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 do último crédito da disciplina de análise numérica.

Docente: Gesil Sampaio Amarantes

ILHÉUS – BAHIA

2024

Sumário

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 3.12.4, 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:

1. pip install sympy
2. 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 1.90.2 e o download poder ser feito pelo link: code.visualstudio.com.

Teste e funcionamento do código

Para executar o código dos métodos que vão ser apresentado, é necessário certificar-se que o interpretador python e as bibliotecas estão devidamente instaladas na máquina.

Abra a pasta contendo os métodos, entradas teste e arquivo com os resultados. Clique na pasta com o nome *"entrada"* e altere o respectivo arquivo teste do método que deseja executar, lembrando de manter sempre a formatação padrão de cada entrada de método. Depois, clique no arquivo com o nome "*exemplos.py*" e após aberto clique em "*run code" (Ctrl + Alt + N)*. Em seguida, o arquivo *"resultados"* com a extensão "*.txt"* vai ser atualizado com as saídas, isso vale para todos os métodos implementados.

Na pasta onde se encontra as entradas teste estão armazenadas em cada entrada de método, algumas opções de teste para os problemas desenvolvidos nesse relatório. Como também um arquivo com o nome "*enTeste*" que contém todos os testes contido em um único arquivo.

Método Euler

Estratégias de implementação

A iteração com os dados foi feita utilizando arquivos externos (*.txt*), foi necessário o uso da biblioteca "*math*" para os cálculos, a função é definida utilizando a expressão lida do arquivo, e o método Euler é chamado com os parâmetros que foi extraído do arquivo de entrada. Os parâmetros estão da seguintes formas: a função que representa a EDO, o ponto inicial (x0 e y0), o tamanho do passo, e o número total de iterações. O método de Euler é então aplicado para avançar na solução da EDO, calculando novos valores de x e y em cada iteração.

O critério de parada é definido pela variável n, que representa o número total de iterações. O loop *for _ in range (n)* itera n vezes, avançando a solução da EDO em passos de tamanho h.

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, dessa forma, a primeira linha está a função, na segunda, a quantidade de pontos y0 , na terceira: o valor do intervalo inicial (a), na quarta: o valor do intervalo final (b) e por fim, h. Essa estrutura foi pensada de maneira intuitiva, já que a ordem dessa dispersão apenas interfere mais na estética do que na organização em si. Essas informações podem ser editadas no arquivo "*entrada_euler.txt*" dentro da pasta entrada.

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado resultado. Este método para ser mais específico está na primeira linha. As saídas de cada método estão organizadas da seguinte maneira:

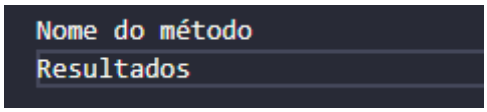


Figura 1

Os resultados deste método assim como na implementação, contém em cada linha, o valor de x na iteração i e o resultado do cálculo naquele ponto.

Problema

Exemplo 1

Para este problema temos os seguintes dados de entrada:

$f(x) = 0.075 * y$

$n: 40 \quad x_0: 0 \quad y_0: 10000 \quad h: 0.5$

E foram obtidas as seguintes saídas:

RESULTADO EULER		RESULTADO EULER MODIFICADO	
x_i	y_i	x_i	y_i
[0] = 0.0	10000.000000	[0] = 0.0	10000.000000
[1] = 0.5	10375.000000	[1] = 0.5	10382.031250
[2] = 1.0	10764.062500	[2] = 1.0	10778.657288
[3] = 1.5	11167.714844	[3] = 1.5	11190.435679
[4] = 2.0	11586.504150	[4] = 2.0	11617.945292
[5] = 2.5	12020.998056	[5] = 2.5	12061.787109
[6] = 3.0	12471.785483	[6] = 3.0	12522.585069
[7] = 3.5	12939.477439	[7] = 3.5	13000.986952
[8] = 4.0	13424.707843	[8] = 4.0	13497.665282
[9] = 4.5	13928.134387	[9] = 4.5	14013.318276
[10] = 5.0	14450.43942	[10] = 5.0	10000.000000
...	...	...	...
[32] = 16.0	32480.250670	[31] = 15.5	31970.710287
[33] = 16.5	33698.260070	[32] = 16.0	33192.091329
[34] = 17.0	34961.944822	[33] = 16.5	34460.132943
[35] = 17.5	36273.017753	[34] = 17.0	35776.617709
[36] = 18.0	37633.255919	[35] = 17.5	37143.396308
[37] = 18.5	39044.503016	[36] = 18.0	38562.390120
[38] = 19.0	40508.671879	[37] = 18.5	40035.593930
[39] = 19.5	42027.747074	[38] = 19.0	41565.078729
[40] = 20.0	43603.787590	[39] = 19.5	43152.994627

Exemplo 2

Para este problema temos os seguintes dados de entrada:

$f(x) = (200 - 2y)/(200 - x)$

$n: 50 \quad x_0: 0 \quad y_0: 0 \quad h: 1$

E foram obtidas as seguintes saídas:

RESULTADO EULER		RESULTADO EULER MODIFICADO	
x_i	y_i	x_i	y_i
[0] = 0.0	0.000000	[0] = 0.0	0.000000
[1] = 1.0	10.000000	[1] = 1.0	9.974874
[2] = 2.0	19.949749	[2] = 2.0	19.899749
[3] = 3.0	29.849246	[3] = 3.0	29.774625
[4] = 4.0	39.698492	[4] = 4.0	39.599501
[5] = 5.0	49.497487	[5] = 5.0	49.374378
[6] = 6.0	59.246231	[6] = 6.0	59.099256
[7] = 7.0	68.944724	[7] = 7.0	68.774134
[8] = 8.0	78.592965	[8] = 8.0	78.399012
[9] = 9.0	88.190955	[9] = 9.0	87.973892
[10] = 10.0	97.738693	[10] = 10.0	97.498772
...	...	...	...
[41] = 41.0	368.793970	[41] = 41.0	367.970361
[42] = 42.0	376.733668	[42] = 42.0	375.895261
[43] = 43.0	384.623116	[43] = 43.0	383.770162
[44] = 44.0	392.462312	[44] = 44.0	391.595063
[45] = 45.0	400.251256	[45] = 45.0	399.369965
[46] = 46.0	407.989950	[46] = 46.0	407.094867
[47] = 47.0	415.678392	[47] = 47.0	414.769770
[48] = 48.0	423.316583	[48] = 48.0	422.394674
[49] = 49.0	430.904523	[49] = 49.0	429.969578

Dificuldades enfrentadas

Não houve dificuldades

Método de Heun (com múltiplas iterações)

Estratégia de implementação

A interação com os dados foi feita utilizando arquivos externos (*.txt*). Foi necessário o uso da biblioteca math.

A escolha do critério de parada baseia-se na predefinição de um número fixo de iterações. No entanto, é importante notar que a escolha de um número fixo de iterações pode não ser a abordagem mais flexível em todos os casos.

O método está implementado da seguinte forma, vamos primeiro para os parâmetros:

- f: Define a função que descreve a variável
- x0 e y0: As condições iniciais
- h: O tamanho do passo
- iteração: O número de iterações (quantidade de pontos intermediários que vamos calcular)

Vamos as iterações:

O loop for itera iterações vezes, a cada iteração, é calculado o valor de y usando o método de Heun, mas, antes, calculamos uma estimativa usando o método de Euler. Em seguida, atualizamos y usando a média ponderada entre a derivada no ponto atual e a derivada no próximo ponto.

Atualizamos o valor de x adicionando y a ele, e então, adicionamos o par ordenado (x, y) na lista *solução.*

Estrutura dos arquivos de entrada\saída

Assim como nos métodos anteriores o arquivo de entrada é organizado de maneira que, em cada linha esteja armazenado um dos dados necessários para o cálculo do método, dessa forma, a primeira linha está a função, na segunda, a quantidade de pontos y0 , na terceira: o valor do intervalo inicial (a), na quarta: o valor do intervalo final (b) e por fim, h. Essa estrutura foi pensada de maneira intuitiva, já que a ordem dessa dispersão apenas interfere mais na estética do que na organização em si. Essas informações podem ser editadas no arquivo "*entrada_heun.txt*" dentro da pasta entrada.

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado resultado. Este método para ser mais específico está na primeira linha. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Foram utilizados o mesmo problema do método anterior, a fim de, fazer uma análise posterior, foram eles:

$f(x) = 0.075 * y$

$n: 40 \quad x_0: 0 \quad y_0: 10000 \quad h: 0.5$

Após a execução do código obtemos:

RESULTADO HEUN	
x_i	y_i
[0] = 0.00	10000.000000
[1] = 0.50	10382.031250
[2] = 1.00	10778.657288
[3] = 1.50	11190.435679
[4] = 2.00	11617.945292
[5] = 2.50	12061.787109
[6] = 3.00	12522.585069
[7] = 3.50	13000.986952
[8] = 4.00	13497.665282
[9] = 4.50	14013.318276
[0] = 0.00	10000.000000
...	...
[31] = 15.50	31970.710287
[32] = 16.00	33192.091329
[33] = 16.50	34460.132943
[34] = 17.00	35776.617709
[35] = 17.50	37143.396308
[36] = 18.00	38562.390120
[37] = 18.50	40035.593930
[38] = 19.00	41565.078729

[39] = 19.50	43152.994627

Exemplo 2

Foram utilizados o mesmo problema do método anterior, a fim de, fazer uma análise posterior, foram eles:

$f(x) = (200 - 2y)/(200 - x)$

$n: 50 \quad x_0: 0 \quad y_0: 0 \quad h: 1$

E foram obtidas as seguintes saídas:

RESULTADO HEUN	
x_i	y_i
[0] = 0.00	0.000000
[1] = 1.00	9.974874
[2] = 2.00	19.899749
[3] = 3.00	29.774625
[4] = 4.00	39.599501
[5] = 5.00	49.374378
[6] = 6.00	59.099256
[7] = 7.00	68.774134
[8] = 8.00	78.399012
[9] = 9.00	87.973892
[10] = 10.00	97.498772
[41] = 41.00	367.970361
[42] = 42.00	375.895261
[43] = 43.00	383.770162
[44] = 44.00	391.595063
[45] = 45.00	399.369965
[46] = 46.00	407.094867
[47] = 47.00	414.769770
[48] = 48.00	422.394674
[49] = 49.00	429.969578

Dificuldades enfrentadas

Não houve dificuldades

Método de Ralston

Estratégias de implementação

A iteração com os dados foi feita utilizando arquivos externos (.txt). foi necessário o uso da biblioteca *sympy* para leitura e conversão da função.

Assim como nos métodos anteriores, o critério de parada é baseado no número de iterações definido pelo parâmetro 'n' na função *ralston_method*. O método foi implementado da seguinte forma:

A função f(x, y) representa a derivada da variável dependente y em relação a x, ela é definida usando a expressão *eval (func).*

Na função do método foi definido os seguintes parâmetros:

- $X0\ e\ y0$: As condições iniciais
- h: O tamanho do passo (intervalos entre os pontos)
- n: O número de iterações (o número de pontos intermediários que queremos calcular)

Estrutura dos arquivos de entrada\saída

Assim como nos métodos anteriores o arquivo de entrada é organizado de maneira que, em cada linha esteja armazenado um dos dados necessários para o cálculo do método, dessa forma, a primeira linha está a função, na segunda, a quantidade de pontos y0 , na terceira: o valor do intervalo inicial (a), na quarta: o valor do intervalo final (b) e por fim, h. Essa estrutura foi pensada de maneira intuitiva, já que a ordem dessa dispersão apenas interfere mais na estética do que na organização em si. Essas informações podem ser editadas no arquivo "*entrada_ralston.txt*" dentro da pasta entrada.

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado resultado. Este método para ser mais específico está na primeira linha. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problema

Exemplo 1

Foram utilizados o mesmo problema do método anterior, a fim de, fazer uma análise posterior, foram eles:

$f(x) = 0.075 * y$

$n: 40 \quad x_0: 0 \quad y_0: 10000 \quad h: 0.5$

Após a execução do código obtemos:

RESULTADO RALSTON	
x_i	y_i
[0] = 0.0	10000.000000
[1] = 0.5	10382.031250
[2] = 1.0	10778.657288
[3] = 1.5	11190.435679
[4] = 2.0	11617.945292
[5] = 2.5	12061.787109
[6] = 3.0	12522.585069
[7] = 3.5	13000.986952
[8] = 4.0	13497.665282
[9] = 4.5	14013.318276
[0] = 0.0	10000.000000
...	...
[31] = 15.5	31970.710287
[32] = 16.0	33192.091329
[33] = 16.5	34460.132943
[34] = 17.0	35776.617709
[35] = 17.5	37143.396308
[36] = 18.0	38562.390120
[37] = 18.5	40035.593930
[38] = 19.0	41565.078729
[39] = 19.5	43152.994627

Exemplo 2

Foram utilizados o mesmo problema do método anterior, a fim de, fazer uma análise posterior, foram eles:

$f(x) = (200 - 2y)/(200 - x)$

$n: 50 \quad x_0: 0 \quad y_0: 0 \quad h: 1$

E foram obtidas as seguintes saídas:

RESULTADO RALSTON	
x_i	y_i
[0] = 0.0	0.000000
[1] = 1.0	9.974906
[2] = 2.0	19.899812
[3] = 3.0	29.774719
[4] = 4.0	39.599626
[5] = 5.0	49.374534
[6] = 6.0	59.099442
[7] = 7.0	68.774351
[8] = 8.0	78.399260
[9] = 9.0	87.974170
[10] = 10.0	97.499080
...	...
[41] = 41.00	367.975000
[42] = 42.00	375.900000
[43] = 43.00	383.775000
[44] = 44.00	391.600000
[45] = 45.00	399.375000
[46] = 46.00	407.100000
[47] = 47.00	414.775000
[48] = 48.00	422.400000
[49] = 49.00	429.975000

Método de Runge-Kutta 4ª Ordem

Estratégia de Implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca *math* para um melhor funcionamento dos cálculos.

A função foi pensada da seguinte forma, primeiro vamos explicar os parâmetros:

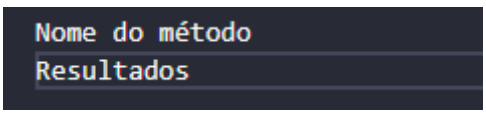
- f: que representa a derivada de y em relação a x
- x_0 *e* y_0: São as condições iniciais de x e y
- h: o tamanho do passo para a integração
- n: o número de passos a ser tomados

São criadas duas listas para armazena os dados de x e y. Um loop executa o método quatro vezes para calcular os valores intermediários $k1, k2, k3$ e $k4$. Esses valores são usados para calcular o próximo valor de y, utilizando uma média ponderada. O próximo valor de x é obtido com o valor atual somado ao tamanho do passo. A função retorna duas listas contendo os valores de x e y.

Estrutura dos arquivos de entrada\saída

Assim como nos métodos anteriores o arquivo de entrada é organizado de maneira que, em cada linha esteja armazenado um dos dados necessários para o cálculo do método, dessa forma, a primeira linha está a função, na segunda, a quantidade de pontos y0 , na terceira: o valor do intervalo inicial (a), na quarta: o valor do intervalo final (b) e por fim, h. Essa estrutura foi pensada de maneira intuitiva, já que a ordem dessa dispersão apenas interfere mais na estética do que na organização em si. Essas informações podem ser editadas no arquivo "*entrada_runge.txt*" dentro da pasta entrada.

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado resultado. Este método para ser mais específico está na primeira linha. As saídas de cada método estão organizadas da seguinte maneira:



Problema

Exemplo 1

Foram utilizados o mesmo problema do método anterior, a fim de, fazer uma análise posterior, foram eles:

$f(x) = 0.075 * y$

$n: 40 \quad x_0: 0 \quad y_0: 10000 \quad h: 0.5$

Após a execução do código, obtemos a seguinte saída:

RESULTADO RUNGE – KUTTA DE 4ª ORDEM	
x_i	y_i
[0] = 0.00	10000.000000
[1] = 0.50	10382.119965
[2] = 1.00	10778.841496
[3] = 1.50	11190.722549
[4] = 2.00	11618.342399
[5] = 2.50	12062.302458
[6] = 3.00	12523.227117
[7] = 3.50	13001.764627
[8] = 4.00	13498.588011
[9] = 4.50	14014.396008
[0] = 0.00	10000.000000
...	...
[31] = 15.50	31979.180268
[32] = 16.00	33201.168591
[33] = 16.50	34469.851528
[34] = 17.00	35787.013372
[35] = 17.50	37154.506600
[36] = 18.00	38574.254475
[37] = 18.50	40048.253751
[38] = 19.00	41578.577481
[39] = 19.50	43167.377937

Exemplo 2

Foram utilizados o mesmo problema do método anterior, a fim de, fazer uma análise posterior, foram eles:

$f(x) = (200 - 2y)/(200 - x)$

$n: 50 \quad x_0: 0 \quad y_0: 0 \quad h: 1$

E foram obtidas as seguintes saídas:

RESULTADO RUNGE – KUTTA DE 4ª ORDEM	
x_i	y_i
[0] = 0.00	0.000000
[1] = 1.00	9.975000
[2] = 2.00	19.900000
[3] = 3.00	29.775000
[4] = 4.00	39.600000
[5] = 5.00	49.375000
[6] = 6.00	59.100000
[7] = 7.00	68.775000
[8] = 8.00	78.400000
[9] = 9.00	87.975000
[10] = 10.00	97.500000
...	...
[41] = 41.00	367.975000
[42] = 42.00	375.900000
[43] = 43.00	383.775000
[44] = 44.00	391.600000
[45] = 45.00	399.375000
[46] = 46.00	407.100000
[47] = 47.00	414.775000
[48] = 48.00	422.400000
[49] = 49.00	429.975000

Dificuldades enfrentadas

Não houve dificuldades.

Método dos sistemas de EDO’s

Estratégia de Implementação

Para implementar o método de sistemas de equações diferenciais ordinárias (EDO’s) em python, a estratégia que foi tomada, começa com a definição rigorosa do sistema de EDO’s, expressando cada equação como dy/dt.

Primeiro vamos a função *f()*, esta função define o sistemas de EDO’s. Ela recebe as variáveis independentes e dependentes e retorna as derivadas de y em relação a z e de z em relação a x.

A função Euler(), aplica um passo do método de Euler ao sistema de EDO’s, calculando as derivadas usando a função f() e atualiza os valores de y e z com base no tamanho do passo h.

Já a função sistemas_edo() resolve o sistema de EDO’s ao longo de um vetor de valores x. Depois de fazer todo o loop a função retorna duas listas: y_n contendo as soluções para y e z_n contendo as soluções para z.

Estrutura dos arquivos de entrada\saída

Assim como nos métodos anteriores o arquivo de entrada é organizado de maneira que, em cada linha esteja armazenado um dos dados necessários para o cálculo do método, dessa forma, a primeira linha está o valor para x, na segunda, a quantidade de pontos y_0, na terceira: o valor de z_0, na quarta: o valor do passo h. Essa estrutura foi pensada de maneira intuitiva, já que a ordem dessa dispersão apenas interfere mais na estética do que na organização em si. Essas informações podem ser editadas no arquivo “*entrada_edos.txt*” dentro da pasta entrada.

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado resultado. Este método para ser mais específico está na primeira linha. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Para este problema foi considerado o seguinte sistema:

$y' = z$

$z' = y + e^x$

Com: $y_0 = 1, z_0 = 0$ e $\in [0{,}02]$

Passo h = 0.1

Após a execução do script temos os seguintes resultados:

RESULTADO DO SISTEMA DE EDO'S		
x	y	z
0.00	1.0000	0.2000
0.01	1.0200	0.4105
0.02	1.0611	0.6347

Dificuldades enfrentadas

Devido aos desafios enfrentados na compreensão dos slides de referência e na busca pela resolução correta dos problemas propostos, a conclusão da implementação do método sugerido se mostrou inviável. Creio também que a dificuldade se deu pela complexidade decorrente do trabalho com matrizes. Apesar dos esforços direcionados à compreensão e à pesquisa adicional, não foi possível alcançar uma implementação satisfatória. Consequentemente, a execução do método não pôde ser realizada.

Método das diferenças finitas

Estratégia de Implementação

A iteração com os dados foi feita utilizando arquivos externos (.txt). foi necessário o uso da biblioteca *sympy* para um melhor funcionamento do código.

Vamos primeiro as funções *p(), q()* e *r()* estas funções definem os coeficientes da equação diferencial ordinária de segunda ordem.

Na função *diferenças_finitas()* relvemos o problema de valor de contorno usando o método de diferenças finitas. Ela recebe como parâmetro; x_0, y_0 (inicial) e x_f , y_f (final) além do número de intervalos.

A matriz a e o vetor b são montados com base nos coeficientes da equação diferencial e nos valores de contorno. A matriz é tri-diagonal, refletindo a natureza do método de diferenças finitas para EDO's de segunda ordem.

Por fim, o sistema é resolvido usando a função *np.linalg.solve(A, b)*.

A solução y representa os valores aproximados da função desconhecida no interior dos pontos da malha, excluindo os pontos de contorno.

Estrutura dos arquivos de entrada\saída

Assim como nos métodos anteriores o arquivo de entrada é organizado de maneira que, em cada linha esteja armazenado um dos dados necessários para o cálculo do método, dessa forma, a primeira linha está o valor que corresponde ao x_0, na segunda, y_0 , na terceira: o valor de x_f, na quarta: o valor de y_f e por fim, n. Essas informações podem ser editadas no arquivo "*entrada_DiferencasFinitas.txt*" dentro da pasta entrada.

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado resultado. Este método para ser mais específico está na primeira linha. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Para esse problema foi utilizada a EDO: $y'' = x/3 * y' + 6x - 1$

Com valores iniciais: $x_0 = 0$ e $y_0 = -1$

Valores de contorno foram admitidos: $x_f = 1$ e $y_f = 0$

Após a execução do script foi obtida o seguinte resultado:

RESULTADO PARA DIFERENÇAS FINITAS
$y(0.25) = -0.5376344086021506$
$y(0.5) = -0.3477858711989826$
$y(0.75) = -0.2980150604768778$

Dificuldades enfrentadas

Não houve dificuldades

Método de Shooting

Estratégia de Implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca sympy para leitura e conversão da função e a biblioteca *math*.

Vamos a função *shooting()*, esta função aplica o método de *shooting* para encontrar a solução de uma EDO que satisfaça as condições de contorno. Nos parâmetros temos; A *função* que é a segunda derivada da função desconhecida y, e *y_inicial* é o valor inicial de y, já o *intervalo*, é uma lista de dois valores iniciais para a derivada de y, o parâmetro *valor* é valor de contorno final desejado e o h é o tamanho do passo e pontos é o número de pontos da malha.

A função inicializa três listas em x, y e z para armazenar os valores de x, y e a primeira derivada de y. Um loop externo itera sobre os valores iniciais da derivada de y, e um loop interno aplica o método de Euler para avançar a solução ao longo da malha.

Após duas primeiras iterações, o código ajusta o chute inicial da derivada de y usando uma interpolação linear. Se a condição de contorno final for satisfeita, a função retorna os valores de y ao longo da malha.

Estrutura dos arquivos de entrada\saída

O arquivo de entrada contém todos os dados em uma única linha. Que estão dispostos da seguinte maneira; função, valor inicial de y, intervalo de chute separado por virgula, valor final desejado, que são representados por h e n, respectivamente. Essas informações podem ser editadas no arquivo "*entrada_shooting.txt*" dentro da pasta entrada.

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado resultado. Este método para ser mais específico está na primeira linha. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Para esse método foi utilizado o exemplo do slide com 10 pontos internos e esse foram os resultados obtidos:

RESULTADO DO MÉTODO DE SHOOTING	
Iteração	Resultado
0	20
1	37.982
2	55.968
3	73.958
4	91.951
5	109.949
6	127.951
7	145.957
8	163.967
9	181.982
10	200.000000000000

Dificuldades enfrentadas

Não houve dificuldade.

Considerações finais

Ao decorrer do desenvolvimento desse projeto, foi possível implementar com sucesso alguns métodos numéricos empregados na resolução de equações diferenciais ordinárias. Destacando esses métodos: Shooting, Diferenças Finitas, Runge-Kutta de 4ª ordem, Ralston, Heun, Euler e Euler Modificado, que foram implementados e verificados de maneira adequada, gerando resultados consistentes.

Analisando os resultados entre o método de Euler simples e Euler modificado, podemos perceber que o Euler modificado para a equação $f(x) = 0.075 * y$ com $n: 40 \;\; x_0: 0 \;\; y_0: 10000 \;\; h: 0.5$ obteve um desempenho melhor, isso é evidente pelo fato dos valores de y_i serem consistentemente maiores do que do Euler simples o que sugere uma maior precisão na aproximação da solução real da equação diferencial. Se observamos a última linha, para x = 20.0, veremos que há uma diferença significativa entre os valores finais indicando uma solução mais exata da equação diferencial para este caso.

Executando o código e mantendo os valores para n, x_0, y_0 e h com os métodos de Heun, Ralston e Runge. Não houve mudanças significativas em comparação ao método do Euler modificado. A semelhança nos resultados sugere que esses métodos estão fornecendo uma aproximação numérica muito próxima um do outro para a solução da equação diferencial com os parâmetros fornecidos. Isso é esperado, pois, todos, são métodos de segunda ordem e, portanto, tem uma precisão semelhante para o tamanho do passo fornecido.

Para a equação $f(x) = (200 - 2y)/(200 - x)$ com $n: 50 \;\; x_0: 0 \;\; y_0: 0 \;\; h: 1$, temos algo parecido. Podemos ver os resultados do método de Euler e do Euler modificado é ligeiramente diferente. Isso já é esperado, pois, o método de Euler modificado é mais preciso que o Euler simples. A diferença nos resultados entre os dois indica que a correção feita no método de Euler modificado está proporcionando uma aproximação mais precisa da solução real da equação diferencial.

Quanto ao método de Heun, os resultados são idênticos ao do Euler modificado. Isso faz sentido, pois, os dois métodos são uma variação do método de Runge Kutta de segunda ordem e, portanto, têm precisão semelhante.

Agora os resultados dos métodos de Ralston e Runge Kutta de 4ª ordem, são parecidos, mas, não idênticos aos métodos de Heun e Euler modificado. Isso é até esperado já que o método de Runge é de quarta ordem, quanto o de Euler, Ralston e Heun são de segunda ordem. Essas pequenas diferenças nos resultados entre os métodos são completamente normais e refletem as características e a precisão de cada método numérico.

Para determinar qual é o método melhor, para esses casos em específico, é preciso levar em consideração outros fatores além da precisão, como complexidade computacional ou a facilidade de implementação.

Quantos aos últimos métodos, deparando-me com obstáculos ao longo do processo, não consegui elaborar a implementação correta do método de Sistemas de EDO’s. Mesmo com os esforços para compreender e utilizar esse método de forma coesa, enfrentei entraves na busca por soluções adequadas para os problemas propostos. Foi feita a implementação, mas, não foi possível gerar resultados que possam ser comparados entre si, a fim de fazer uma análise coerente.

Por fim, foi verificado que alguns métodos são mais eficazes quando se trata de precisão, já outros em velocidade. Entretanto, todos os métodos que foram implementados e discorridos neste relatório, alcançaram resultados satisfatórios.