



UNIVERSIDADE ESTADUAL DE SANTA CRUZ – UESC

ISRAEL SANTANA DOS ANJOS

**RELATÓRIO DE IMPLEMENTAÇÃO DE MÉTODOS PARA A DISCIPLINA DE
ANÁLISE NÚMERICA**

ILHÉUS – BAHIA

2024

ISRAEL SANTANA DOS ANJOS

RELATÓRIO DE IMPLEMENTAÇÃO DE MÉTODOS PARA A DISCIPLINA DE ANÁLISE
NÚMERICA

Relatório da análise de implementação dos métodos apresentado ao Curso de graduação em Ciência da computação da Universidade Estadual de Santa Cruz, como parte de um dos créditos da disciplina de análise numérica.

Docente: Gesil Sampaio Amarantes

ILHÉUS – BAHIA

2024

Sumário

RELATÓRIO DE IMPLEMENTAÇÃO DE MÉTODOS PARA A DISCIPLINA DE ANÁLISE NÚMERICA	1
Linguagem escolhida e justificativa	5
Teste e funcionamento do código.....	5
Método da Bissecção.....	6
Estratégias da implementação	6
Estrutura dos arquivos de entrada/saída	7
Problema teste 3.3:	7
Análise.....	8
Problema teste 3.6:	9
Análise.....	9
Problema teste 3.8:	10
Plano A.....	10
Plano B.....	10
Análise:.....	11
Dificuldades enfrentadas	11
Método do ponto fixo	11
Estratégia de implementação.....	11
Estrutura dos arquivos de entrada/saída	12
Problema 3.3:	12
Análise.....	13
Problema 3.6:	13
Análise.....	13
Problema 3.8:	14
Plano A.....	14
Plano B.....	14
Análise.....	15
Dificuldades enfrentadas	15
Secante	15
Estratégia de implementação.....	15
Estrutura dos arquivos de entrada/saída	15
Problema 3.3:	15
Problema 3.6:	16
Problema 3.6:	17
Plano A.....	17

Plano B.....	17
Análise.....	18
Dificuldades enfrentadas	18
Jacobi.....	18
Estratégia de implementação.....	18
Estrutura dos arquivos de entrada/saída	18
Problema 1:.....	19
Problema 2:.....	19
Análise:.....	20
Dificuldades enfrentadas	20
Eliminação de Gauss e Fatoração LU.....	20
Estratégia de implementação.....	20
Estrutura dos arquivos de entrada/saída	20
Problema 3:.....	21
Análise dos métodos de eliminação de Gauss e fatoração LU	21
Dificuldades enfrentadas	21

Linguagem escolhida e justificativa

Para um melhor desenvolvimento foi escolhido o python como linguagem de programação, pois, essa linguagem é bastante simples implementar métodos numéricos, como também, possui uma vasta biblioteca e comunidade grande. A versão foi a 3.11.7, o interpretador pode ser baixado através do link: www.python.org.

A instalação das bibliotecas utilizadas pode ser feita pelo prompt de comando, ou terminal da IDE, pelo seguinte comando:

- pip install sympy
- pip install numpy

Uma vez instalado não se faz necessário chamar os comandos novamente.

Como ambiente de desenvolvimento foi escolhido o *visual studio code*, por facilitar a escrita e depuração de códigos, sendo também, uma opção de fácil identificação de erros de sintaxe e formatação concisa. A versão utilizada foi a 1.87.2 e o download poder ser feito pelo link: code.visualstudio.com.

Outra ferramenta utilizada na realização dos cálculos e plotar os gráficos das funções foi o *geoGebra*. Essa ferramenta, foi de grande auxílio na escolha dos intervalos para a resolução dos problemas. O *geoGebra* pode ser acessado pelo link: www.geogebra.org.

Teste e funcionamento do código

Para executar o código de cada método que vai ser apresentado, é necessário certificar-se que o interpretador python e as bibliotecas está devidamente instalado na máquina.

Abra a pasta e clique no arquivo com a terminação “.py” do exemplo escolhido, e ao abrir com o *visual studio code*, clique no menu superior em “executar” e depois em “iniciar depuração” (ou pressione F5). Por conseguinte, o arquivo “resultadoNomedométodo.txt” será atualizado, esses passos valem para todos os métodos implementados, exceto para o método da bisseccção.

Para o método da bisseccção em particular, é preciso abrir a pasta *input* e alterar o arquivo de entrada com os intervalos, erro tolerado e função. Após, basta clicar na pasta métodos e abrir também com o *visual studio code*, e então, iniciar a depuração. Um arquivo será gerado na pasta *output* com a saída.

Na pasta onde estão os métodos há um arquivo “entradasDeTeste.txt”, nele, contém algumas entradas teste para os problemas desenvolvidos nesse relatório.

Método da Bissecção

Estratégias da implementação

A iteração dos dados foi feita através de arquivos externos (.txt). Foi necessário o uso da biblioteca `sympy` para a conversão das strings (x e função) em variáveis simbólicas e função.

Após a leitura do arquivo e atribuições necessárias, é feita a verificação inicial (figura 1), o código apura se o valor da função tem o mesmo sinal.

```
if f.subs(x, a) * f.subs(x, b) >= 0:
    print("A função deve ter sinais opostos em (a) e (b).")
    SystemExit()
```

Figura 1

Se tiverem, uma mensagem de erro é exibida e o programa é encerrado '`SystemExit()`'. Isso ocorre porque o método da bissecção depende da mudança de sinal dentro do intervalo.

O loop `while` é executado enquanto a diferença entre ' a ' e ' b ' dividida por 2.0 for maior que a tolerância especificada (figura 2).

```
while (b - a) / 2.0 > erro:
    c = (a + b) / 2.0
    cont+=1
    if f.subs(x, c) <= erro:
        break
    elif f.subs(x, c) * f.subs(x, a) < 0:
        b = c
    else:
        a = c
```

Figura 2

Dentro do loop ' c ' é calculado como ponto médio do intervalo. A variável '`cont`' é um contador para quantificar o número de iterações.

Três condições são verificadas, mostrado na figura 2:

- **Caso base:** Se ' $f.subs(x, c)$ ' for menor que a tolerância.
- **Mudança de sinal:** Se ' $f.subs(x, c)$ ' e ' $f.subs(x, a)$ ' tem sinais opostos, indica que a raiz está no subintervalo $[a, c]$, então b é atualizado para diminuir a busca.
- **Mesmo sinal:** Se ' $f.subs(x, c)$ ' tem o mesmo sinal que ' $f.subs(x, a)$ ' então, a raiz está no subintervalo $[c, b]$. Portanto, a é atualizado.

Após o loop terminar o resultado é calculado como a média de 'a' e 'b', aproximando a raiz da função 'f' (raiz estimada) dentro da tolerância especificada (figura 3).

```
resultado = (a + b) / 2.0  
fx = f.subs(x, resultado)
```

Figura 3

Estrutura dos arquivos de entrada/saída

O arquivo de entrada é organizado de maneira que em cada linha esteja armazenado um dos dados necessários para o cálculo do método. Neste sentido, a primeira linha contém o valor do intervalo inicial 'a', em seguida, na segunda linha, o valor do final 'b', na terceira, o erro tolerado e por último a função (figura 4).

Essa estrutura foi pensada no intuito de criar uma padronização, de forma que, o algoritmo não encontre dificuldades para encontrar as informações necessárias.

```
a  
b  
erro  
função|
```

Figura 4

A saída esta estruturada da seguinte forma; a primeira linha contém o título com o número do problema, na segunda linha o intervalo, e na terceira e quarta linha, respectivamente, a função aplicando a raiz e o número de iterações (figura 5).

```
===== PROBLEMA 0.0 =====  
A raiz encontrada é: X  
Valor da função aplicada a raiz é: Y  
Numero de iterações necessária: Z
```

Figura 5

Problema teste 3.3:

Para $g = 0.1$ e intervalo inicial e final, mostrados na figura 6, foi encontrada a seguinte raiz (figura 8):

```

Bisseccao > ≡ entrada.txt
1  5.000
2  6.000
3  0.001
4  0.1 - (1 + x + (x**2)/2)*exp(-x)

```

Figura 6 - entrada

```

Bisseccao > ≡ resultado.txt
1  ===== PROBLEMA 3.3 =====
2  Intervalo: [5.000, 6.000]
3  A raiz encontrada é: 5.25
4  Valor da função aplicada à raiz é: -0.00511435293360210
5  Número de iterações necessário: 2

```

Figura 7 – saída

Para $g = 0.9$ e dados de entrada e saída mostrados respectivamente na figura 8 e 9, foi encontrada a seguinte raiz:

```

Bisseccao > ≡ entrada.txt
1  1.000
2  2.000
3  0.001
4  0.9 - (1 + x + (x**2)/2)*exp(-x)

```

Figura 8 - entrada

```

Bisseccao > ≡ resultado.txt
1  ===== PROBLEMA 3.3 =====
2  Intervalo: [1.000, 2.000]
3  A raiz encontrada é: 1.0625
4  Valor da função aplicada à raiz é: -0.00785070745318506
5  Número de iterações necessário: 4

```

Figura 9 – saída

Análise

O método da bissecção parece ser eficiente para encontrar as raízes das funções fornecidas, com um número relativamente baixo de iterações necessárias para alcançar uma boa precisão.

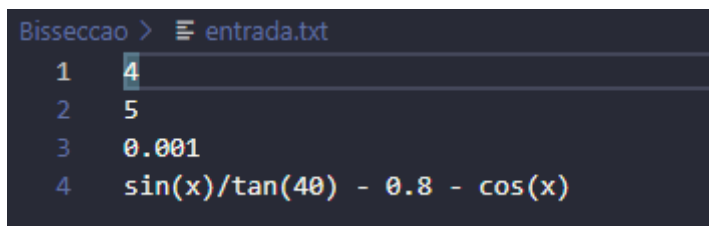
Na primeira iteração, a raiz foi encontrada com uma precisão de 0.001 dentro do intervalo [5.000, 6.000]. O valor da função na raiz é muito próximo de zero (-0.005114), indicando que a raiz foi encontrada com boa precisão. Na segunda iteração, a raiz foi encontrada com uma

precisão de 0.001 dentro do intervalo [1.000, 2.000]. O valor da função na raiz é novamente próximo de zero (-0.007850), indicando que a raiz foi encontrada com boa precisão.

Comparando as duas iterações, podemos observar que a primeira iteração precisou de menos iterações (2) do que a segunda iteração (4) para encontrar a raiz com a mesma precisão. Isso pode ser devido à forma da função na primeira iteração, que apresenta uma declividade mais acentuada no intervalo inicial. O valor da função na raiz na primeira iteração (-0.005114) é menor em magnitude do que o valor da função na raiz na segunda iteração (-0.007850). Isso pode indicar que a primeira raiz está mais próxima de um zero verdadeiro da função do que a segunda raiz.

Problema teste 3.6:

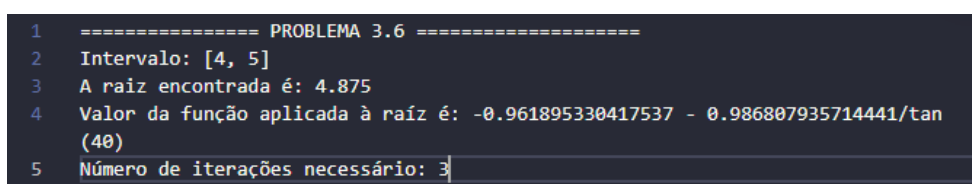
Ao plotar o gráfico para a função $\left(\frac{\sin(x)}{\tan(40)}\right) - 0.8 - \cos(x)$ verificou-se que não é possível encontrar raiz para o intervalo inicialmente proposto, então, foi definido um novo intervalo para que seja possível o teste. Nesse sentido, ficou-se então definido os seguintes intervalos (figura 10):



```
Bisseccao > ≡ entrada.txt
1 4
2 5
3 0.001
4 sin(x)/tan(40) - 0.8 - cos(x)
```

Figura 10 – entrada

Esses foram os resultados:



```
Bisseccao > ≡
1 ===== PROBLEMA 3.6 =====
2 Intervalo: [4, 5]
3 A raiz encontrada é: 4.875
4 Valor da função aplicada à raiz é: -0.961895330417537 - 0.986807935714441/tan(40)
5 Número de iterações necessário: 3
```

Figura 11 – saída

Análise

A função em questão, $(\sin(x)/\tan(40)) - 0.8 - \cos(x)$, apresenta uma combinação de funções trigonométricas e seno e cosseno, além de uma constante. Essa combinação pode levar a um comportamento complexo da função, dificultando a análise precisa da raiz.

O número relativamente baixo de iterações necessárias (3) sugere que a raiz está em uma região onde a função apresenta uma declividade acentuada, o que facilita a convergência do método.

Problema teste 3.8:

Plano A

Neste problema ao plotar novamente o gráfico, assim como no problema anterior, foi verificado que existe uma raiz entre os pontos 1.01 e 1.2.

```
Bisseccao > entrada.txt
1 1.01
2 1.2
3 0.001
4 (5.2830*(x**10)) - (6.2830*(x**9)) + 1
```

Figura 12 - entrada

```
Bisseccao > resultado.txt
1 ===== PROBLEMA 3.8 =====
2 Intervalo: [1.01, 1.2]
3 A raiz encontrada é: 1.105
4 Valor da função aplicada à raiz é: -0.0937008957302652
5 Número de iterações necessário: 1
```

Figura 13 – saída

Plano B

Nesta função o problema se repete quanto a raiz, foi verificado ao projetar o gráfico no *geogebra* que existe uma raiz entre 1.01 e 1.2, o resultado gerado pelo algoritmo está demonstrado na figura 15.

```
Bisseccao > entrada.txt
1 1.01
2 1.2
3 0.001
4 (6.5116*(x**13)) - (7.5116*(x**12)) + 1
```

Figura 14 - entrada

```
Bisseccao > resultado.txt
1  ===== PROBLEMA 3.8 =====
2  Intervalo: [1.01, 1.2]
3  A raiz encontrada é: 1.105
4  Valor da função aplicada à raiz é: -0.0481460756501555
5  Número de iterações necessário: 1
```

Figura 15 – saída

Análise:

Fazendo uma análise comparativa entre as duas funções e suas respectivas saídas, podemos inferir que:

Apesar de as duas funções apresentarem potências elevadas de x (x^{10} e x^{13}), o método da bissecção convergiu rapidamente para as raízes em ambas as iterações. Isso pode ser devido à forma das funções, que apresentam um comportamento relativamente simples no intervalo $[1.01, 1.2]$.

O valor da função na raiz na primeira iteração (-0.093701) é maior em magnitude do que o valor da função na raiz na segunda iteração (-0.048146). Isso sugere que a primeira raiz está mais distante de um zero verdadeiro da função do que a segunda raiz.

Dificuldades enfrentadas

Não houve maiores dificuldades, com a utilização da ferramenta geogebra, foi bastante simples encontrar a raiz que desejamos.

Método do ponto fixo

Estratégia de implementação

Similar ao método da bissecção, a iteração com os dados neste método foi realizada utilizando arquivos externos, com exceção da função em si. A linguagem Python facilita a manipulação de arquivos, tornando a implementação do método bastante eficiente. No entanto, a biblioteca "math" foi utilizada para realizar o cálculo de algumas funções.

O método do ponto fixo foi desenvolvido em um arquivo .py com o seu respectivo nome. A função, a abertura do arquivo de entrada (que contém o intervalo e o erro tolerado) e a escrita no arquivo de saída (que contém a solução) foram implementadas em outro arquivo.

Estrutura dos arquivos de entrada/saída

Para padronizar os resultados, os formatos dos arquivos de entrada e saída permaneceram os mesmos, com exceção da última linha no de entrada e a terceira linha no de saída. Deste modo, os arquivos de entrada contêm três linhas sendo elas; a primeira contendo o valor de 'a', a segunda com o valor de 'b' e a terceira com o valor da tolerância. O arquivo de saída, têm quatro linhas, contém respectivamente; nome do problema, intervalo, raiz encontrada e o número de iteração.

Problema 3.3:

Para este problema se fez necessário rearranjar a função pois, com a função dada no problema o algoritmo não estava conseguindo calcular em tempo hábil e apresentava um erro de overflow. Após fazer esse ajuste e organizar os arquivos de entrada temos o seguinte arquivo de entrada para o problema como $g = 0.1$ e $f(x) = 0.1 - (1 + x + (x ** 2)/2) * \exp(x)$:

```
Input > entradaPonto.txt
1      5.000
2      6.000
3      0.001
```

Figura 16 – entrada

Depois de executar o código temos a seguinte saída:

```
Output > resultadoPonto.txt
1      ===== PROBLEMA 3.3 =====
2      Intervalo: [6.000, 5.000]
3      A raiz encontrada é: -0.3760650921439541
4      Número de iterações necessário: 67
```

Figura 17 – saída

Para o problema como $g = 0.9$ e $f(x) = 0.9 - (1 + x + (x ** 2)/2) * \exp(x)$ temos a seguinte entrada e saída respectivamente:

```
Input > entradaPonto.txt
1      1.000
2      2.000
3      0.001
```

Figura 18 - entrada

```

Output > resultadoPonto.txt
1  ===== PROBLEMA 3.3 - G = 0.9 =====
2  Intervalo: [2.000, 1.000]
3  A raiz encontrada é: 0.8026085547715119
4  Número de iterações necessário: 99

```

Figura 19 – saída

Análise

Para $g = 0.1$ o método necessitou de um número elevado de iterações (67) para convergir para a raiz. Isso pode ser devido à escolha da função iterativa ou do intervalo inicial. (figura 17)

Para $g = 0.9$ o método necessitou de um número elevado de iterações (99) para convergir para a raiz. Isso pode ser devido à escolha da função iterativa ou do intervalo inicial. (figura 19)

Problema 3.6:

Para a $f(x) = (\sin(x)/\tan(40)) - 0.8 - \cos(x)$ e as seguintes entradas, foram obtidos os resultados a seguir (figura 21).

```

Input > entradaPonto.txt
1  2.5
2  3
3  0.001

```

Figura 20 - entrada

```

Output > resultadoPonto.txt
1  ===== PROBLEMA 3.6 =====
2  Intervalo: [3, 2.5]
3  A raiz encontrada é: -2.0994056805825014
4  Número de iterações necessário: 99

```

Figura 21 – saída

Análise

O método não convergiu para a raiz dentro do limite de 100 iterações. Isso pode ser devido à escolha da função iterativa, que pode não ter um ponto fixo no intervalo inicial ou que pode levar a uma divergência do método.

Problema 3.8:

Plano A

Ao plotar o gráfico para a $f(x) = (5.2830 * (x ** 10)) - (6.2830 * (x ** 9)) + 1$, assim como, descrito no método da bissecção, foi observado que existe uma raiz entre $x < 1.01$ e $x < 1.2$. Logo, foram obtidos os seguintes resultados:

```
Input > entradaPonto.txt
1 1.01
2 1.2
3 0.001
```

Figura 22 - entrada

```
Output > resultadoPonto.txt
1 ===== PROBLEMA 3.8 - Plano A =====
2 Intervalo: [1.2, 1.01]
3 A raiz encontrada é: 1.0
4 Número de iterações necessário: 99
```

Figura 23 – saída

Plano B

Plotando o gráfico no geogebra para a seguinte função: $f(x) = (6.5116 * (x ** 13)) - (7.5116 * (x ** 12)) + 1$ foi obtida os seguintes resultados (figura 24) como as seguintes entradas (figura 23):

```
Input > entradaPonto.txt
1 1.01
2 1.2
3 0.001
```

Figura 24 - entrada

```
Output > resultadoPonto.txt
1 ===== PROBLEMA 3.8 - Plano B =====
2 Intervalo: [1.2, 1.01]
3 A raiz encontrada é: 1.0
4 Número de iterações necessário: 99
```

Figura 25 – saída

Análise

Ao plotar os gráficos no geogebra pode -se perceber que existem duas raízes entre os pontos, ao executar o algoritmo para essas funções pôde ser encontrada a raiz mais próxima de zero, que era o que queríamos.

Dificuldades enfrentadas

Inicialmente foi difícil desenvolver um algoritmo que mostrasse um resultado aceitável. Com exceção das duas últimas funções todas apresentaram bons resultados.

Secante

Estratégia de implementação

Similar ao método da bissecção, a iteração com os dados neste método foi realizada utilizando arquivos externos, com exceção da função em si. A linguagem Python facilita a manipulação de arquivos, tornando a implementação do método bastante eficiente. No entanto, a biblioteca "math" foi utilizada para realizar o cálculo de algumas funções.

A secante foi desenvolvida em um arquivo .py. A função, a abertura do arquivo de entrada (que contém o intervalo e o erro tolerado) e a escrita no arquivo de saída (que contém a solução) também foram implementadas neste arquivo.

Estrutura dos arquivos de entrada/saída

Para padronizar os resultados, os formatos dos arquivos de entrada e saída permaneceram os mesmos, com exceção da última linha no de entrada e a terceira linha no de saída. Deste modo, os arquivos de entrada contêm três linhas sendo elas; a primeira contendo o valor de 'a', a segunda com o valor de 'b' e a terceira com o valor da tolerância. O arquivo de saída, têm quatro linhas, contém respectivamente; nome do problema, intervalo, raiz encontrada e o número de iteração.

Problema 3.3:

Para a resolução desta questão, procedeu-se a uma reestruturação da função, de modo a viabilizar a obtenção de um resultado em tempo real. A função proposta pelo problema anterior não pôde ser representada adequadamente, uma vez que a linguagem Python apresenta limitações na representação de strings de grande extensão. A na imagem (figura 26) contém a entrada que foi utilizada:

```

Input > entrada.txt
1      5.000
2      6.000
3      0.001

```

Figura 26 – entrada

A seguir temos a saída:

```

Output > resultadoSecante1.txt
1  ===== PROBLEMA 3.3 =====
2  Intervalo: [6.000, 5.000]
3  A raiz encontrada é: -3.7528715942444286
4  Número de iterações necessário: 24

```

Figura 27 – saída

A mesma reestruturação supracitada, foi necessária para $g = 0.9$. A seguir temos a saída:

```

Input > entrada1.txt
1      1.000
2      2.000
3      0.001

```

Figura 28 - entrada

```

Output > resultadoSecante2.txt
1  ===== PROBLEMA 3.3 - G = 0.9 =====
2  Intervalo: [1.000, 2.000]
3  A raiz encontrada é: -0.05225944632404851
4  Número de iterações necessário: 7

```

Figura 29 – saída

O método convergiu rapidamente para a raiz, demonstrando boa eficiência neste caso.

Problema 3.6:

Reescrevendo a função e organizado nosso conjunto de dados, temos:

```

Input > entrada2.txt
1      2.5
2      3
3      0.001

```

Abaixo temos a saída:


```

Output > resultadoSecante3.txt
1  ===== PROBLEMA 3.6 =====
2  Intervalo: [2.5, 3]
3  A raiz encontrada é: 2.939338839014308
4  Número de iterações necessário: 2

```

Problema 3.6:

Plano A

No problema $f(x) = (5.2830 * (x ** 10)) - (6.2830 * (x ** 9)) + 1$ ao plotar o gráfico usando o método da secante, assim como no problema anterior, foi verificado que existe uma raiz entre os pontos 1.01 e 1.2.

Essa foi a entrada:

```

Input > entrada3.txt
1  1.01
2  1.2
3  0.001

```

E a saída:

```

Output > resultadoSecante4.txt
1  ===== PROBLEMA 3.8 - Plano A =====
2  Intervalo: [1.01, 1.2]
3  A raiz encontrada é: 1.0001558679330382
4  Número de iterações necessário: 4

```

Plano B

Do mesmo modo que no plano A, plotou – se o gráfico da $f(x) = (6.5116 * (x ** 13)) - (7.5116 * (x ** 12)) + 1$ e esses foram os seguintes resultados.

A seguir as entradas que foram utilizadas:

```

Input > entrada4.txt
1  1
2  1.2
3  0.001

```

Na imagem seguinte a saída obtida a partir da execução do algoritmo:

```
Output > resultadoSecante5.txt
1  ===== PROBLEMA 3.8 - Plano B =====
2  Intervalo: [1, 1.2]
3  A raiz encontrada é: 1.0
4  Número de iterações necessário: 1
```

Análise

O método da secante se mostrou eficiente para encontrar as raízes das cinco funções fornecidas, com um número relativamente baixo de iterações necessárias para alcançar a precisão desejada de 0.001. Em todas as entradas, o método convergiu para uma raiz dentro do intervalo inicial especificado. O número de iterações necessárias variou de 1 a 24, dependendo da função e do intervalo inicial.

Para os problemas 3.3 $g = 0.1$, 3.3 $g = 0.9$, 3.8 plano A e 3.8 plano B, o método da secante se mostrou uma boa escolha para encontrar as raízes das funções.

Dificuldades enfrentadas

Não houve maiores dificuldades

Jacobi

Estratégia de implementação

A iteração com os dados neste método foi realizada utilizando arquivos externos. A linguagem Python facilita a manipulação de arquivos, tornando a implementação do método bastante eficiente.

O método de Jacobi foi desenvolvido em um arquivo .py. Enquanto a matriz e o vetor em outro arquivo.

Estrutura dos arquivos de entrada/saída

Para padronizar os resultados, os formatos dos arquivos de entrada e saída permaneceram os mesmos para os sistemas lineares. Deste modo, os arquivos de entrada contêm cinco linhas sendo elas; a primeira, segunda e terceira linha os elementos da matriz. Nas linhas subsequentes contêm os caracteres "-" para separar a última linha do arquivo, que contém o vetor b.

O arquivo de saída, têm sete linhas, contendo respectivamente; a primeira linha com o título do problema, a segunda até a quarta linha matriz A, apresentada em formato matricial com seus elementos organizados em linhas e colunas, a quinta o vetor “b” listado sequencialmente e as últimas contendo os valores obtidos após a resolução do sistema e o número de iterações.

Problema 1:

Para o problema da figura 30, foi definida um épsilon de 0.05 e foi apresentada a seguinte saída (figura 31):

```
Input > entrada6.txt
1  10 2 1
2  1 5 1
3  2 3 10
4  -----
5  7 -8 6
```

Figura 30 – entrada

```
Output > resultadoJacobi.txt
1  ===== PROBLEMA 1 =====
2  Matriz A:
3  [10.0] [0.2] [0.1]
4  [0.2] [5.0] [0.2]
5  [0.2] [0.3] [10.0]
6  Vetor b: [0.7, -1.6, 0.6]
7  Vetor resultante: [0.9994, -1.9888000000000001, 0.9984]
```

Figura 31 – saída

Problema 2:

Para o problema a seguir obtivemos a seguinte saída (figura 32):

```
1  5 1 1
2  3 4 1
3  3 3 6
4  -----
5  5 6 0
```

```
Output > resultadoJacobi.txt
1  ===== PROBLEMA 2 =====
2  Matriz A:
3  [5.0] [0.2] [0.2]
4  [0.75] [4.0] [0.25]
5  [0.5] [0.5] [6.0]
6  Vetor b: [1.0, 1.5, 0.0]
7  Vetor resultante: [1.0097306640625, 1.0165526123046875, -0.9817258300781251]
```

Figura 32 - saída

Análise:

O método de Jacobi se mostrou eficiente para resolver os dois sistemas de equações lineares fornecidos, convergindo para a solução em apenas duas iterações em ambos os casos.

Em ambas as entradas, o método de Jacobi apresentou uma taxa de convergência rápida, demonstrando ser um método iterativo eficiente para resolver sistemas de equações lineares com coeficientes diagonais dominantes.

Dificuldades enfrentadas

Não houve maiores dificuldades

Eliminação de Gauss e Fatoração LU

Estratégia de implementação

A iteração com os dados nos métodos descritos foi realizada utilizando arquivos externos, aproveitando a facilidade que a linguagem Python oferece para manipulação de arquivos. Essa abordagem resultou em uma implementação extremamente eficiente, otimizando o tempo de processamento e a legibilidade do código.

A organização modular foi adotada, com os métodos implementados em um arquivo *.py e a matriz e o vetor armazenados em um arquivo *.txt separado. Essa estrutura facilita a manutenção e reutilização do código, além de promover maior clareza e organização.

Para executar os testes, basta clicar no "exemplo7.py" e executar o código. Todas as entradas e pastas de saída já foram previamente definidas, garantindo um processo de teste simples e automatizado.

Estrutura dos arquivos de entrada/saída

Para padronizar os resultados, os formatos dos arquivos de entrada e saída permaneceram os mesmos para os sistemas lineares. Deste modo, os arquivos de entrada contêm cinco linhas sendo elas; a primeira, segunda e terceira linha os elementos da matriz. Nas linhas subsequentes contêm os caracteres "-" para separar a última linha do arquivo, que contém o vetor b.

O arquivo de saída, têm sete linhas, contendo respectivamente; a primeira linha com o título do problema, a segunda até a quarta linha matriz A, apresentada em formato matricial com seus elementos organizados em linhas e colunas, a quinta o vetor "b" listado sequencialmente e as últimas contendo os valores obtidos após a resolução do sistema e o número de iterações.

Problema 3:

Os testes para os métodos de Eliminação de Gauss e Fatoração LU foram executados no mesmo ambiente, utilizando o mesmo sistema. As saídas obtidas estão detalhadas nas Figuras 33 e 34, respectivamente.

```
Input > entrada7.txt
1  5 1 1
2  3 4 1
3  3 3 6
4  -----
5  5 6 0
```

```
Output > resultadoFatoLU.txt
1  ===== PROBLEMA 3 =====
2  Matriz A:
3  [5] [1] [1]
4  [0] [3.4] [0.4]
5  [0] [0] [5.11764705882353]
6  Vetor b: [5, 3.0, -5.117647058823529]
7  Vetor resultante: [1.0, 1.0, -0.999999999999998]
8  Número de interações: 1
```

Figura 33- saída, LU

```
Output > resultadoElimGaus.txt
1  ===== PROBLEMA 3 =====
2  Matriz A:
3  [5] [1] [1]
4  [0] [3.4] [0.4]
5  [0] [0] [5.11764705882353]
6  Vetor b: [5, 3.0, -5.117647058823529]
7  Vetor resultante: [1.0, 1.0, -0.999999999999998]
8  Número de interações: 1
```

Figura 34- Saída, Gauss

Análise dos métodos de eliminação de Gauss e fatoração LU

Tanto o método de eliminação de Gauss quanto a fatoração LU se mostraram eficientes para resolver o sistema de equações lineares fornecido, necessitando de apenas uma iteração em ambos os casos.

Os resultados obtidos com os dois métodos são praticamente idênticos, demonstrando que ambos são métodos matematicamente equivalentes para resolver sistemas de equações lineares triangulares superiores.

Dificuldades enfrentadas

Não houve maiores dificuldades

