



Universidade Estadual de Santa Cruz - UESC

Terceiro Relatório 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

Índice.....	2
Lista de Figuras.....	4
Método de Euler.....	4
Método de Euler Modificado.....	4
Método de Heun.....	4
Método de Ralston.....	4
Método de Runge-Kutta (4 ordem).....	4
Método das Diferenças Finitas.....	5
Sistemas EDO.....	5
Linguagem escolhida e justificativas.....	6
Instruções gerais.....	7
Método 1 (Euler).....	8
Estratégia de Implementação.....	8
Estrutura dos Arquivos de Entrada/Saída.....	8
Problema teste 1.....	9
Problema teste 2.....	10
Dificuldades enfrentadas.....	11
Método 2 (Euler Modificado).....	12
Estratégia de Implementação.....	12
Estrutura dos Arquivos de Entrada/Saída.....	12
Problema teste 1.....	13
Problema teste 2.....	14
Dificuldades enfrentadas.....	15
Método 3 (Heun).....	16
Estratégia de Implementação.....	16
Estrutura dos Arquivos de Entrada/Saída.....	16
Problema teste 1.....	17
Problema teste 2.....	18
Dificuldades enfrentadas.....	19
Método 4 (Ralston).....	21
Estratégia de Implementação.....	21
Estrutura dos Arquivos de Entrada/Saída.....	21
Problema teste 1.....	21
Problema teste 2.....	22
Dificuldades enfrentadas.....	23

Método 5 (Runge-Kutta de 4a ordem).....	24
Estratégia de Implementação.....	24
Estrutura dos Arquivos de Entrada/Saída.....	24
Problema teste 1.....	24
Problema teste 2.....	25
Dificuldades enfrentadas.....	26
Método 6 (Diferenças Finitas).....	27
Estratégia de Implementação.....	27
Estrutura dos Arquivos de Entrada/Saída.....	27
Problema teste.....	28
Dificuldades enfrentadas.....	29
Método 7 (Sistema Edo).....	30
Estratégia de Implementação.....	30
Estrutura dos Arquivos de Entrada/Saída.....	30
Problema teste.....	31
Dificuldades enfrentadas.....	31
Conclusão.....	33

Lista de Figuras

Método de Euler

- Imagem 1.1 - Arquivo de entrada do exemplo 1
- Imagem 1.2 - Arquivo de saída do exemplo 1
- Imagem 1.3 - Arquivo de entrada do exemplo 2
- Imagem 1.4 - Arquivo de saída do exemplo 2

Método de Euler Modificado

- Imagem 2.1 - Arquivo de entrada do exemplo 1
- Imagem 2.2 - Arquivo de saída do exemplo 1
- Imagem 2.3 - Arquivo de entrada do exemplo 2
- Imagem 2.4 - Arquivo de saída do exemplo 2

Método de Heun

- Imagem 3.1 - Arquivo de entrada do exemplo 1
- Imagem 3.2 - Arquivo de saída do exemplo 1
- Imagem 3.3 - Arquivo de entrada do exemplo 2
- Imagem 3.4 - Arquivo de saída do exemplo 2

Método de Ralston

- Imagem 4.1 - Arquivo de entrada do exemplo 1
- Imagem 4.2 - Arquivo de saída do exemplo 1
- Imagem 4.3 - Arquivo de entrada do exemplo 2
- Imagem 4.4 - Arquivo de saída do exemplo 2

Método de Runge-Kutta (4 ordem)

- Imagem 5.1 - Arquivo de entrada do exemplo 1
- Imagem 5.2 - Arquivo de saída do exemplo 1
- Imagem 5.3 - Arquivo de entrada do exemplo 2
- Imagem 5.4 - Arquivo de saída do exemplo 2

Método das Diferenças Finitas

Imagem 6.1 - Arquivo de entrada do exemplo 3

Imagem 6.2 - Arquivo de saída do exemplo 3

Sistemas EDO

Imagem 7.1 - Arquivo de entrada do exemplo 3

Imagem 7.2 - Arquivo de saída do exemplo 3

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.

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 (Euler)

Estratégia de Implementação

Para iniciar o procedimento, criou-se a função "euler_method", a qual recebe os parâmetros cruciais para a implementação do Método de Euler. Esses parâmetros englobam a função diferencial "f", que representa a equação diferencial a ser resolvida, os valores iniciais de "x0" e "y0", o incremento do passo "h" e o número desejado de iterações "iterations". A função "euler_method" tem como objetivo calcular os pontos que compõem a solução.

Dentro da função, uma lista chamada "points" é gerada para armazenar as coordenadas dos pontos da solução. As variáveis "x" e "y" são inicializadas com os valores iniciais fornecidos. Posteriormente, um loop é implementado para realizar as iterações conforme especificado por "n".

Durante cada iteração, a inclinação no ponto atual é calculada utilizando a função diferencial "f" e os valores correntes de "x" e "y". O valor de "y" é então atualizado empregando o Método de Euler, multiplicando a inclinação pelo incremento do passo "h" e adicionando o resultado ao valor corrente de "y". Adicionalmente, o valor de "x" é incrementado em um passo "h" para progredir para o próximo ponto. A cada iteração, o ponto atual é agregado à lista "pontos" como uma tupla contendo os valores de "x" e "y".

Ao finalizar o loop, a função retorna a lista "pontos" contendo todos os valores da solução. Além disso, foi incorporada a leitura dos dados de entrada a partir do arquivo "euler.txt" que se encontra na pasta "entradas" e a escrita da saída no arquivo "euler.txt" armazenado na pasta "saídas".

Estrutura dos Arquivos de Entrada/Saída

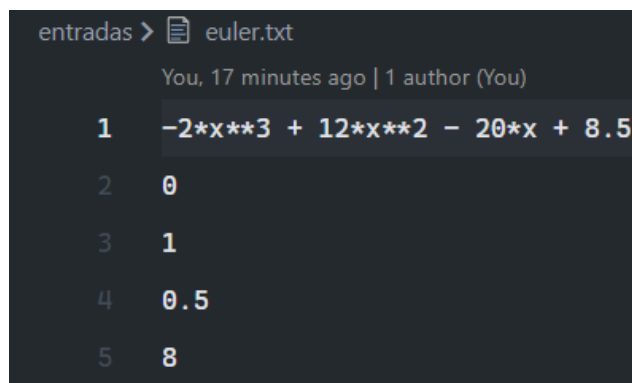
O arquivo de entrada, intitulado "euler.txt", segue um formato específico projetado para simplificar a leitura dos dados. A primeira linha contém a expressão da função diferencial em formato de string. Nas linhas subsequentes, são registrados, em sequência, o valor inicial de "x", o valor inicial de "y", o incremento do passo "h" e o número desejado de iterações "n".

No arquivo de saída, cada linha representa um ponto da solução e segue um formato determinado. Cada linha é registrada no arquivo com a estrutura: "x = valor de x, y = valor de y". Os valores de "x" e "y" são apresentados com três casas decimais.

A adoção dessa estrutura nos arquivos de entrada e saída tem o propósito de simplificar a extração dos dados iniciais de um arquivo e a gravação dos resultados em um arquivo separado. Isso proporciona uma abordagem organizada e eficiente para a análise dos resultados obtidos por meio do Método de Euler.

Problema teste 1

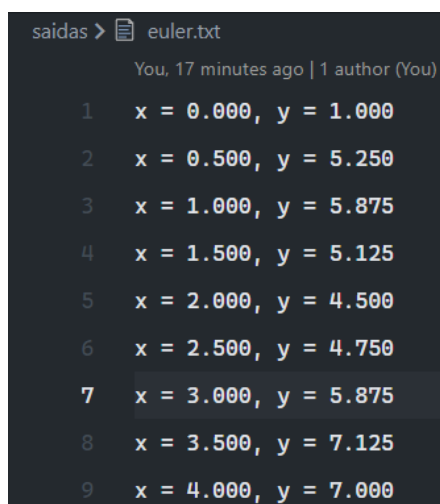
- **Entrada**



```
entradas > euler.txt
You, 17 minutes ago | 1 author (You)
1  -2*x**3 + 12*x**2 - 20*x + 8.5
2  0
3  1
4  0.5
5  8
```

Imagem 1.1

- **Saída**

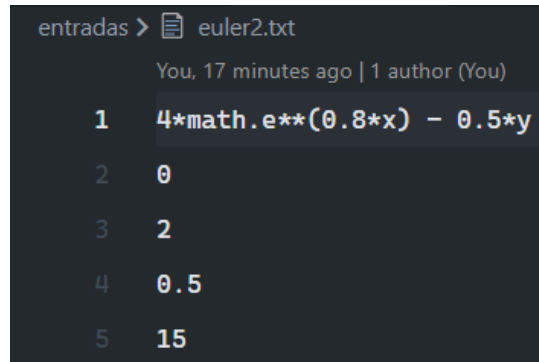


```
saidas > euler.txt
You, 17 minutes ago | 1 author (You)
1  x = 0.000, y = 1.000
2  x = 0.500, y = 5.250
3  x = 1.000, y = 5.875
4  x = 1.500, y = 5.125
5  x = 2.000, y = 4.500
6  x = 2.500, y = 4.750
7  x = 3.000, y = 5.875
8  x = 3.500, y = 7.125
9  x = 4.000, y = 7.000
```

Imagem 1.2

Problema teste 2

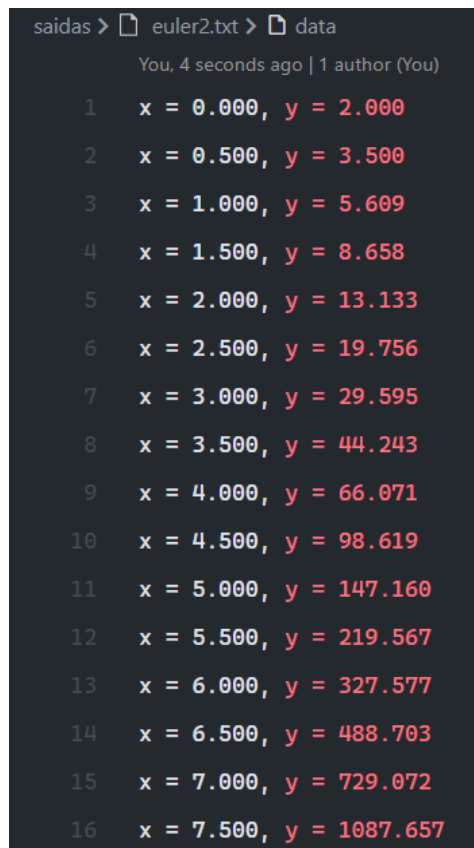
- Entrada



```
entradas > euler2.txt
You, 17 minutes ago | 1 author (You)
1 4*math.e**(0.8*x) - 0.5*y
2 0
3 2
4 0.5
5 15
```

Imagem 1.3

- Saída



```
saidas > euler2.txt > data
You, 4 seconds ago | 1 author (You)
1 x = 0.000, y = 2.000
2 x = 0.500, y = 3.500
3 x = 1.000, y = 5.609
4 x = 1.500, y = 8.658
5 x = 2.000, y = 13.133
6 x = 2.500, y = 19.756
7 x = 3.000, y = 29.595
8 x = 3.500, y = 44.243
9 x = 4.000, y = 66.071
10 x = 4.500, y = 98.619
11 x = 5.000, y = 147.160
12 x = 5.500, y = 219.567
13 x = 6.000, y = 327.577
14 x = 6.500, y = 488.703
15 x = 7.000, y = 729.072
16 x = 7.500, y = 1087.657
```

Imagem 1.4

Dificuldades enfrentadas

Não houveram dificuldades.

Método 2 (Euler Modificado)

Estratégia de Implementação

Inicialmente, foi implementada uma função denominada "modified_euler_method" para realizar os cálculos do método em questão. Esta função recebe vários parâmetros, incluindo a função diferencial, os valores iniciais, o tamanho do passo e o número de iterações, e retorna uma lista de pontos que representa a solução aproximada. Adicionalmente, foi definida uma segunda função, nomeada $f(x, y)$, para expressar a equação diferencial, sendo essa função fornecida por meio de um arquivo externo denominado "euler_modificado.txt".

O algoritmo opera iterativamente por um número específico de iterações (iterations). Em cada iteração, calculamos as taxas de variação em dois pontos distintos. Dentro do loop de iterações, iniciamos o cálculo da taxa de variação no ponto atual (slope1) usando a função $f(x, y)$. Em seguida, realizamos o cálculo utilizando um valor temporário para y (y_temp), somando o produto do tamanho do passo (h) pela taxa de variação calculada.

Após obter o valor temporário de y , calculamos a taxa de variação no próximo ponto (slope2) utilizando o valor temporário de y e o próximo valor de x ($x + h$). Por fim, atualizamos os valores de x e y , considerando a média ponderada das duas taxas de variação e o tamanho do passo.

A cada iteração, o ponto calculado (x, y) é adicionado à lista de pontos. Ao término das iterações, a lista contém todos os valores da solução aproximada. Esta abordagem foi projetada para oferecer uma descrição detalhada do método utilizado, proporcionando uma compreensão clara do processo de resolução numérica da equação diferencial.

Estrutura dos Arquivos de Entrada/Saída

O documento "euler_modificado.txt" desempenha o papel de entrada para o programa, incorporando informações cruciais para sua execução. Sua organização segue o seguinte padrão: a primeira linha especifica a equação diferencial a ser resolvida; a segunda linha contém o valor inicial de x (x_0); a terceira linha apresenta o valor inicial de

y (y0); a quarta linha detalha o tamanho do passo (h); e a quinta linha indica o número de iterações (n).

Por outro lado, o arquivo de saída "saida_euler_modificado.txt" é destinado a armazenar os resultados da solução aproximada. A estrutura do arquivo é simples: as linhas exibem os pontos da solução. Cada linha contém as coordenadas x e y de um ponto, proporcionando uma representação clara e organizada dos resultados obtidos por meio do método aplicado.

Problema teste 1

- **Entrada**

```
entradas > euler_modificado.txt
You, 19 minutes ago | 1 author (You)
1  -2*x**3 + 12*x**2 - 20*x + 8.5
2  0
3  1
4  0.5
5  8
```

Imagem 2.1

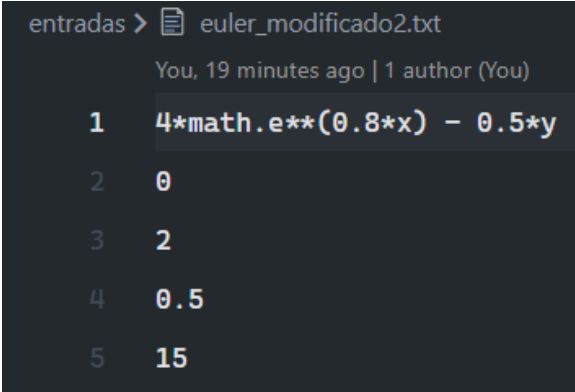
- **Saída**

```
saidas > euler_modificado.txt
You, 19 minutes ago | 1 author (You)
1  x = 0.000, y = 1.000
2  x = 0.500, y = 3.438
3  x = 1.000, y = 3.375
4  x = 1.500, y = 2.688
5  x = 2.000, y = 2.500
6  x = 2.500, y = 3.188
7  x = 3.000, y = 4.375
8  x = 3.500, y = 4.938
9  x = 4.000, y = 3.000
```

Imagem 2.2

Problema teste 2

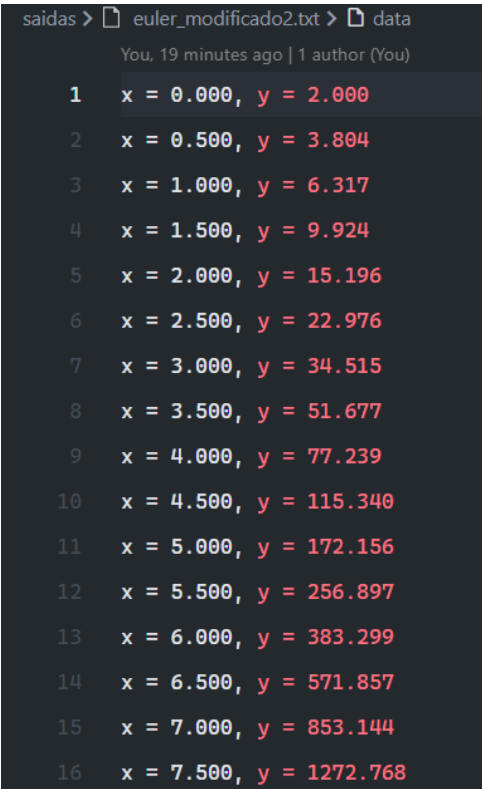
- Entrada



```
entradas > euler_modificado2.txt
You, 19 minutes ago | 1 author (You)
1 4*math.e**(0.8*x) - 0.5*y
2 0
3 2
4 0.5
5 15
```

Imagem 2.3

- Saída



```
saidas > euler_modificado2.txt > data
You, 19 minutes ago | 1 author (You)
1 x = 0.000, y = 2.000
2 x = 0.500, y = 3.804
3 x = 1.000, y = 6.317
4 x = 1.500, y = 9.924
5 x = 2.000, y = 15.196
6 x = 2.500, y = 22.976
7 x = 3.000, y = 34.515
8 x = 3.500, y = 51.677
9 x = 4.000, y = 77.239
10 x = 4.500, y = 115.340
11 x = 5.000, y = 172.156
12 x = 5.500, y = 256.897
13 x = 6.000, y = 383.299
14 x = 6.500, y = 571.857
15 x = 7.000, y = 853.144
16 x = 7.500, y = 1272.768
```

Imagem 2.4

Dificuldades enfrentadas

Não houveram dificuldades.

Método 3 (Heun)

Estratégia de Implementação

A abordagem de implementação selecionada envolve a criação de uma função chamada "heun_method". Nesta função, a equação diferencial, os valores iniciais (x_0 e y_0), o tamanho do passo (h) e o número desejado de iterações são parâmetros. Dentro desse contexto, um loop é executado para calcular a solução aproximada da equação utilizando o método de Heun.

Durante cada iteração, os valores de "x" e "y" são iterativamente atualizados, e a função " $f(x, y)$ " é avaliada para determinar as inclinações necessárias. Os pares de valores "x" e "y" são armazenados em uma lista de soluções a cada passo. Ao final do loop, a função retorna a lista contendo as soluções.

Na próxima etapa do código, os parâmetros da equação são lidos de um arquivo de texto. A função a ser utilizada é então definida, e a função "heun" é invocada com os parâmetros apropriados. Os resultados obtidos são registrados em um arquivo de saída.

Resumidamente, o código implementa de forma eficiente o método de Heun para resolver equações diferenciais ordinárias. Ele extrai os parâmetros necessários de um arquivo de texto, utiliza a função específica fornecida e armazena os resultados em um arquivo distinto. Essa abordagem oferece uma solução versátil e modular para resolver problemas diferenciais, proporcionando flexibilidade na escolha e manipulação das equações.

Estrutura dos Arquivos de Entrada/Saída

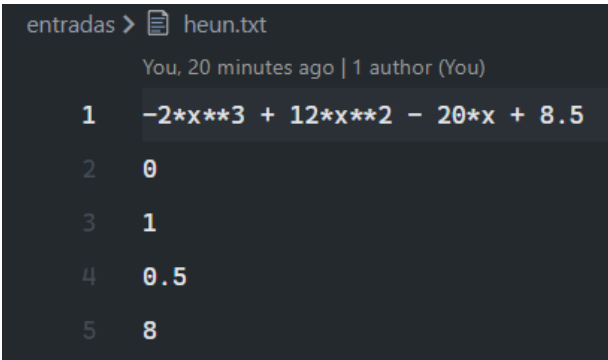
Os documentos de entrada, identificados como "heun.txt" e "heun2.txt", adotam uma estrutura específica. Na primeira linha, é necessário inserir a equação diferencial a ser resolvida, representada como uma string. Na segunda linha, deve-se fornecer o valor inicial de " x_0 " como um número de ponto flutuante, seguido, na terceira linha, pelo valor inicial de " y_0 " também como um número de ponto flutuante. A quarta linha contém o tamanho do passo, representado por " h ", como um número de ponto flutuante, e a quinta linha indica o número de passos desejados, representado por "interacoes", como um número inteiro.

O arquivo de saída, intitulado "saida_heun.txt", é utilizado para armazenar os resultados do método de Heun aplicado à equação diferencial. A estrutura do arquivo é simples, composta por linhas de texto no formato "x = {x:.2f}, y = {y:.4f}". As formatações "{x:.2f}" e "{y:.4f}" garantem que os valores de "x" e "y" sejam exibidos com duas e quatro casas decimais, respectivamente.

Cada linha no arquivo representa um ponto da solução encontrada pelo método de Heun, onde "x" é a coordenada independente e "y" é a correspondente coordenada dependente calculada para esse ponto. A separação de cada ponto por uma quebra de linha facilita a leitura e interpretação dos resultados, proporcionando uma visualização clara das soluções obtidas.

Problema teste 1

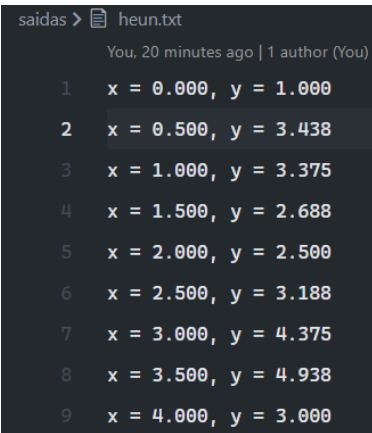
- **Entrada**



```
entradas > heun.txt
You, 20 minutes ago | 1 author (You)
1 -2*x**3 + 12*x**2 - 20*x + 8.5
2 0
3 1
4 0.5
5 8
```

Imagem 3.1

- **Saída**

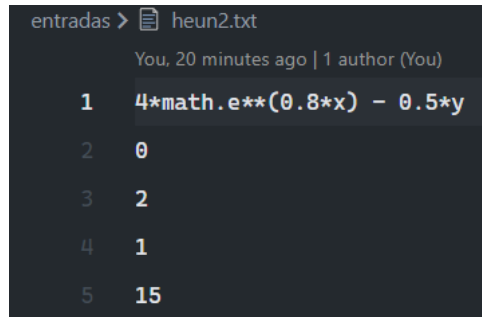


```
saidas > heun.txt
You, 20 minutes ago | 1 author (You)
1 x = 0.000, y = 1.000
2 x = 0.500, y = 3.438
3 x = 1.000, y = 3.375
4 x = 1.500, y = 2.688
5 x = 2.000, y = 2.500
6 x = 2.500, y = 3.188
7 x = 3.000, y = 4.375
8 x = 3.500, y = 4.938
9 x = 4.000, y = 3.000
```

Imagem 3.2

Problema teste 2

- Entrada

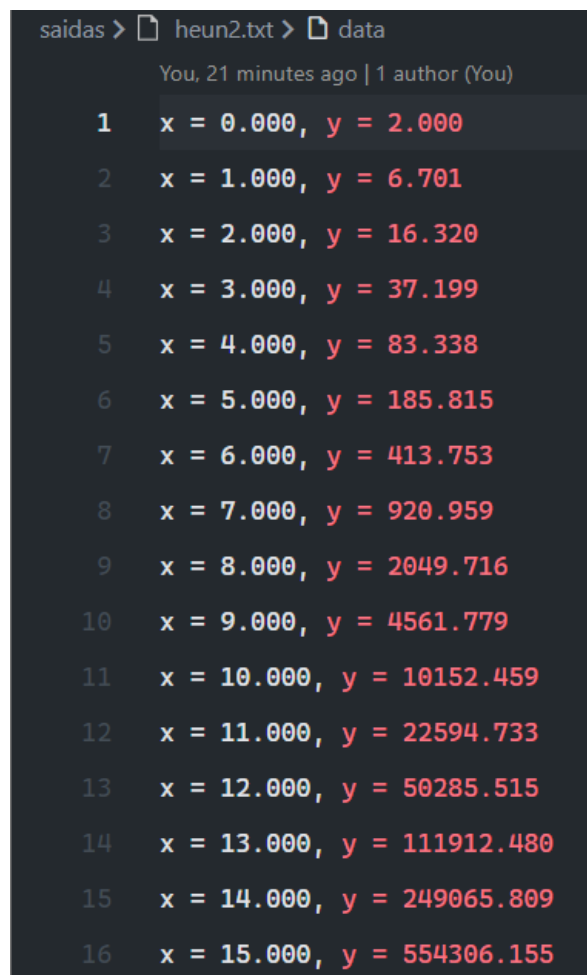


A screenshot of a code editor window titled 'entradas > heun2.txt'. The window shows a list of five input values, each preceded by a line number from 1 to 5. The values are: 1. $4 * \text{math.e}^{(0.8 * x)} - 0.5 * y$, 2. 0, 3. 2, 4. 1, and 5. 15. The text is displayed in a dark-themed editor with light-colored syntax highlighting.

```
1 4*math.e**(0.8*x) - 0.5*y
2 0
3 2
4 1
5 15
```

Imagem 3.3

- Saída



A screenshot of a code editor window titled 'saídas > heun2.txt > data'. The window shows a list of 16 output lines, each preceded by a line number from 1 to 16. Each line displays the values of x and y at a specific step, formatted as 'x = [value], y = [value]'. The values of x range from 0.000 to 15.000 in increments of 1.000, and the values of y increase exponentially. The text is displayed in a dark-themed editor with light-colored syntax highlighting.

```
1 x = 0.000, y = 2.000
2 x = 1.000, y = 6.701
3 x = 2.000, y = 16.320
4 x = 3.000, y = 37.199
5 x = 4.000, y = 83.338
6 x = 5.000, y = 185.815
7 x = 6.000, y = 413.753
8 x = 7.000, y = 920.959
9 x = 8.000, y = 2049.716
10 x = 9.000, y = 4561.779
11 x = 10.000, y = 10152.459
12 x = 11.000, y = 22594.733
13 x = 12.000, y = 50285.515
14 x = 13.000, y = 111912.480
15 x = 14.000, y = 249065.809
16 x = 15.000, y = 554306.155
```

Imagem 3.4

Dificuldades enfrentadas

Durante a implementação do método, a principal complexidade estava relacionada à incorporação das interações passadas ao arquivo. Inicialmente, os resultados diferiam significativamente das expectativas, demandando esforço para compreender onde as interações deveriam ser integradas no loop. A disparidade inicial revelou-se uma questão de compreensão sobre como as interações passadas deveriam ser adequadamente integradas ao ciclo de execução.

O desafio central consistiu em ajustar a implementação para garantir a incorporação correta das interações passadas ao processo iterativo, evidenciando a importância de uma abordagem precisa na execução do algoritmo. Essa experiência destaca a necessidade de uma compreensão profunda do processo para superar desafios iniciais e assegurar a precisão desejada na implementação do método.

Método 4 (Ralston)

Estratégia de Implementação

A abordagem de implementação neste código consiste na criação da função "ralston_method", a qual emprega o método de Ralston para resolver equações diferenciais ordinárias (EDOs) de primeira ordem. Essa função recebe como parâmetros a função "f", que descreve a derivada da função desconhecida, o valor inicial (x0, y0), o tamanho do passo "h" e o número de iterações "iterations".

A função "ralston" inicializa uma lista de pontos com o ponto inicial (x0, y0) e, em seguida, utiliza um loop para conduzir as iterações do método de Ralston. No interior do loop, calcula o valor do declive "slope1" no ponto atual (x, y) utilizando a função "f". Posteriormente, calcula o declive "slope2" no ponto avançado. A etapa subsequente consiste em atualizar os valores de x e y por meio da fórmula do método de Ralston: $y = y + (1/4) * h * (slope1 + 3 * slope2)$ e $x = x + h$. O ponto atualizado (x, y) é então adicionado à lista de pontos. Ao concluir o loop, a função retorna a lista de pontos, que contém os pontos da solução aproximada.

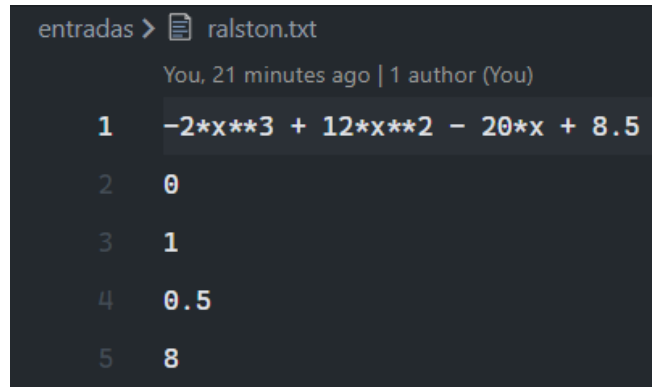
Estrutura dos Arquivos de Entrada/Saída

A configuração dos arquivos de entrada e saída neste código adota uma estrutura particular, buscando simplificar o processamento das informações. Cada tipo de arquivo está na sua respectiva pasta (entrada e saída). O arquivo de entrada, nomeado "ralston.txt", tem cinco linhas. A primeira linha contém a expressão da derivada, descrevendo a taxa de variação da função desconhecida em relação às variáveis independentes. As linhas subsequentes fornecem o valor inicial de x (x0), o valor inicial de y (y0), o tamanho do passo (h) e o número desejado de iterações (n) para a solução.

Já o arquivo de saída, denominado "ralston.txt", também segue um formato específico. Cada ponto da solução aproximada é registrado, exibindo os valores correspondentes de x e y. Esses valores são arredondados para duas casas decimais, proporcionando uma apresentação clara e facilitando a análise dos resultados.

Problema teste 1

- Entrada

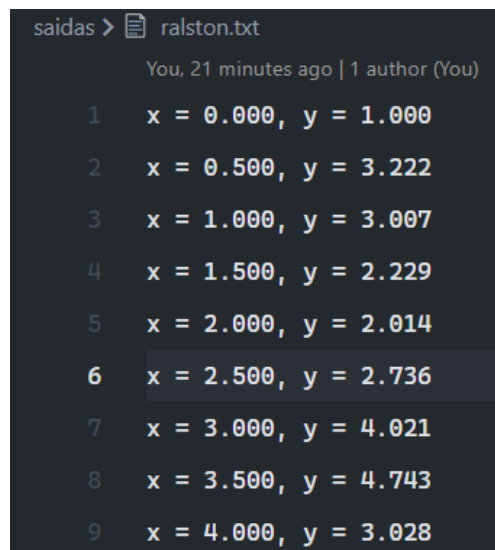


A screenshot of a code editor window titled 'entradas > ralston.txt'. The editor shows a polynomial equation and five input values. The first line is the equation $-2x^3 + 12x^2 - 20x + 8.5$. The following lines are the input values: 0, 1, 0.5, and 8. The line numbers 1 through 5 are visible on the left side of the editor.

```
entradas > ralston.txt
You, 21 minutes ago | 1 author (You)
1  -2*x**3 + 12*x**2 - 20*x + 8.5
2  0
3  1
4  0.5
5  8
```

Imagem 4.1

- Saída



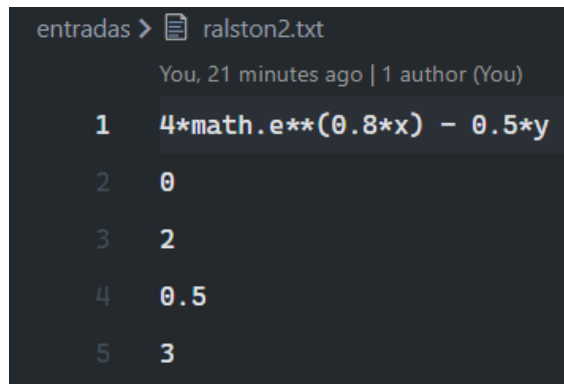
A screenshot of a code editor window titled 'saidas > ralston.txt'. The editor shows nine lines of output, each displaying the value of x and y. The output lines are: x = 0.000, y = 1.000; x = 0.500, y = 3.222; x = 1.000, y = 3.007; x = 1.500, y = 2.229; x = 2.000, y = 2.014; x = 2.500, y = 2.736; x = 3.000, y = 4.021; x = 3.500, y = 4.743; and x = 4.000, y = 3.028. The line numbers 1 through 9 are visible on the left side of the editor.

```
saidas > ralston.txt
You, 21 minutes ago | 1 author (You)
1  x = 0.000, y = 1.000
2  x = 0.500, y = 3.222
3  x = 1.000, y = 3.007
4  x = 1.500, y = 2.229
5  x = 2.000, y = 2.014
6  x = 2.500, y = 2.736
7  x = 3.000, y = 4.021
8  x = 3.500, y = 4.743
9  x = 4.000, y = 3.028
```

Imagem 4.2

Problema teste 2

- Entrada

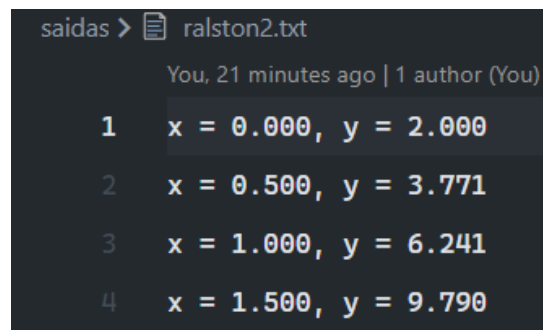


A screenshot of a code editor window titled 'entradas > ralston2.txt'. It shows a list of five input lines, each preceded by a line number in a light gray font. The first line contains a mathematical expression, and the following four lines contain numerical values.

```
1 4*math.e**(0.8*x) - 0.5*y
2 0
3 2
4 0.5
5 3
```

Imagem 4.3

- Saída



A screenshot of a code editor window titled 'saidas > ralston2.txt'. It shows a list of four output lines, each preceded by a line number in a light gray font. Each line displays the values of variables x and y at a specific step.

```
1 x = 0.000, y = 2.000
2 x = 0.500, y = 3.771
3 x = 1.000, y = 6.241
4 x = 1.500, y = 9.790
```

Imagem 4.4

Dificuldades enfrentadas

Não houveram dificuldades.

Método 5 (Runge-Kutta de 4ª ordem)

Estratégia de Implementação

A função chamada "runge_kutta_method" inicia uma lista chamada "points" com o ponto inicial (x_0 , y_0) e, em seguida, utiliza um loop para realizar n iterações, aplicando o método de Runge-Kutta de 4ª ordem. Dentro desse loop, o método é executado em quatro etapas distintas. Primeiramente, o valor de k_1 é calculado multiplicando o tamanho do passo h pela função f no ponto (x, y) . Em seguida, os valores de k_2 , k_3 e k_4 são computados usando fórmulas adequadas, considerando avanços proporcionais ao tamanho do passo h e combinações dos valores de k_1 , k_2 e k_3 .

Na etapa seguinte, os valores de x e y são atualizados por meio da fórmula característica do método de Runge-Kutta de 4ª ordem. O ponto atualizado (x, y) é então adicionado à lista "pontos". Após a conclusão do loop, a função retorna a lista "pontos", que agora contém os valores da solução aproximada. Este procedimento oferece uma abordagem sistemática e eficaz para a aplicação do método de Runge-Kutta de 4ª ordem, resultando em uma lista abrangente de pontos que representam a solução aproximada para o problema em questão.

Estrutura dos Arquivos de Entrada/Saída

Dentro do escopo deste trabalho, o arquivo de entrada, intitulado "runge_kutta4.txt", adota uma organização simples. Nele, as informações seguem uma disposição sequencial: a primeira linha contém a equação, seguida pela especificação do valor inicial de x (x_0), do valor inicial de y (y_0), do tamanho do passo (h) – indicando o intervalo entre os pontos consecutivos da solução – e do número de iterações (n). Este último determina quantas vezes o método de Runge-Kutta de 4ª ordem será aplicado para avançar na solução.

No que diz respeito ao arquivo de saída, cada ponto da solução aproximada é registrado, apresentando os valores de x e y com duas casas decimais de precisão. Cada ponto é registrado em uma linha separada, aprimorando a legibilidade e simplificando a análise dos resultados.

Problema teste 1

- Entrada

```
entradas > runge_kutta4.txt
You, 22 minutes ago | 1 author (You)
1 -2*x**3 + 12*x**2 - 20*x + 8.5
2 0
3 1
4 0.5
5 8
```

Imagem 5.1

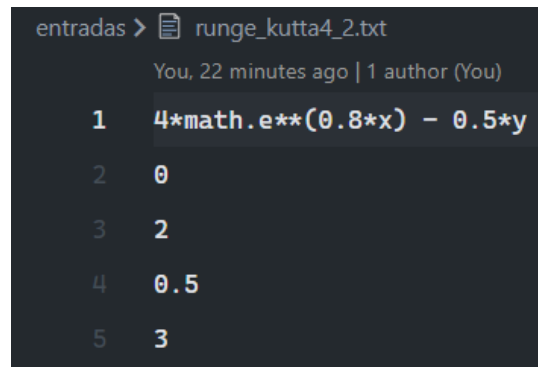
- Saída

```
saidas > runge_kutta4.txt
You, 22 minutes ago | 1 author (You)
1 x = 0.000, y = 1.000
2 x = 0.500, y = 3.219
3 x = 1.000, y = 3.000
4 x = 1.500, y = 2.219
5 x = 2.000, y = 2.000
6 x = 2.500, y = 2.719
7 x = 3.000, y = 4.000
8 x = 3.500, y = 4.719
9 x = 4.000, y = 3.000
```

Imagem 5.2

Problema teste 2

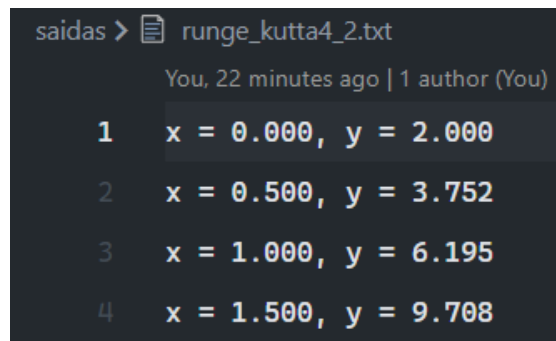
- **Entrada**



```
entradas > runge_kutta4_2.txt
You, 22 minutes ago | 1 author (You)
1 4*math.e**(0.8*x) - 0.5*y
2 0
3 2
4 0.5
5 3
```

Imagem 5.3

- **Saída**



```
saidas > runge_kutta4_2.txt
You, 22 minutes ago | 1 author (You)
1 x = 0.000, y = 2.000
2 x = 0.500, y = 3.752
3 x = 1.000, y = 6.195
4 x = 1.500, y = 9.708
```

Imagem 5.4

Dificuldades enfrentadas

Desafios significativos apareceram ao tentar compreender a fórmula associada ao método de quarta ordem. Para superar essa dificuldade, foi crucial buscar informações em fontes adicionais, o que contribuiu de maneira crucial para a compreensão e aplicação bem-sucedidas do método. A necessidade de recorrer a recursos externos destaca a importância de explorar diferentes materiais para aprofundar o entendimento, evidenciando a abordagem proativa na resolução de desafios específicos na compreensão de conceitos complexos.

Método 6 (Diferenças Finitas)

Estratégia de Implementação

O código começa lendo dados de um arquivo de texto chamado "diferencas_finitas.txt" por meio da função 'loadtext' da biblioteca NumPy. Em seguida, a função "verify_matrix" desempenha o papel de garantir que a matriz A e o vetor b tenham as dimensões adequadas para resolver o sistema. Isso inclui validar que a matriz A seja quadrada (com número de linhas igual ao número de colunas) e que o tamanho do vetor B coincida com o número de linhas da matriz A. Caso alguma dessas condições não seja atendida, a função emite um ValueError.

A função "solve_linear_system" utiliza o método das diferenças finitas para resolver o sistema tridiagonal. O processo começa inicializando o vetor x de soluções com valores zero, onde $x[0]$ é definido como $b[0] / A[0, 0]$. A função itera sobre as linhas da matriz A, realizando atualizações nos valores da matriz e do vetor b para eliminar coeficientes abaixo da diagonal principal, conhecido como eliminação de Gauss. Em seguida, ocorre uma nova iteração sobre as linhas de A, mas dessa vez de baixo para cima, para resolver o sistema triangular superior, substituindo os valores de x. O resultado final é retornado como o vetor x de solução.

Por fim, o vetor solução x é armazenado em um arquivo de texto chamado "diferenca_finita.txt" usando a função np.savetxt. Esse arquivo serve como registro do resultado do processo de resolução do sistema tridiagonal.

Estrutura dos Arquivos de Entrada/Saída

Neste código específico, a organização dos dados de entrada e saída utiliza matrizes e vetores, isso contribui para o aproveitando a eficiência do Python na manipulação de arquivos de texto, especialmente com a biblioteca NumPy. Essa biblioteca oferece funcionalidades úteis, como a função numpy.savetxt(), que simplifica a escrita e leitura direta de matrizes e vetores em arquivos de texto.

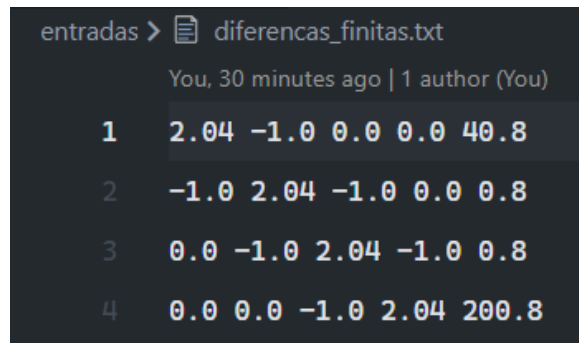
Os arquivos de entrada seguem a estrutura do arquivo "diferencas_finitas.txt". Quando os dados são unidimensionais, a matriz A é composta por todos os elementos, excluindo o último, enquanto o vetor b é definido como o último elemento do arquivo. No caso de

dados bidimensionais, a matriz A é formada por todas as colunas, exceto a última, e o vetor B é composto pela última coluna.

Os resultados são armazenados no arquivo de saída, denominado "diferenca_finita.txt", que contém o vetor solução x. Cada elemento é formatado com precisão de 8 casas decimais, e os elementos são separados por vírgula e espaço para facilitar a interpretação e análise dos resultados. Essa abordagem proporciona uma organização clara e eficiente dos dados, simplificando tanto a entrada quanto a saída de informações neste código específico.

Problema teste

- **Entrada**



```
entradas > [icon] diferencas_finitas.txt
You, 30 minutes ago | 1 author (You)

1  2.04 -1.0 0.0 0.0 40.8
2  -1.0 2.04 -1.0 0.0 0.8
3  0.0 -1.0 2.04 -1.0 0.8
4  0.0 0.0 -1.0 2.04 200.8
```

Imagem 6.1

- **Saída**



```
saidas > [icon] diferencas_finitas.txt
You, 30 minutes ago | 1 author (You)

1  65.96983437
2  93.77846211
3  124.53822833
4  159.47952369
```

Imagem 6.2

Dificuldades enfrentadas

A tentativa de gerar a matriz usando a função do arquivo de texto não foi bem-sucedida, levando à necessidade de inserção direta dos valores da matriz no conteúdo a ser lido.

Essa adaptação foi realizada devido à identificação da falta de funcionalidade no arquivo mencionado para criar a matriz conforme desejado. Embora tenha resolvido o problema imediatamente, essa experiência destaca a importância de revisar e adaptar métodos de entrada de dados para garantir a correta execução do programa. Isso ressalta a flexibilidade necessária ao lidar com diversas fontes de dados e destaca a importância da resolução eficiente de desafios técnicos para assegurar a robustez e eficácia do código.

Método 7 (Sistema Edo)

Estratégia de Implementação

Inicialmente, foram realizadas as importações essenciais das bibliotecas NumPy e `scipy.integrate`, reconhecidas por sua utilidade em operações numéricas e resolução de equações diferenciais ordinárias (EDOs). Em seguida, a função `"sistema()"` foi definida para representar o conjunto de equações diferenciais, estabelecendo variáveis independentes, como j e k , juntamente com parâmetros a e b . Essa organização visa não apenas eficácia no código, mas também flexibilidade para modificações futuras.

A leitura dos dados de entrada foi realizada a partir do arquivo `"edo.txt"`, assumindo que o formato do arquivo estava correto e continha as informações necessárias. A geração dos pontos de integração t foi efetuada usando a função `np.linspace`, criando um vetor de pontos equidistantes no intervalo de tempo $[t_{\text{inicio}}, t_{\text{fim}}]$. A quantidade de pontos foi especificada no arquivo de entrada, oferecendo flexibilidade na determinação da resolução da integração.

A função `"odeint"` da biblioteca `scipy.integrate` foi empregada para resolver numericamente o sistema de equações diferenciais. Recebendo a função `'sistema'`, as condições iniciais `'y_inicial'` e o vetor de pontos t como argumentos, a função armazenou o resultado da integração na variável `'sol'`, contendo os valores das variáveis dependentes do j e k em cada ponto de integração. Essa abordagem permite a resolução eficiente e flexível de sistemas de EDOs.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada, `"edo.txt"`, segue uma estrutura organizada para especificar os dados necessários, incluindo condições iniciais, tempo inicial e final, e número de pontos de integração. Essa abordagem visa clareza e concisão na definição de parâmetros, facilitando a leitura e permitindo ajustes rápidos no sistema, quando necessário.

No arquivo de saída, `"edo.txt"`, cada linha apresenta os valores correspondentes a um ponto de integração, incluindo tempo, j e k . Essa escolha de formato busca oferecer uma saída legível e compreensível, representando instantâneas temporais para facilitar a identificação dos resultados ao longo do tempo. A estrutura adotada nos arquivos visa

otimizar a organização e interpretação dos dados, contribuindo para uma análise eficaz e possibilitando ajustes rápidos no sistema, se necessário.

Problema teste

- **Entrada**

```
entradas > edo.txt
You, 39 minutes ago | 1 author (You)
1 4.0 6.0
2 1.0 0.0
3 10
```

Imagem 7.1

- **Saída**

```
saídas > edo.txt > data
You, 39 minutes ago | 1 author (You)
1 t = 1.000, j = 4.000, k = 6.000
2 t = 0.889, j = 3.907, k = 6.473
3 t = 0.778, j = 3.851, k = 7.006
4 t = 0.667, j = 3.830, k = 7.604
5 t = 0.556, j = 3.844, k = 8.272
6 t = 0.444, j = 3.895, k = 9.017
7 t = 0.333, j = 3.981, k = 9.845
8 t = 0.222, j = 4.104, k = 10.765
9 t = 0.111, j = 4.266, k = 11.785
10 t = 0.000, j = 4.466, k = 12.913
```

Imagem 7.2

Dificuldades enfrentadas

Após a implementação do código, enfrentamos desafios na compreensão dos parâmetros da estrutura do arquivo de entrada, resultando em dificuldades na execução

do programa. Destaca-se a importância de uma revisão detalhada na saída, especialmente nos arquivos de apoio. É crucial realizar uma análise crítica da saída, ressaltando a necessidade de uma abordagem aprofundada na revisão dos arquivos de suporte, visto que a falta de verificação prévia desses elementos pode ter contribuído para inconsistências na saída.

Recomenda-se uma revisão cuidadosa dos parâmetros e uma análise criteriosa dos arquivos de apoio para aprimorar a confiabilidade e precisão do processo, assegurando a integridade e exatidão dos resultados. Essa abordagem contribuirá significativamente para a validação e confiabilidade do trabalho.

Conclusão

Resumindo, a elaboração deste relatório sobre os métodos de análise numérica me proporcionou uma valiosa experiência. Durante o processo de pesquisa e redação, aprofundei meu entendimento sobre esses métodos e sua aplicação na resolução de problemas numéricos.

Ao explorar o método de Euler, pude compreender sua simplicidade e facilidade de implementação, embora tenha ficado claro que ele pode ser impreciso em contextos mais complexos. Por outro lado, o método de Euler modificado revelou-se uma alternativa mais precisa, embora demande um custo computacional ligeiramente maior. Tanto o método de Heun quanto o método de Ralston se destacaram pela precisão e robustez, oferecendo resultados mais confiáveis em comparação com o método de Euler.

A investigação dos sistemas de equações diferenciais ordinárias foi particularmente instigante. A compreensão das técnicas de diferenças finitas permitiu a discretização de problemas contínuos, proporcionando uma abordagem numérica para a solução. Além disso, o método de shooting mostrou-se uma ferramenta útil na resolução de problemas de valor de contorno, abrindo caminho para lidar com desafios mais complexos.

Embora não tenha concluído a implementação dos métodos de sistemas de EDO e shooting, o estudo me proporcionou uma compreensão substancial de seu funcionamento, tornando a experiência de aprendizado valiosa.

Por fim, a análise do método de Runge-Kutta de quarta ordem destacou sua superioridade em termos de precisão e estabilidade, apesar do custo computacional um pouco mais elevado. Sua aplicabilidade ampla e confiabilidade o posicionam como uma escolha frequente em problemas de análise numérica.

Este relatório enfatiza a relevância da análise numérica na resolução de problemas práticos e reais, onde a obtenção de soluções analíticas pode ser impraticável ou extremamente complexa. Os métodos examinados oferecem ferramentas valiosas para a obtenção de soluções numéricas e viabilizam a análise de fenômenos em diversas áreas, como física, engenharia, finanças e outras.