

Professor Gesil Sampaio Amarante II

Ilhéus – BA
2025

Índice

Linguagem(ns) Escolhida(s) e justificativas

Python foi escolhido devido à sua simplicidade, versatilidade e grande suporte de bibliotecas, como NumPy, que facilita o trabalho com dados numéricos e operações matemáticas. Sua sintaxe clara e direta contribui para um desenvolvimento rápido e eficiente.

Link - Python

GeoGebra foi utilizado para criar gráficos e visualizar funções, permitindo uma melhor análise e compreensão dos dados. Esta ferramenta é especialmente útil para representar graficamente funções matemáticas e visualizar interações numéricas de forma intuitiva.

Link - CodeSnap

VS Code foi escolhido como o ambiente de desenvolvimento devido à sua leveza, alta personalização e excelente suporte a extensões. Ele oferece ferramentas poderosas para codificação, depuração e controle de versões, tornando o desenvolvimento mais produtivo e organizado.

Link - VS Code

CodeSnap foi utilizado para capturar prints do código de maneira organizada e visualmente atraente, facilitando a documentação do projeto. Esta ferramenta é útil para criar imagens de trechos de código para incluir em relatórios e apresentações.

Link - CodeSnap

Bissecção

Estratégia de Implementação:

Para receber as expressões, utilizamos a função eval() com a variável x definida no escopo. Esse foi o método mais simples e facilitado para as funções pois o eval já está na base do Python.

Para o critério de parada, seguimos para o valor absoluto de f(x) menor ou igual à tolerância e o comprimento do intervalo menor que a tolerância. Pois, com ambos, podemos ter um melhor aproveitamento do código sem adaptações conforme outros testes.

Primeiro o programa verifica se tem mudança de sinal no intervalo inicial. Caso não, lança um erro informando que o intervalo é inválido.

Para evitar divisão por zero no cálculo de (b-a)/a, foi aplicada uma condição que zera o valor quando a = 0.

A saída foi registrada em arquivo .txt, com uma tabela organizada contendo os valores de cada iteração, e a raiz aproximada no final.

Estrutura dos Arquivos de Entrada/Saída

Entrada: bisseccao-fun.txt

1. Função adaptada como String para ser reconhecida pelo eval() do próprio Python;
2. Intervalo de pontos de X onde possivelmente está uma das raízes. Separados por espaço somente. Podendo ser Int ou Float;
3. Tolerância (Sendo válido para valor absoluto e comprimento do intervalo, o que se aproximar primeiro).

Saída: bisseccao-res.txt

1. Cabeçalho:
   - a e b: Limites atuais do intervalo de busca;
   - f(a) e f(b): Valores da função nos extremos;
   - (b - a): Tamanho do intervalo;
   - (b - a)/a: Taxa relativa de redução do intervalo;
   - x: Ponto médio (candidato à raiz);
   - f(x): Valor da função no ponto médio.
2. Raiz aproximada com 6 casas decimais por causa da flutuação referente à tolerância escolhida.

Problema teste 3.3

O problema envolve calcular o tempo de subida de um amplificador RC, resolvendo numericamente equações que determinam quando a resposta atinge 10% e 90% do valor final.

$$g(T) = 1 - (1 + T + \frac{T^2}{2})e^{-T} \ ,$$

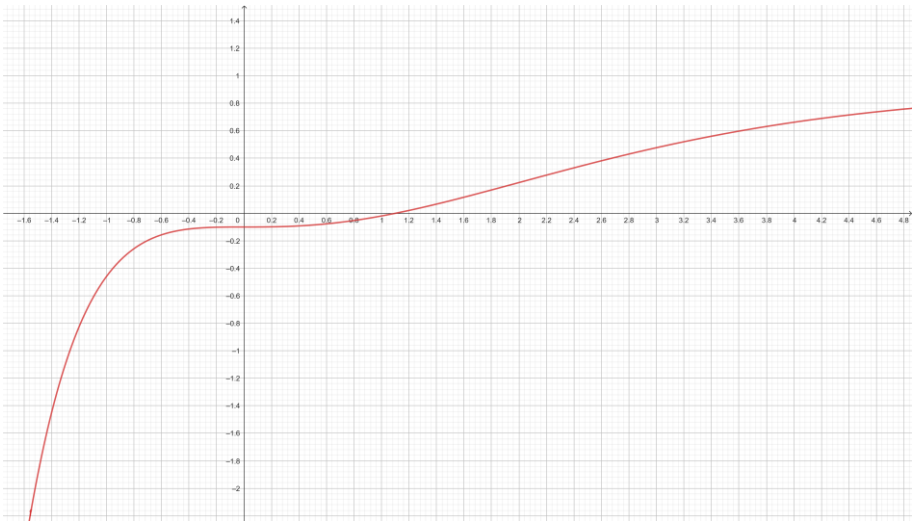
$$0.1 = 1 - (1 + T + \frac{T^2}{2})e^{-T} \ .$$

$$0.9 = 1 - (1 + T + \frac{T^2}{2})e^{-T} \ .$$

Função (10%):

1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.1, onde X = T.

Geogebra:



Entrada:

```
1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.1

1.1 1.2

0.000001
```

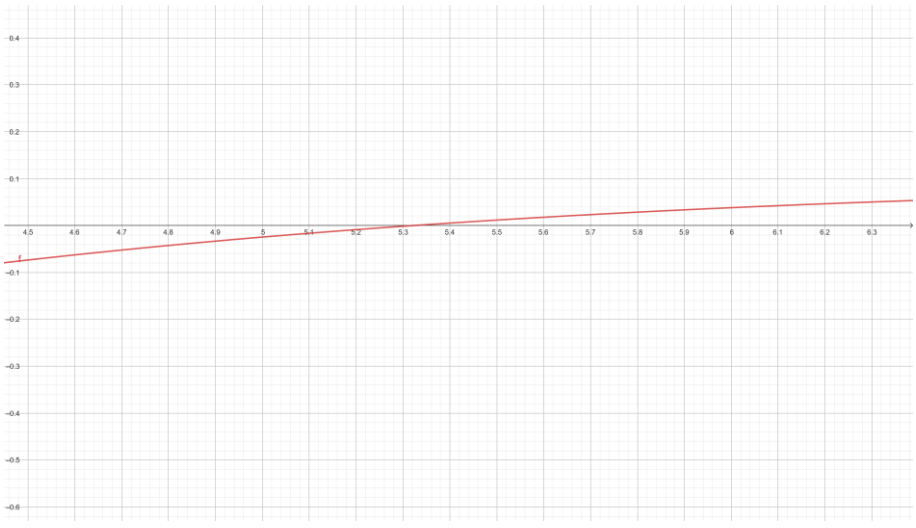
Saída:

```
a          b          f(a)       f(b)       (b-a)      (b-a)/a    x          f(x)

1.100000   1.200000   -0.000416  0.020513   0.100000   0.090909   1.150000   0.009855

1.100000   1.150000   -0.000416  0.009855   0.050000   0.045455   1.125000   0.004669

1.100000   1.125000   -0.000416  0.004669   0.025000   0.022727   1.112500   0.002114

1.100000   1.112500   -0.000416  0.002114   0.012500   0.011364   1.106250   0.000846

1.100000   1.106250   -0.000416  0.000846   0.006250   0.005682   1.103125   0.000214

1.100000   1.103125   -0.000416  0.000214   0.003125   0.002841   1.101562   -0.00010
1
1.101562   1.103125   -0.000101  0.000214   0.001563   0.001418   1.102344   0.000056

1.101562   1.102344   -0.000101  0.000056   0.000781   0.000709   1.101953   -0.00002
3
1.101953   1.102344   -0.000023  0.000056   0.000391   0.000354   1.102148   0.000017

1.101953   1.102148   -0.000023  0.000017   0.000195   0.000177   1.102051   -0.00000
3
1.102051   1.102148   -0.000003  0.000017   0.000098   0.000089   1.102100   0.000007

1.102051   1.102100   -0.000003  0.000007   0.000049   0.000044   1.102075   0.000002

1.102051   1.102075   -0.000003  0.000002   0.000024   0.000022   1.102063   -0.00000
0
1.102063   1.102075   -0.000000  0.000002   0.000012   0.000011   1.102069   0.000001

1.102063   1.102069   -0.000000  0.000001   0.000006   0.000006   1.102066   0.000000

1.102063   1.102066   -0.000000  0.000000   0.000003   0.000003   1.102065   -0.00000
0
1.102065   1.102066   -0.000000  0.000000   0.000002   0.000001   1.102065   -0.00000
0


Raiz aproximada = 1.102065
```

Função (90%):

1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.9, onde X = T.

Geogebra:



Entrada:

```
1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.9

5.3 5.4

0.000001
```

Saída:

a	b	f(a)	f(b)	(b-a)	(b-a)/a	x	f(x)
5.300000	5.400000	-0.001554	0.005242	0.100000	0.018868	5.350000	0.001897
5.300000	5.350000	-0.001554	0.001897	0.050000	0.009434	5.325000	0.000185
5.300000	5.325000	-0.001554	0.000185	0.025000	0.004717	5.312500	-0.000681
5.312500	5.325000	-0.000681	0.000185	0.012500	0.002353	5.318750	-0.000247
5.318750	5.325000	-0.000247	0.000185	0.006250	0.001175	5.321875	-0.000031
5.321875	5.325000	-0.000031	0.000185	0.003125	0.000587	5.323437	0.000077
5.321875	5.323437	-0.000031	0.000077	0.001562	0.000294	5.322656	0.000023
5.321875	5.322656	-0.000031	0.000023	0.000781	0.000147	5.322266	-0.000004
5.322266	5.322656	-0.000004	0.000023	0.000391	0.000073	5.322461	0.000010
5.322266	5.322461	-0.000004	0.000010	0.000195	0.000037	5.322363	0.000003
5.322266	5.322363	-0.000004	0.000003	0.000098	0.000018	5.322314	-0.000000

Raiz aproximada = 5.322314

Problema teste 3.6

A questão busca determinar o ângulo de lançamento α\alphaα para que um míssil atinja um alvo, resolvendo uma equação trigonométrica com dados conhecidos.

$$tg\left(\frac{\theta}{2}\right) = \frac{sen\ \alpha\ cos\ \alpha}{\frac{g\ R}{v^2} - cos^2\ \alpha},$$

- α - *ângulo de inclinação com a superfície da Terra com a qual é feita o lançamento do míssil* ,
- g - *aceleração da gravidade* $\simeq 9.81\ m/s^2$,
- R - *raio da Terra* $\simeq 6371000\ m$,
- v - *velocidade de lançamento do míssil,* m/s ,
- θ - *ângulo (medido do centro da Terra) entre o ponto de lançamento e o ponto de impacto desejado* ,

Resolva o problema considerando: $\theta = 80^o$ *e* v *tal que* $\frac{v^2}{gR} = 1.25$, *ou seja, aproximadamente* $8.840\ m/s$.

Função:

(math.sin(x) * math.cos(x)) / (0.8 - math.cos(x)**2) - math.tan(math.radians(40))

Geogebra:

Entrada:

```
(math.sin(x) * math.cos(x)) / (0.8 - math.cos(x)**2) - math.tan(math.radians(4
0))
1 1.05

0.000001
```

Saída:

```
a          b          f(a)       f(b)       (b-a)      (b-a)/a    x          f(x)

1.000000   1.050000   0.055749   -0.057806  0.050000   0.050000   1.025000   -0.00281
2
1.000000   1.025000   0.055749   -0.002812  0.025000   0.025000   1.012500   0.025991

1.012500   1.025000   0.025991   -0.002812  0.012500   0.012346   1.018750   0.011475

1.018750   1.025000   0.011475   -0.002812  0.006250   0.006135   1.021875   0.004303

1.021875   1.025000   0.004303   -0.002812  0.003125   0.003058   1.023438   0.000739

1.023438   1.025000   0.000739   -0.002812  0.001562   0.001527   1.024219   -0.00103
8
1.023438   1.024219   0.000739   -0.001038  0.000781   0.000763   1.023828   -0.00015
0
1.023438   1.023828   0.000739   -0.000150  0.000391   0.000382   1.023633   0.000294

1.023633   1.023828   0.000294   -0.000150  0.000195   0.000191   1.023730   0.000072

1.023730   1.023828   0.000072   -0.000150  0.000098   0.000095   1.023779   -0.00003
9
1.023730   1.023779   0.000072   -0.000039  0.000049   0.000048   1.023755   0.000017

1.023755   1.023779   0.000017   -0.000039  0.000024   0.000024   1.023767   -0.00001
1
1.023755   1.023767   0.000017   -0.000011  0.000012   0.000012   1.023761   0.000003

1.023761   1.023767   0.000003   -0.000011  0.000006   0.000006   1.023764   -0.00000
4
1.023761   1.023764   0.000003   -0.000004  0.000003   0.000003   1.023763   -0.00000
1


Raiz aproximada = 1.023763
```

Problema teste 3.8

O problema compara dois planos de financiamento. A taxa de juros de cada plano é obtida resolvendo uma equação não linear. O plano com menor raiz tem juros mais baixos.

$$f(x) = kx^{P+1} - (k+1)x^P + 1 = 0 \ .$$

3.8 - *Uma loja de eletrodomésticos oferece dois planos de financiamento para um produto cujo preço a vista é R$ 162,00:*

- *Plano A: entrada de R$ 22,00 + 9 prestações iguais de R$ 26,50,*
- *Plano B: entrada de R$ 22,00 + 12 prestações de R$ 21,50.*

Qual dos dois planos apresenta a menor taxa de juros, sendo portanto melhor para o consumidor?

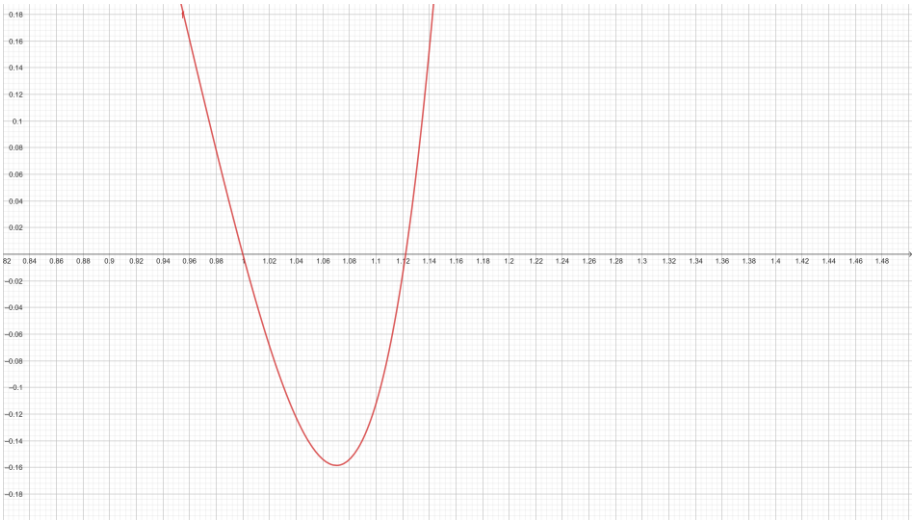
Observação: *Sabe-se que a equação que relaciona os juros* **(J)** *e o prazo* **(P)** *com o valor financiado (***VF** *= preço à vista - entrada) e a prestação mensal* **PM** *é dada por:*

$$\frac{1-(1+J)^{-P}}{J} = \frac{VF}{PM} \ . \qquad (3.33)$$

Função (A):

5.283 * x**10 - 6.283 * x**9 + 1

Geogebra:



Entrada:

```
5.283 * x**10 - 6.283 * x**9 +
1
1.1 1.2

0.000001
```

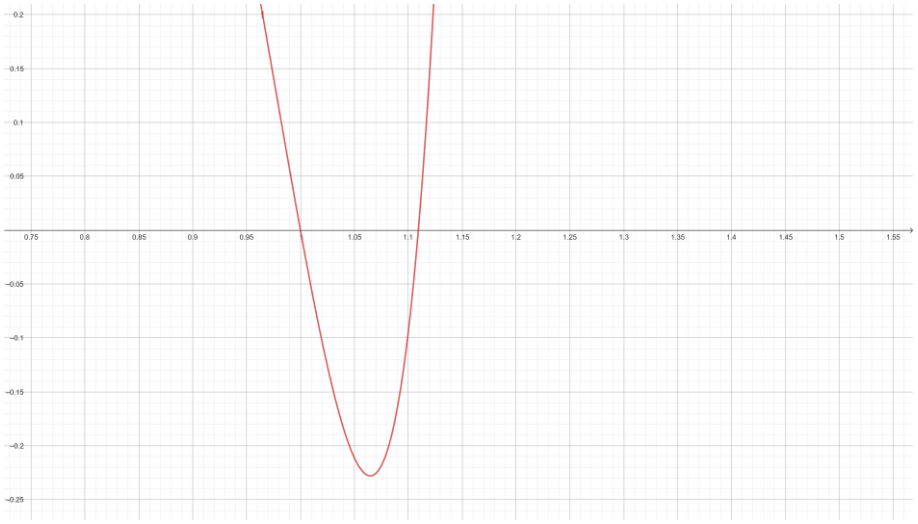
Saída:

```
a          b          f(a)       f(b)       (b-a)      (b-a)/a    x          f(x)

1.100000   1.200000   -0.112244  1.292044   0.100000   0.090909   1.150000   0.269865

1.100000   1.150000   -0.112244  0.269865   0.050000   0.045455   1.125000   0.019670

1.100000   1.125000   -0.112244  0.019670   0.025000   0.022727   1.112500   -0.05892
5
1.112500   1.125000   -0.058925  0.019670   0.012500   0.011236   1.118750   -0.02303
7
1.118750   1.125000   -0.023037  0.019670   0.006250   0.005587   1.121875   -0.00256
9
1.121875   1.125000   -0.002569  0.019670   0.003125   0.002786   1.123438   0.008325

1.121875   1.123438   -0.002569  0.008325   0.001563   0.001393   1.122656   0.002822

1.121875   1.122656   -0.002569  0.002822   0.000781   0.000696   1.122266   0.000113

1.121875   1.122266   -0.002569  0.000113   0.000391   0.000348   1.122070   -0.00123
1
1.122070   1.122266   -0.001231  0.000113   0.000195   0.000174   1.122168   -0.00056
0
1.122168   1.122266   -0.000560  0.000113   0.000098   0.000087   1.122217   -0.00022
4
1.122217   1.122266   -0.000224  0.000113   0.000049   0.000044   1.122241   -0.00005
5
1.122241   1.122266   -0.000055  0.000113   0.000024   0.000022   1.122253   0.000029

1.122241   1.122253   -0.000055  0.000029   0.000012   0.000011   1.122247   -0.00001
3
1.122247   1.122253   -0.000013  0.000029   0.000006   0.000005   1.122250   0.000008

1.122247   1.122250   -0.000013  0.000008   0.000003   0.000003   1.122249   -0.00000
3
1.122249   1.122250   -0.000003  0.000008   0.000002   0.000001   1.122250   0.000002


Raiz aproximada = 1.122250
```

Função (B):

$$6.512 * x^{13} - 7.512 * x^{12} + 1$$

Geogebra:



Entrada:

```
6.512 * x**13 - 7.512 * x**12 +
1
1.1 1.2

0.000001
```

Saída:

```
a          b          f(a)       f(b)       (b-a)      (b-a)/a    x          f(x)

1.100000   1.200000   -0.094684  3.696229   0.100000   0.090909   1.150000   0.875874

1.100000   1.150000   -0.094684  0.875874   0.050000   0.045455   1.125000   0.235560

1.100000   1.125000   -0.094684  0.235560   0.025000   0.022727   1.112500   0.038915

1.100000   1.112500   -0.094684  0.038915   0.012500   0.011364   1.106250   -0.03497
8
1.106250   1.112500   -0.034978  0.038915   0.006250   0.005650   1.109375   0.000102

1.106250   1.109375   -0.034978  0.000102   0.003125   0.002825   1.107813   -0.01789
3
1.107813   1.109375   -0.017893  0.000102   0.001562   0.001410   1.108594   -0.00901
1
1.108594   1.109375   -0.009011  0.000102   0.000781   0.000705   1.108984   -0.00448
4
1.108984   1.109375   -0.004484  0.000102   0.000391   0.000352   1.109180   -0.00219
8
1.109180   1.109375   -0.002198  0.000102   0.000195   0.000176   1.109277   -0.00105
0
1.109277   1.109375   -0.001050  0.000102   0.000098   0.000088   1.109326   -0.00047
5
1.109326   1.109375   -0.000475  0.000102   0.000049   0.000044   1.109351   -0.00018
7
1.109351   1.109375   -0.000187  0.000102   0.000024   0.000022   1.109363   -0.00004
2
1.109363   1.109375   -0.000042  0.000102   0.000012   0.000011   1.109369   0.000030

1.109363   1.109369   -0.000042  0.000030   0.000006   0.000006   1.109366   -0.00000
6
1.109366   1.109369   -0.000006  0.000030   0.000003   0.000003   1.109367   0.000012

1.109366   1.109367   -0.000006  0.000012   0.000002   0.000001   1.109367   0.000003


Raiz aproximada = 1.109367
```

Dificuldades enfrentadas

Uma das maiores dificuldades foi entender bem o enunciado das questões do livro, que exigiam atenção para interpretar corretamente os dados e o que deveria ser resolvido. Também deu um pouco de trabalho montar as expressões certas na linguagem usando eval(). Por fim, organizar a saída em arquivo de forma clara, com todos os dados.

Posição Falsa

Estratégia de Implementação:

As expressões matemáticas foram avaliadas usando eval(). Essa abordagem foi adotada por ser prática e já integrada à linguagem Python, sem necessidade de bibliotecas externas.

O critério de parada foi definido como o valor absoluto de f(c) menor ou igual à tolerância. Isso garante que o valor da função no ponto estimado da raiz esteja suficientemente próximo de zero. O intervalo não é usado como critério de parada nesse caso pois como não temos uma divisão simétrica. Neste caso, é um pouco mais interessante ver a evolução dos valores do intervalo no nosso resultado.

Todos os resultados foram registrados em um arquivo .txt, com uma tabela contendo os valores de cada iteração, e a raiz aproximada ao final com 6 casas decimais, respeitando a tolerância definida.

Estrutura dos Arquivos de Entrada/Saída

Entrada: posicaofalsa-fun.txt

1. Função como string avaliada com eval();
2. Intervalo de busca: dois valores (a e b), separados por espaço;
3. Tolerância.

Saída: posicaofalsa-res.txt

1. Cabeçalho da Tabela:
    - a, b: Intervalo atual;
    - f(a), f(b): Valores da função nos extremos;
    - (b - a): Tamanho do intervalo (só para acompanhamento, não afeta parada);
    - c: Raiz aproximada da iteração (usando interpolação linear);
    - f(c): Valor da função no ponto c.
2. Resultado final:
    - Linha ao final com a raiz aproximada:

Problema teste 3.3

Neste método, os valores de entrada são quase semelhantes. Para fins de análise, apenas a tolerância foi mudada. Para mostrar a diferença em quantidade de iterações.

A para o método anterior era de 0,000001 e a para o atual é de 0,0000001

Função (10%):

1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.1

Entrada:

```
1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.1

1.1 1.2

0.0000001
```

Saída:

```
a           b           f(a)        f(b)        (b-a)       c           f(c)
1.100000    1.200000    -0.000416   0.020513    0.100000    1.101989    -0.000015
1.101989    1.200000    -0.000015   0.020513    0.098011    1.102063    -0.000001
1.102063    1.200000    -0.000001   0.020513    0.097937    1.102065    -0.000000
Raiz aproximada = 1.102065
```

Em f(c) podemos ver que a diferença entre a 1ª, 2ª e 3ª iteração vai se tornando insignificante bem cedo.

Função (90%):

1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.9

Entrada:

```
1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.9

5.3 5.4

0.0000001
```

Saída:

```
a           b           f(a)        f(b)        (b-a)       c           f(c)
5.300000    5.400000    -0.001554   0.005242    0.100000    5.322866    0.000038
5.300000    5.322866    -0.001554   0.000038    0.022866    5.322324    0.000000
5.300000    5.322324    -0.001554   0.000000    0.022324    5.322320    0.000000
Raiz aproximada = 5.322320
```

Problema teste 3.6

Da mesma forma, também na entrada a única diferença é a tolerância.

Entrada:

```
(math.sin(x) * math.cos(x)) / (0.8 - math.cos(x)**2) - math.tan(math.radians(40))
1 1.05
0.0000001
```

Saída:

```
a           b           f(a)        f(b)        (b-a)       c           f(c)
1.000000    1.050000    0.055749    -0.057806   0.050000    1.024547    -0.001784
1.024547    1.050000    -0.001784   -0.057806   0.025453    1.023737    0.000058
1.024547    1.023737    -0.001784   0.000058    -0.000811   1.023762    -0.000000
Raiz aproximada = 1.023762
```

Problema teste 3.8

Nesse caso foi possível notar suficiência ao utilizar tolerância de 0,00001.

Função (A):

5.283 * x**10 - 6.283 * x**9 + 1

Entrada:

```
5.283 * x**10 - 6.283 * x**9 + 1
1.1 1.2
0.00001
```

Saída:

a	b	f(a)	f(b)	(b-a)	c	f(c)
1.100000	1.200000	-0.112244	1.292044	0.100000	1.107993	-0.080859
1.107993	1.200000	-0.080859	1.292044	0.092007	1.113412	-0.054094
1.113412	1.200000	-0.054094	1.292044	0.086588	1.116891	-0.034398
1.116891	1.200000	-0.034398	1.292044	0.083109	1.119047	-0.021169
1.119047	1.200000	-0.021169	1.292044	0.080953	1.120352	-0.012764
1.120352	1.200000	-0.012764	1.292044	0.079648	1.121131	-0.007602
1.121131	1.200000	-0.007602	1.292044	0.078869	1.121592	-0.004494
1.121592	1.200000	-0.004494	1.292044	0.078408	1.121864	-0.002645
1.121864	1.200000	-0.002645	1.292044	0.078136	1.122023	-0.001553
1.122023	1.200000	-0.001553	1.292044	0.077977	1.122117	-0.000910
1.122117	1.200000	-0.000910	1.292044	0.077883	1.122172	-0.000533
1.122172	1.200000	-0.000533	1.292044	0.077828	1.122204	-0.000312
1.122204	1.200000	-0.000312	1.292044	0.077796	1.122223	-0.000183
1.122223	1.200000	-0.000183	1.292044	0.077777	1.122234	-0.000107
1.122234	1.200000	-0.000107	1.292044	0.077766	1.122240	-0.000063
1.122240	1.200000	-0.000063	1.292044	0.077760	1.122244	-0.000037
1.122244	1.200000	-0.000037	1.292044	0.077756	1.122246	-0.000021
1.122246	1.200000	-0.000021	1.292044	0.077754	1.122247	-0.000013
1.122247	1.200000	-0.000013	1.292044	0.077753	1.122248	-0.000007

Raiz aproximada = 1.122248

Aqui podemos notar a evolução de '**c**' nas iterações, e como a diferença vem diminuindo ao decorrer delas.

Função (B):

6.512 * x**13 - 7.512 * x**12 + 1

Entrada:

```
6.512 * x**13 - 7.512 * x**12 + 1

1.1 1.2

0.0001
```

Saída:

```
a           b           f(a)        f(b)        (b-a)       c           f(c)
1.100000    1.200000    -0.094684   3.696229    0.100000    1.102498    -0.072432
1.102498    1.200000    -0.072432   3.696229    0.097502    1.104372    -0.054346
1.104372    1.200000    -0.054346   3.696229    0.095628    1.105757    -0.040181
1.105757    1.200000    -0.040181   3.696229    0.094243    1.106771    -0.029384
1.106771    1.200000    -0.029384   3.696229    0.093229    1.107506    -0.021315
1.107506    1.200000    -0.021315   3.696229    0.092494    1.108036    -0.015371
1.108036    1.200000    -0.015371   3.696229    0.091964    1.108417    -0.011038
1.108417    1.200000    -0.011038   3.696229    0.091583    1.108690    -0.007902
1.108690    1.200000    -0.007902   3.696229    0.091310    1.108885    -0.005644
1.108885    1.200000    -0.005644   3.696229    0.091115    1.109024    -0.004026
1.109024    1.200000    -0.004026   3.696229    0.090976    1.109123    -0.002868
1.109123    1.200000    -0.002868   3.696229    0.090877    1.109193    -0.002041
1.109193    1.200000    -0.002041   3.696229    0.090807    1.109243    -0.001452
1.109243    1.200000    -0.001452   3.696229    0.090757    1.109279    -0.001033
1.109279    1.200000    -0.001033   3.696229    0.090721    1.109304    -0.000734
1.109304    1.200000    -0.000734   3.696229    0.090696    1.109322    -0.000522
1.109322    1.200000    -0.000522   3.696229    0.090678    1.109335    -0.000371
1.109335    1.200000    -0.000371   3.696229    0.090665    1.109344    -0.000264
1.109344    1.200000    -0.000264   3.696229    0.090656    1.109351    -0.000187
1.109351    1.200000    -0.000187   3.696229    0.090649    1.109355    -0.000133
1.109355    1.200000    -0.000133   3.696229    0.090645    1.109358    -0.000095
Raiz aproximada = 1.109358
```

Dificuldades enfrentadas

A implementação do método da posição falsa foi relativamente tranquila, já que grande parte da estrutura foi reaproveitada do método da bissecção. As principais mudanças ficaram por conta da fórmula do ponto estimado e do critério de parada.

Newton-Raphson

Estratégia de Implementação:

É um método iterativo baseado em derivadas. Ele começa com um chute inicial e usa a seguinte fórmula para refinar o valor de x:

x = x - (eval(expressao) / eval(derivada))

É necessário fornecer a derivada da função porque o método depende diretamente dela para saber em que direção e com que intensidade ajustar o valor de x.

Novamente o uso da função eval() para interpretar tanto a função quanto sua derivada diretamente como strings.

O critério de parada foi o f(x). Pois, como é um método rápido, essa condição já garante alta precisão em poucos passos. Já que, ela não precisa dos dois pontos e pode avançar rapidamente em direção à raiz.

Estrutura dos Arquivos de Entrada/Saída

Entrada: newtonraphson-fun.txt

1. Função representada como string;
2. Derivada da função, também em string;
3. Intervalo [a, b] para chute inicial o programa começa pelo ponto médio.
4. Tolerância.

Saída: newtonraphson-res.txt

1. Cabeçalho:
    - x_n: Valor atual da iteração;
    - f(x_n): Valor da função nesse ponto.
2. Raiz Aproximada.

Problema teste 3.3

Entrada (10%):

```
1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.1
(x**2 / 2) * math.exp(-x)
1.1 1.2
0.00001
```

Saída:

```
 x_n              f(x_n)
 1.150000         0.009855
 1.102932         0.000175
 1.102066         0.000000
 Raiz aproximada = 1.102066
```

Entrada (90%):

```
1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.9
(x**2 / 2) * math.exp(-x)
5.3 5.4
0.000001
```

Saída:

```
x_n                f(x_n)
5.350000           0.001897
5.322079           -0.000017
5.322320           -0.000000
Raiz aproximada = 5.322320
```

Problema teste 3.6

Entrada:

```
(math.sin(x) * math.cos(x)) / (0.8 - math.cos(x)**2) - math.tan(math.radians(40))
((math.cos(x)**2 - math.sin(x)**2) * (0.8 - math.cos(x)**2) - 2 * math.sin(x)**2 * math.cos(x)**2) / (0.8 - math.cos(x)**2)**2
1 1.05
0.0001
```

Saída:

```
x_n                f(x_n)
5.350000           0.001897
5.322079           -0.000017
5.322320           -0.000000
Raiz aproximada = 5.322320
```

Problema teste 3.8

Entrada (A):

```
5.283 * x**10 - 6.283 * x**9 + 1
52.83 * x**9 - 56.547 * x**8
1.1 1.2
0.000001
```

Saída:

```
x_n             f(x_n)
1.150000        0.269865
1.129033        0.051096
1.122790        0.003755
1.122253        0.000026
1.122249        0.000000
Raiz aproximada = 1.122249
```

Entrada (B):

```
6.512 * x**13 - 7.512 * x**12 + 1
84.656 * x**12 - 90.144 * x**11
1.1 1.2
0.000001
```

Saída:

```
x_n             f(x_n)
1.150000        0.875874
1.123890        0.215205
1.111980        0.032188
1.109471        0.001239
1.109367        0.000002
1.109366        0.000000
Raiz aproximada = 1.109366
```

Dificuldades enfrentadas

A adaptação para receber derivadas. De começo a ideia seria colocar dentro do código, mas, percebi que o tratamento de erros seria muito complexo. Então, optei em ser uma das entradas.

Secante

Estratégia de Implementação:

Sem a necessidade de derivadas, o método começa com dois valores iniciais, e a cada iteração calcula um novo valor x_n utilizando a fórmula:

proximo_x = (f_atual * x_antigo - f_antigo * x_atual) / (f_atual - f_antigo)

Esse valor é calculado até que a função f(x_n) seja suficientemente próxima de zero, dentro da tolerância estabelecida.

A função eval() é utilizada para interpretar a função como string e avaliá-la a cada iteração.

O critério de parada utilizado é apenas o valor absoluto de f(x_n) porque o método da Secante, ao contrário de outros métodos, não precisa da atualização do intervalo, já que a função converge rapidamente em direção à raiz.

Estrutura dos Arquivos de Entrada/Saída

Entrada: secante-fun.txt

1. Função representada como string;
2. Dois valores iniciais de x (separados por espaço) como ponto de partida para o método;
3. Tolerância.

Saída: secante-res.txt

1. Cabeçalho:
    - x_n: Valor atual da iteração;
    - f(x_n): Valor da função nesse ponto.
2. Resultado final:
    - Raiz aproximada encontrada, com precisão definida pela tolerância.

Problema teste 3.3

Entrada (10%):

```
1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.1
1.1 1.2
0.000001
```

Saída:

```
x_n                 f(x_n)
1.100000            -0.000416
1.200000            0.020513
1.101989            -0.000015
1.102063            -0.000001
Raiz aproximada = 1.102063
```

Entrada (90%):

```
1 - (1 + x + (x**2 / 2)) * math.e**-x - 0.9
5.3 5.4
0.000001
```

Saída:

```
x_n                 f(x_n)
5.300000            -0.001554
5.400000            0.005242
5.322866            0.000038
5.322307            -0.000001
Raiz aproximada = 5.322307
```

Problema teste 3.6

Entrada:

```
(math.sin(x) * math.cos(x)) / (0.8 - math.cos(x)**2) - math.tan(math.radians(40))
1 1.05
0.000001
```

Saída:

```
x_n                 f(x_n)
1.000000            0.055749
1.050000            -0.057806
1.024547            -0.001784
1.023737            0.000058
1.023762            -0.000000
Raiz aproximada = 1.023762
```

Problema teste 3.8

Entrada (A):

```
5.283 * x**10 - 6.283 * x**9 + 1

1.1 1.2

0.000001
```

Saída:

```
x_n             f(x_n)
1.100000        -0.112244
1.200000        1.292044
1.107993        -0.080859
1.113412        -0.054094
1.124364        0.014995
1.121987        -0.001804
1.122242        -0.000050
1.122249        0.000000
Raiz aproximada = 1.122249
```

Entrada (B):

```
6.512 * x**13 - 7.512 * x**12 + 1

1.1 1.2

0.0001
```

Saída:

```
x_n             f(x_n)
1.100000        -0.094684
1.200000        3.696229
1.102498        -0.072432
1.104372        -0.054346
1.110003        0.007591
1.109312        -0.000636
1.109366        -0.000007
Raiz aproximada = 1.109366
```

Dificuldades enfrentadas

Não foi muito difícil tendo em vista que já havia uma boa base. Único detalhe a mais foi o uso da fórmula do método, mas nada complexo.

Eliminação de Gauss

Estratégia de Implementação:

A implementação começa com a leitura da matriz aumentada do sistema linear do arquivo. O código aplica a eliminação de Gauss para escalonar a matriz e torná-la triangular superior. Caso o pivô seja zero, a função trocar_linhas é utilizada para trocar as linhas da matriz e evitar problemas de divisão por zero. Após o escalonamento, a substituição inversa é realizada para resolver o sistema, calculando as variáveis de baixo para cima. Cada solução de xxx é calculada a partir da última equação e retrocedendo, usando as soluções já calculadas. O resultado é então armazenado em um arquivo de saída, incluindo a matriz escalonada e as soluções das variáveis. Essa abordagem é eficiente e lida adequadamente com o caso de pivôs nulos.

Estrutura dos Arquivos de Entrada/Saída

Entrada (eliminacaogauss-mat.txt):

Contém a matriz aumentada do sistema linear, onde cada linha representa uma equação do sistema e a última coluna contém os termos independentes. A estrutura da entrada foi escolhida por ser direta e compatível com o método da eliminação de Gauss.

Saída (eliminacaogauss-res.txt):

Contém a matriz escalonada após a eliminação de Gauss, seguida das soluções do sistema no formato xi=valor, uma por linha. Esse formato facilita a leitura e compreensão das soluções do sistema linear.

Problema teste 4.1

Entrada:

```
8 -4 -2 10

-4 6 -2 0

-2 -2 10 4
```

Saída:

```
8.0 -4.0 -2.0 22.0689655172413
8
0.0 4.0 -3.0 9.241379310344827

0.0 0.0 7.25 10.25

x0 = 2.7586206896551726

x1 = 2.310344827586207

x2 = 1.4137931034482758
```

Problema teste 4.3

Entrada:

```
-4 1 0 1 0 0 0 0 0 0

1 -4 1 0 1 0 0 0 0 0

0 1 -4 0 0 1 0 0 0 0

1 0 0 -4 1 0 1 0 0 0

0 1 0 1 -4 1 0 1 0 0

0 0 1 0 1 -4 0 0 1 0

0 0 0 1 0 0 -4 1 0 0

0 0 0 0 1 0 1 -4 1 0

0 0 0 0 0 1 0 1 -4 0
```

Saída:

```
-4.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0

0.0 -3.75 1.0 0.25 1.0 0.0 0.0 0.0 0.0 0.0

0.0 0.0 -3.7333333333333334 0.06666666666666667 0.26666666666666666 1.0 0.0 0.0 0.0 0.
0
0.0 0.0 0.0 -3.732142857142857 1.0714285714285714 0.017857142857142856 1.0 0.0 0.0 0.0

0.0 0.0 0.0 0.0 -3.4066985645933014 1.076555023923445 0.28708133971291866 1.0 0.0 0.0

0.0 0.0 0.0 0.0 0.0 -3.3918539325842696 0.09550561797752807 0.31601123595505615 1.0 0.
0
0.0 0.0 0.0 0.0 0.0 0.0 -3.705175983436853 1.093167701863354 0.028157349896480333 0.0

0.0 0.0 0.0 0.0 0.0 0.0 0.0 -3.3544926240500668 1.1014751899865893 0.0

0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -3.343283582089552 0.0

x0 = -0.0

x1 = -0.0

x2 = -0.0

x3 = -0.0

x4 = -0.0

x5 = -0.0

x6 = -0.0

x7 = -0.0

x8 = -0.0
```

Problema teste 4.6

Entrada:

```
14 4 4 100

4 7 19 100

4 7 18 100
```

Saída:

```
14.0 4.0 4.0 51.21951219512195

0.0 5.857142857142858 17.857142857142858 71.4285714285714
3
0.0 0.0 -1.0 0.0

x0 = 3.658536585365854

x1 = 12.195121951219512

x2 = -0.0
```

Dificuldades enfrentadas

Converter corretamente os enunciados dos exercícios para o formato exigido pelo programa exigiu bastante atenção. Além disso, houve desafios na parte lógica da implementação, principalmente no controle dos índices das matrizes durante os cálculos.

Fatoração LU

Estratégia de Implementação:

É realizada no código a partir da leitura da matriz aumentada do sistema, que é dividida em b (termos independentes) e U (coeficientes da matriz). O processo começa com o pivotamento parcial, que troca as linhas da matriz U para garantir que o maior valor absoluto na coluna seja o pivô, aumentando a estabilidade numérica.

Em seguida, o algoritmo de eliminação de Gauss é aplicado para transformar a matriz UU em uma matriz triangular superior, ao mesmo tempo em que são calculados os multiplicadores, que são armazenados em uma lista. Esses multiplicadores são usados para preencher a matriz triangular inferior L, onde a diagonal é preenchida com 1s e os valores abaixo da diagonal correspondem aos multiplicadores.

Após a decomposição, o sistema $L \cdot y = b$ é resolvido por substituição direta, e o sistema $U \cdot x = y$ é resolvido por substituição regressiva para encontrar a solução x. Essa abordagem garante a solução do sistema linear de forma eficiente e numericamente estável.

Estrutura dos Arquivos de Entrada/Saída

Entrada (fatoracaoLU-mat.txt):

Contém uma matriz aumentada representando o sistema linear. O formato da entrada é uma matriz de nnn linhas e n+1n+1n+1 colunas, onde a última coluna contém os termos independentes do sistema. Essa escolha foi feita para simplificar a leitura e a manipulação dos dados, pois a matriz aumentada já contém tudo o que é necessário para resolver o sistema.

Saída (fatoracaoLU-res.txt):

Contém as soluções das variáveis do sistema linear. Cada solução é salva em uma linha do arquivo no formato xi = valor. O formato de saída foi escolhido para ser simples e direto, permitindo fácil leitura e compreensão dos resultados.

Problema teste 4.1

Entrada:

```
 8 -4 -2 10

-4 6 -2 0

-2 -2 10 4
```

Saída:

```
x1 = 2.7586206896551726

x2 = 2.310344827586207

x3 = 1.4137931034482758
```

Problema teste 4.3

Entrada:

```
-4 1 0 1 0 0 0 0 0 0

1 -4 1 0 1 0 0 0 0 0

0 1 -4 0 0 1 0 0 0 0

1 0 0 -4 1 0 1 0 0 0

0 1 0 1 -4 1 0 1 0 0

0 0 1 0 1 -4 0 0 1 0

0 0 0 1 0 0 -4 1 0 0

0 0 0 0 1 0 1 -4 1 0

0 0 0 0 0 1 0 1 -4 0
```

Saída:

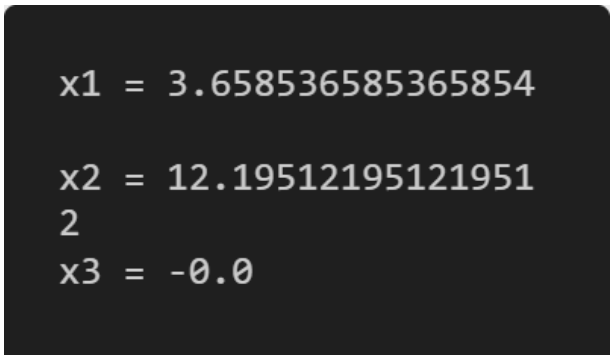
```
x1 = -0.0

x2 = -0.0

x3 = -0.0

x4 = -0.0

x5 = -0.0

x6 = -0.0

x7 = -0.0

x8 = -0.0

x9 = -0.0
```

Problema teste 4.6

Entrada:

```
14 4 4 100

4 7 19 100

4 7 18 100
```

Saída:



Dificuldades enfrentadas

A principal dificuldade foi manipular os arquivos e separar corretamente os coeficientes e termos independentes. Além disso, lidar com as matrizes L, U e o vetor b exigiu atenção à lógica, especialmente nas triangulações e estrutura dos dados.

Jacobi

Estratégia de Implementação:

Foi utilizada a biblioteca NumPy para facilitar as operações com matrizes e vetores. O código verifica automaticamente se a matriz é diagonalmente dominante, condição necessária para a convergência do método de Jacobi. Caso não seja, uma mensagem é salva no arquivo de saída e a execução é interrompida.

O critério de parada é baseado na norma do erro entre iterações consecutivas, com tolerância definida no arquivo de entrada. Isso garante precisão ajustável e controle sobre a convergência.

Estrutura dos Arquivos de Entrada/Saída

Entrada: jacobi-mat.txt

- Primeira linha: tolerância
- Linhas seguintes: coeficientes das equações com o termo independente ao final

Saída: jacobi-res.txt

- As soluções x_1, x_2, etc., uma por linha
- Número de iterações realizadas
- Ou uma mensagem de erro se a matriz não permitir convergência

Problema teste 5.1

Entrada:

```
0.000001

4 -1 0 -1 0 0 10
0
-1 4 -1 0 -1 0 0

0 -1 4 0 0 -1 0

-1 0 0 4 -1 0 10
0
0 -1 0 -1 4 -1 0

0 0 -1 0 -1 4 0
```

Saída:

```
x1 = 38.095237762007706

x2 = 14.285713814455352

x3 = 4.761904428674373

x4 = 38.095237762007706

x5 = 14.28571381445535

x6 = 4.761904428674374

iteracoes necessarias: 35
```

Problema teste 5.2

Entrada:

```
0.000001

20 -10 -4 26

-10 25 -5 0

-4 -5 20 7
```

Saída:

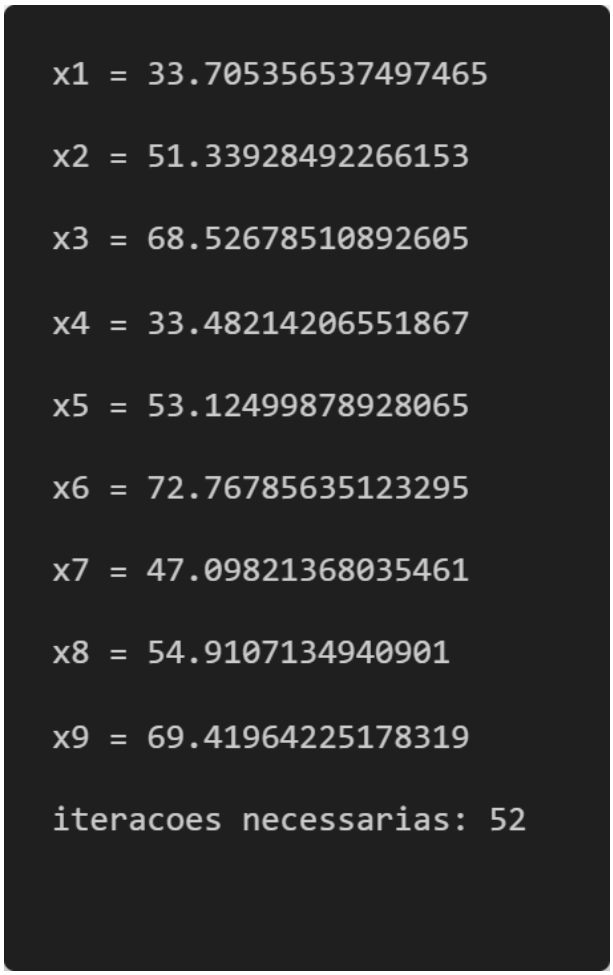
```
x1 = 1.9999991746285506

x2 = 0.9999992499757945

x3 = 0.9999994098523705

iteracoes necessarias: 28
```

Problema teste 5.5

Entrada:

```
0.000001

-4 1 0 1 0 0 0 0 0 -50

1 -4 1 0 1 0 0 0 0 -50

0 1 -4 0 0 1 0 0 0 -150

1 0 0 -4 1 0 1 0 0 0

0 1 0 1 -4 1 0 1 0 0

0 0 1 0 1 -4 0 0 1 -100

0 0 0 1 0 0 -4 1 0 -100

0 0 0 0 1 0 1 -4 1 -50

0 0 0 0 0 1 0 1 -4 -150
```

Saída:



Dificuldades enfrentadas

A maior dificuldade foi transformar a fórmula matemática em matriz e definir o critério de parada. Apesar da leitura estar resolvida, o resultado final divergente levantou dúvidas, possivelmente relacionadas ao tamanho ou à qualidade da entrada.

Gauss-Seidel

Estratégia de Implementação:

Foi utilizada a biblioteca NumPy para facilitar o trabalho com matrizes e vetores. O código verifica automaticamente se a matriz é diagonalmente dominante, condição necessária para a convergência do método de Gauss-Seidel. Caso essa condição não seja satisfeita, uma mensagem é gravada no arquivo de saída e a execução é encerrada. O critério de parada adotado é baseado na norma do erro entre duas iterações consecutivas, comparado com uma tolerância fornecida pelo usuário no próprio arquivo de entrada. Toda a saída é registrada em arquivo, mantendo o terminal limpo e favorecendo o uso automatizado ou a verificação posterior dos resultados.

Estrutura dos Arquivos de Entrada/Saída

Entrada: gaussseidel-mat.txt

- Primeira linha: tolerância do erro
- Linhas seguintes: coeficientes do sistema linear, com o termo independente ao final de cada linha

Saída: gaussseidel-res.txt

- As soluções x_1, x_2, etc., uma por linha
- Número de iterações realizadas
- Ou, uma mensagem de erro, caso a matriz não seja diagonalmente dominante e o método não possa ser aplicado

Problema teste 5.1

Entrada:

```
0.000001

4 -1 0 -1 0 0 100

-1 4 -1 0 -1 0 0

0 -1 4 0 0 -1 0

-1 0 0 4 -1 0 100

0 -1 0 -1 4 -1 0

0 0 -1 0 -1 4 0
```

Saída:

```
x1 = 38.095237762007706

x2 = 14.285714001284356

x3 = 4.761904640516697

x4 = 38.09523789411576

x5 = 14.285714114045637

x6 = 4.761904688640584

iteracoes necessarias: 18
```

Problema teste 5.2

Entrada:

```
0.000001

20 -10 -4 26

-10 25 -5 0

-4 -5 20 7
```

Saída:

```
x1 = 1.9999997875135869

x2 = 0.9999998745088652

x3 = 0.9999999261299336

iteracoes necessarias: 16
```

Problema teste 5.5

Entrada:

```
0.000001

-4 1 0 1 0 0 0 0 0 -50

1 -4 1 0 1 0 0 0 0 -50

0 1 -4 0 0 1 0 0 0 -150

1 0 0 -4 1 0 1 0 0 0

0 1 0 1 -4 1 0 1 0 0

0 0 1 0 1 -4 0 0 1 -100

0 0 0 1 0 0 -4 1 0 -100

0 0 0 0 1 0 1 -4 1 -50

0 0 0 0 0 1 0 1 -4 -150
```

Saída:

```
x1 = 33.70535671212045

x2 = 51.339285283549025

x3 = 68.52678549891738

x4 = 33.48214242640616

x5 = 53.12499956926331

x6 = 72.7678569274888

x7 = 47.09821407034594

x8 = 54.91071407034594

x9 = 69.41964274945869

iteracoes necessarias: 28
```

Dificuldades enfrentadas

Não houve muita dificuldade já que, são praticamente as mesmas dificuldades do Jacobi, só muda o quando será atualizado o vetor.

Gauss-Jordan

Estratégia de Implementação:

O método de Gauss-Jordan é um algoritmo direto usado para resolver sistemas lineares, baseado na eliminação de variáveis por meio da transformação da matriz aumentada em uma matriz identidade. Cada linha é manipulada para que os coeficientes da diagonal principal sejam 1 e os demais 0, permitindo obter as soluções diretamente. Utilizamos a biblioteca NumPy para facilitar essas operações. Diferente do método de Gauss-Seidel, que é iterativo e exige verificação de convergência, o Gauss-Jordan não depende de aproximações sucessivas nem de condições como a dominância diagonal, o que o torna mais direto, embora computacionalmente mais custoso para sistemas maiores.

Estrutura dos Arquivos de Entrada/Saída

Entrada (gaussjordan-mat.txt):

- Primeira linha: tolerância (não é utilizada neste método, mas incluída para manter compatibilidade com outros métodos)
- Linhas seguintes: coeficientes das equações, incluindo o termo independente ao final de cada linha

Saída (gaussjordan-res.txt):

- As soluções x_1, x_2, etc., uma por linha

Problema teste 5.1

Entrada:

```
0.000001
4 -1 0 -1 0 0 100
-1 4 -1 0 -1 0 0
0 -1 4 0 0 -1 0
-1 0 0 4 -1 0 100
0 -1 0 -1 4 -1 0
0 0 -1 0 -1 4 0
```

Saída:

```
x1 = 38.0952380952381

x2 = 14.285714285714286

x3 = 4.761904761904762

x4 = 38.095238095238095

x5 = 14.285714285714285

x6 = 4.761904761904762
```

Problema teste 5.2

Entrada:

```
0.000001

20 -10 -4 26

-10 25 -5 0

-4 -5 20 7
```

Saída:

```
x1 = 2.0

x2 = 1.0

x3 = 1.0
```

Problema teste 5.5

Entrada:

```
0.000001

-4 1 0 1 0 0 0 0 0 -50

1 -4 1 0 1 0 0 0 0 -50

0 1 -4 0 0 1 0 0 0 -150

1 0 0 -4 1 0 1 0 0 0

0 1 0 1 -4 1 0 1 0 0

0 0 1 0 1 -4 0 0 1 -100

0 0 0 1 0 0 -4 1 0 -100

0 0 0 0 1 0 1 -4 1 -50

0 0 0 0 0 1 0 1 -4 -150
```

Saída:

```
x1 = 33.70535714285714

x2 = 51.339285714285715

x3 = 68.52678571428571

x4 = 33.482142857142854

x5 = 53.125

x6 = 72.76785714285714

x7 = 47.09821428571429

x8 = 54.91071428571429

x9 = 69.41964285714286
```

Dificuldades enfrentadas

É mais complexa que Gauss-Seidel por exigir mais operações, uso de pivôs não nulos, manipulação precisa da matriz aumentada e ser sensível a erros numéricos. Para testar, ter a certeza de que as aproximações são válidas me preocupou.

Considerações Finais

O projeto também me proporcionou uma visão mais detalhada sobre como as iterações ocorrem durante a execução dos algoritmos. Foi interessante ver como a tolerância impacta diretamente o comportamento do algoritmo, seja afetando o número de iterações, o custo de desempenho ou até mesmo não fazendo muita diferença em determinados casos.

Quando ajustei a tolerância, percebi que em algumas situações, mesmo alterando o valor, o algoritmo ainda levava muitas iterações para alcançar um resultado satisfatório. Isso gerava uma discrepância no desempenho, especialmente em sistemas maiores, onde o tempo de execução aumentava. Por outro lado, em outros casos, a modificação da tolerância não trouxe uma diferença perceptível no resultado, mas teve um grande impacto no tempo de execução, tornando o processo mais rápido sem perder precisão.

Essas observações me ajudaram a entender que a tolerância não é um valor fixo, mas sim algo que depende das características do problema específico e da precisão que se deseja alcançar. A manipulação cuidadosa da tolerância, portanto, exige um equilíbrio entre precisão e eficiência, e foi um detalhe que, embora aparentemente simples, teve um grande efeito no desempenho geral do algoritmo.