



Universidade Estadual de Santa Cruz – UESC

**Relatório de Implementações de Métodos da Disciplina Análise
Numérica - parte III**

**Relatório de implementações realizadas
por Flávia Alessandra S J.**

Disciplina Análise Numérica.

Curso Ciência da Computação

Semestre 2023.2

Professor Gesil Sampaio Amarante II

**Ilhéus – BA
2023**

Linguagem Escolhida e justificativas.....	3
Python e Visual Studio Code.....	3
Testes e Funcionamento do Código.....	3
Método de Euler.....	4
Estratégia de Implementação.....	4
Estrutura dos Arquivos de Entrada/Saída.....	4
Problemas Teste.....	5
Dificuldades Enfrentadas.....	6
Método de Euler Modificado.....	7
Estratégia de Implementação.....	7
Estrutura dos Arquivos de Entrada/Saída.....	7
Problemas Teste.....	8
Dificuldades Enfrentadas.....	9
Método de Heun (com múltiplas iterações).....	10
Estratégia de Implementação.....	10
Estrutura dos Arquivos de Entrada/Saída.....	10
Problemas Teste.....	11
Dificuldades Enfrentadas.....	12
Método de Ralston.....	13
Estratégia de Implementação.....	13
Estrutura dos Arquivos de Entrada/Saída.....	13
Problemas Teste.....	14
Dificuldades Enfrentadas.....	15
Método de Runge-Kutta de 4ª Ordem.....	16
Estratégia de Implementação.....	16
Estrutura dos Arquivos de Entrada/Saída.....	16
Problemas Teste.....	17
Dificuldades Enfrentadas.....	18
Método dos Sistemas de EDOs.....	19
Estratégia de Implementação.....	19
Dificuldades Enfrentadas.....	19
Método das Diferenças Finitas.....	20
Estratégia de Implementação.....	20
Estrutura dos Arquivos de Entrada/Saída.....	20
Problema Teste.....	21
Dificuldades Enfrentadas.....	21
Método de Shooting.....	22
Estratégia de Implementação.....	22
Estrutura dos Arquivos de Entrada/Saída.....	22
Problemas Teste.....	23
Dificuldades Enfrentadas.....	23
Considerações Finais.....	24

Linguagem Escolhida e justificativas

Python e Visual Studio Code

Conforme mencionado nos relatórios anteriores, a decisão de adotar a linguagem Python foi motivada por sua facilidade na manipulação de fórmulas matemáticas e operações de leitura e escrita de arquivos. A vantagem proeminente do Python reside especialmente na eficácia ao lidar com fórmulas matemáticas, destacando-se a utilidade da função "eval" para processar essas equações, especialmente quando provenientes de arquivos externos.

No contexto da manipulação de arquivos, o Python oferece funções integradas que simplificam significativamente a leitura e escrita de dados, graças à sua biblioteca padrão. Não foram identificadas limitações no uso desta linguagem, ela atendeu de maneira eficiente a todas as necessidades.

A versão específica do Python empregada foi a 3.7.8, e o interpretador pode ser obtido em: <https://www.python.org>. A instalação das bibliotecas pode ser realizada no terminal do Visual Studio Code (ou prompt de comando) por meio do seguinte comando (após a instalação do interpretador):

- pip install sympy
- pip install numpy

Uma vez instaladas, não é necessário executar esses comandos novamente.

Optou-se pelo uso do editor de texto Visual Studio Code, que dispõe de um compilador interno para diversas linguagens. Esse ambiente pode ser baixado em: <https://code.visualstudio.com>.

Testes e Funcionamento do Código

Para rodar o código de cada método, certifique-se de ter o interpretador Python em sua máquina e as bibliotecas instaladas. Abra a pasta de métodos, clique na pasta do método desejado. Edite o arquivo 'entrada.txt' com a entrada desejada e salve-o. Entre no arquivo .py e clique em 'Run Code' (Ctrl + Alt + N). Após isso, o arquivo 'resultado.txt' será atualizado. Esse passo a passo vale para todos os métodos implementados.

Na pasta onde estão as implementações há um arquivo 'entradasTeste.txt'. Nele, estão armazenadas as opções de entrada para os problemas desenvolvidos neste relatório.

Método de Euler

Estratégia de Implementação

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

Na função *main*, os parâmetros do problema são lidos a partir de um arquivo de entrada. A função *f* é definida utilizando a expressão lida do arquivo, e o método de Euler é chamado com os parâmetros fornecidos. Os resultados são então gravados em um arquivo de saída. A função *euler* recebe como parâmetros a função *f* que representa a EDO, o ponto inicial (x_0 , y_0), o tamanho do passo *h* e o número total de iterações *n*. O método de Euler é então aplicado para avançar na solução da EDO, calculando novos valores de *x* e *y* em cada iteração.

Em relação ao critério de parada, o mesmo 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, na primeira linha se encontra a função, na segunda, a quantidade de pontos y_0 , 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 serão editadas no arquivo “*entrada.txt*” dentro da pasta do método em questão.

```
Euler > ≡ entrada.txt
1  0.075 * y
2  40
3  0
4  10000
5  0.5
```

O arquivo de saída contém em cada linha a iteração *i* e o resultado do cálculo naquele ponto.

```
Euler > ≡ resultado.txt
40  |  ...
41  x[39] = 19.5,  y = 42027.747074
42  x[40] = 20.0,  y = 43603.787590
```

Problemas Teste

12.3		12.10	
f(x): (2000 - 2y) / (200 - x) n: 50 x ₀ : 0 y ₀ : 0 h: 1		f(x): 0.075y n: 40 x ₀ : 0 y ₀ : 10000 h: 0.5	
X _i	Y _i	X _i	Y _i
0:	0.000000	0.5	10000.000000
1:	10.000000	1.0	10375.000000
2:	19.949749	1.5	10764.062500
3:	29.849246	2.0	11167.714844
4:	39.698492	2.5	11586.504150
5:	49.497487	3.0	12020.998056
6:	59.246231	3.5	12471.785483
7:	68.944724	4.0	12939.477439
8:	78.592965	4.5	13424.707843
9:	88.190955	5.0	13928.134387
10:	97.738693	5.5	14450.439426
11:	107.236181	6.0	14992.330905
12:	116.683417	6.5	15554.543314
13:	126.080402	7.0	16137.838688
14:	135.427136	7.5	16743.007639
15:	144.723618	8.0	17370.870425
16:	153.969849	8.5	18022.278066
17:	163.165829	9.0	18698.113494
18:	172.311558	9.5	19399.292750
19:	181.407035	10.0	20126.766228
20:	190.452261	10.5	20881.519961
21:	199.447236	11.0	21664.576960
22:	208.391960	11.5	22476.998596
23:	217.286432	12.0	23319.886043
24:	226.130653	12.5	24194.381770
25:	234.924623	13.0	25101.671086
26:	243.668342	13.5	26042.983752
27:	252.361809	...	
28:	261.005025	15.5	31306.265706
29:	269.597990	16.0	32480.250670
30:	278.140704	16.5	33698.260070
...	...	17.0	34961.944822
45:	400.251256	17.5	36273.017753
46:	407.989950	18.0	37633.255919
47:	415.678392	18.5	39044.503016
48:	423.316583	19.0	40508.671879
49:	430.904523	19.5	42027.747074
50:	438.442211	20.0	43603.787590

12.3 Esse exercício propõe que um corpo com uma massa inicial de 200 kg é acelerado por uma força constante de 200N. A massa decresce a uma taxa de 1 kg/s. Considerando que o corpo está em repouso em $t = 0$, vamos encontrar sua velocidade ao final de 50s.

Temos que a equação diferencial é:

$$\frac{dv}{dt} = \frac{2000 - 2v}{200 - t}.$$

Assim, convertemos os dados para o arquivo de entrada e teremos:

f(x): $(2000 - 2y)/(200 - x)$ **n:** 50 **x₀:** 0 **y₀:** 0 **h:** 1

Ao aplicar a entrada ao método de Euler, temos que a velocidade encontrada é de 438,442211 ao final de 50s.

12.10 Esse exercício trata-se de um estudo sobre o crescimento da população e apresenta um modelo simples, onde o crescimento está sujeito a hipótese de que a taxa de variação da população p é proporcional à população num instante t , ou seja:

$$\frac{dp}{dt} = Gp,$$

onde G é a taxa de variação (por ano). Tomamos então $Gp = f(x) = 0.075y$, **n** = 40, **x₀** = 0, **y₀** = 10000, **h** = 0.5.

Dessa forma, temos que a previsão da população em $t = 20$ anos, usando o comprimento de passo de 0.5 ano é de 43.603,787590.

Dificuldades Enfrentadas

A implementação do método em questão foi efetuada de maneira eficiente, sem inconvenientes notáveis.

Método de Euler Modificado

Estratégia de Implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca *math* para os cálculos. Esse método não requer muitos recursos externos para a implementação.

Em relação ao critério de parada, é usado o número total de iterações especificado pelo valor de *n*. O método de Euler modificado calcula os valores de *x* e *y* para cada iteração até atingir o número desejado de passos definido por *n*. Este é um critério comum de parada em métodos numéricos para EDOs, pois permite controlar a precisão da solução ajustando a quantidade de iterações.

A função principal lê os valores de entrada de um arquivo, define a função a partir da expressão fornecida e chama o método de Euler Modificado. Após o cálculo, os resultados são escritos no arquivo resultado.txt.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada segue uma organização em que cada linha contém um dos dados necessários para calcular o método. Assim, na primeira linha está armazenada 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, o valor de *h*. Essa estrutura foi concebida de maneira intuitiva, considerando que a ordem dessa disposição afeta mais a estética do arquivo do que a organização em si. Essas informações devem ser editadas no arquivo "entrada.txt" dentro da pasta do método em questão.

```
Euler > ≡ entrada.txt
1  0.075 * y
2  40
3  0
4  10000
5  0.5
```

O arquivo de saída, assim como a implementação do método de Euler, contém em cada linha o valor de *x* na iteração *i* e o resultado do cálculo naquele ponto.

```
Euler > ≡ resultado.txt
40  |  ...
41  x[39] = 19.5,  y = 43152.994627
42  x[40] = 20.0,  y = 44801.573875
```

Problemas Teste

12.3		12.10	
f(x): (2000 - 2y) / (200 - x) n: 50 x ₀ : 0 y ₀ : 0 h: 1		f(x): 0.075y n: 40 x ₀ : 0 y ₀ : 10000 h: 0.5	
X _i	Y _i	X _i	Y _i
0:	0.000000	0.5	10000.000000
1:	9.974874	1.0	10382.031250
2:	19.899749	1.5	10778.657288
3:	29.774625	2.0	11190.435679
4:	39.599501	2.5	11617.945292
5:	49.374378	3.0	12061.787109
6:	59.099256	3.5	12522.585069
7:	68.774134	4.0	13000.986952
8:	78.399012	4.5	13497.665282
9:	87.973892	5.0	14013.318276
10:	97.498772	5.5	14548.670825
11:	106.973652	6.0	15104.475515
12:	116.398534	6.5	15681.513682
13:	125.773416	7.0	16280.596509
14:	135.098298	7.5	16902.566172
15:	144.373181	8.0	17548.297021
16:	153.598065	8.5	18218.696805
17:	162.772949	9.0	18914.707957
18:	171.897834	9.5	19637.308909
19:	180.972720	10.0	20387.515476
20:	189.997606	10.5	21166.382278
21:	198.972493	11.0	21975.004226
22:	207.897380	11.5	22814.518060
23:	216.772269	12.0	23686.103945
24:	225.597157	12.5	24590.987135
25:	234.372047	13.0	25530.439690
26:	243.096937	13.5	26505.782269
27:	251.771827	...	
28:	260.396718	15.0	30794.272833
29:	268.971610	15.5	31970.710287
30:	277.496503	16.0	33192.091329
...	...	16.5	34460.132943
44:	391.595063	17.0	35776.617709
45:	399.369965	17.5	37143.396308
46:	407.094867	18.0	38562.390120
47:	414.769770	18.5	40035.593930
48:	422.394674	19.0	41565.078729
49:	429.969578	19.5	43152.994627
50:	437.494483	20.0	44801.573875

12.3 Ao aplicar a entrada ao método de Euler Modificado, temos que a velocidade encontrada é de 437,494483 ao final de 50s. Enquanto que no método anterior (Euler) o resultado foi: 438,442211.

12.10 Ao aplicar a entrada ao método de Euler Modificado, temos que a previsão da população em $t = 20$ anos, usando o comprimento de passo de 0.5 ano é de 44.801,573875. Enquanto que no método anterior (Euler) o resultado foi: 43.603,787590.

Dificuldades Enfrentadas

A implementação do método em questão foi efetuada de maneira eficiente, sem inconvenientes notáveis.

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.

Ele executa um número predeterminado de iterações, inicialmente realizando uma estimativa com o método de Euler e, posteriormente, aplicando correções adicionais para aprimorar a precisão da solução. Os resultados são registrados em uma lista e posteriormente escritos no arquivo resultado.txt.

Estrutura dos Arquivos de Entrada/Saída

Seguindo o mesmo padrão dos métodos anteriores, o arquivo de entrada é estruturado de modo que cada linha contenha um dos dados essenciais para o cálculo do método. Portanto, na primeira linha, encontra-se a função; na segunda, a quantidade de pontos (n); na terceira, o valor do ponto inicial (a); na quarta, o valor do ponto final (b); e, por fim, o valor do passo (h).

```
Heun > ≡ entrada.txt
1  (2000 - 2 * y) / (200 - x)
2  50
3  0
4  0
5  1
```

Cada linha do arquivo de saída apresenta o valor de x na iteração i e o resultado do cálculo correspondente nesse ponto.

```
Heun > ≡ resultado.txt
50  |  |  ...
51  x[49] = 49.00,  y[49] = 429.9696
52  x[50] = 50.00,  y[50] = 437.4945
```

Problemas Teste

12.3	12.10
f(x): $(2000 - 2y) / (200 - x)$ n: 50 x₀: 0 y₀: 0 h: 1	f(x): $0.075y$ n: 40 x₀: 0 y₀: 10000 h: 0.5

X_i	Y_i	X_i	Y_i
0:	0.0000	0	10000.000000
1:	9.9749	0.5	10382.031250
2:	19.8997	1.0	10778.657288
3:	29.7746	1.5	11190.435679
4:	39.5995	2.0	11617.945292
5:	49.3744	2.5	12061.787109
6:	59.0993	3.0	12522.585069
7:	68.7741	3.5	13000.986952
8:	78.3990	4.0	13497.665282
9:	87.9739	4.5	14013.318276
10:	97.4988	5.0	14548.670825
11:	106.9737	5.5	15104.475515
12:	116.3985	6.0	15681.513682
13:	125.7734	6.5	16280.596509
14:	135.0983	7.0	16902.566172
15:	144.3732	7.5	17548.297021
16:	153.5981	8.0	18218.696805
17:	162.7729	8.5	18914.707957
18:	171.8978	9.0	19637.308909
19:	180.9727	9.5	20387.515476
20:	189.9976	10.0	21166.382278
21:	198.9725	10.5	21975.004226
22:	207.8974	11.0	22814.518060
23:	216.7723	11.5	23686.103945
24:	225.5972	12.0	24590.987135
25:	234.3720	12.5	25530.439690
26:	243.0969	13.0	26505.782269
27:	251.7718	13.5	27518.385982
28:	260.3967	...	
29:	268.9716	15.0	30794.272833
30:	277.4965	15.5	31970.710287
...		16.0	33192.091329
43:	383.7702	16.5	34460.132943
44:	391.5951	17.0	35776.617709
45:	399.3700	17.5	37143.396308
46:	407.0949	18.0	38562.390120
47:	414.7698	18.5	40035.593930
48:	422.3947	19.0	41565.078729
49:	429.9696	19.5	43152.994627
50:	437.4945	20.0	44801.573875

12.3 Ao aplicar a entrada ao método de Heun, temos que a velocidade encontrada é de 437,4945 ao final de 50s. Enquanto que nos métodos anteriores os resultados foram: Euler - 438,442211; Euler Modificado - 437,494483.

12.10 Ao aplicar a entrada ao método de Heun, temos que a previsão da população em $t = 20$ anos, usando o comprimento de passo de 0.5 ano é de 44.801,573875. Enquanto que nos métodos anteriores os resultados foram: Euler - 43.603,787590; Euler Modificado - 44.801,573875.

Dificuldades Enfrentadas

A implementação do método em questão foi realizada de forma eficiente e sem contratempos significativos.

Método de Ralston

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.

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 utiliza dois coeficientes para atualizar a solução a cada passo. A implementação utiliza uma lista de pontos para armazenar as coordenadas da solução. Os valores de entrada são lidos de um arquivo .txt, a função é definida a partir de uma expressão fornecida e o método de Ralston é chamado. Os resultados são escritos no arquivo `resultado.txt`.

Estrutura dos Arquivos de Entrada/Saída

Para o arquivo de entrada, foi elaborada uma estrutura organizada, onde cada linha contém um dos dados necessários para o cálculo do método. A disposição dos dados é intuitiva, com a função na primeira linha, a quantidade de pontos (y_0) na segunda, o valor do intervalo inicial (a) na terceira, o valor do intervalo final (b) na quarta e, por último, o valor de h . As informações devem ser atualizadas no arquivo "entrada.txt" localizado na pasta específica do método em questão.

```
Ralston >  entrada.txt
1  (2000 - 2 * y) / (200 - x)
2  50
3  0
4  0
5  1
```

O arquivo de saída contém em cada linha o valor de x na iteração i e o resultado do cálculo naquele ponto.

```
Ralston >  resultado.txt
50  |  |  ...
51  | x[49] = 49.0,  y = 429.970939
52  | x[50] = 50.0,  y = 437.495868
```

Problemas Teste

12.3		12.10	
f(x): (2000 - 2y) / (200 - x) n: 50 x ₀ : 0 y ₀ : 0 h: 1		f(x): 0.075y n: 40 x ₀ : 0 y ₀ : 10000 h: 0.5	
X _i	Y _i	X _i	Y _i
0:	0.000000	0	10000.000000
1:	9.974906	0.5	10382.031250
2:	19.899812	1.0	10778.657288
3:	29.774719	1.5	11190.435679
4:	39.599626	2.0	11617.945292
5:	49.374534	2.5	12061.787109
6:	59.099442	3.0	12522.585069
7:	68.774351	3.5	13000.986952
8:	78.399260	4.0	13497.665282
9:	87.974170	4.5	14013.318276
10:	97.499080	5.0	14548.670825
11:	106.973991	5.5	15104.475515
12:	116.398902	6.0	15681.513682
13:	125.773813	6.5	16280.596509
14:	135.098725	7.0	16902.566172
15:	144.373638	7.5	17548.297021
16:	153.598551	8.0	18218.696805
17:	162.773464	8.5	18914.707957
18:	171.898378	9.0	19637.308909
19:	180.973292	9.5	20387.515476
20:	189.998207	10.0	21166.382278
21:	198.973122	10.5	21975.004226
22:	207.898038	11.0	22814.518060
23:	216.772954	11.5	23686.103945
24:	225.597871	12.0	24590.987135
25:	234.372788	12.5	25530.439690
26:	243.097706	13.0	26505.782269
27:	251.772624	13.5	27518.385982
28:	260.397542	...	
29:	268.972461	15.5	31970.710287
30:	277.497381	16.0	33192.091329
...		16.5	34460.132943
44:	391.596302	17.0	35776.617709
45:	399.371229	17.5	37143.396308
46:	407.096156	18.0	38562.390120
47:	414.771083	18.5	40035.593930
48:	422.396011	19.0	41565.078729
49:	429.970939	19.5	43152.994627
50:	437.495868	20.0	44801.573875

12.3 Ao aplicar a entrada ao método de Ralston, temos que a velocidade encontrada é de 437,495868 ao final de 50s. Enquanto que nos métodos anteriores os resultados foram: Euler - 438,442211; Euler Modificado - 437,494483; Heun - 437,4945.

12.10 Ao aplicar a entrada ao método de Ralston, temos que a previsão da população em $t = 20$ anos, usando o comprimento de passo de 0.5 ano é de 44.801,573875. Enquanto que nos métodos anteriores os resultados foram: Euler - 43.603,787590; Euler Modificado - 44.801,573875; Heun - 44.801,573875.

Dificuldades Enfrentadas

Não houve dificuldades significativas para a implementação desse método.

Método de Runge-Kutta de 4ª Ordem

Estratégia de Implementação

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

Em relação ao critério de parada, o utilizado foi o seguinte: quantidade de iterações especificada pelo valor de *n*.

O método de Runge-Kutta é utilizado para estimar a próxima solução de uma equação diferencial ordinária (EDO) por meio do cálculo de quatro coeficientes intermediários (*k*₁, *k*₂, *k*₃, *k*₄). Este método é aplicado em um contexto em que os valores de entrada são lidos de um arquivo. A função a ser resolvida é definida a partir de uma expressão fornecida nesse arquivo. Após a execução do método de Runge-Kutta, os resultados são registrados em um arquivo de saída.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada segue uma disposição onde cada linha armazena um dos dados essenciais para o cálculo do método. Assim, a função está na primeira linha, a quantidade de pontos (*y*₀) na segunda, o valor do intervalo inicial (*a*) na terceira, o valor do intervalo final (*b*) na quarta, e, por último, o valor de *h*. Essas informações serão modificadas no arquivo "entrada.txt" localizado na pasta específica do método em questão.

```
RungeKutta > ≡ entrada.txt
1  (2000 - 2 * y) / (200 - x)
2  50
3  0
4  0
5  1
```

Para o arquivo de saída, em cada linha se encontra o valor de *x* na iteração *i* e o resultado do cálculo no dito ponto.

```
RungeKutta > ≡ resultado.txt
```

50		...
51	x[49] = 49,00,	y = 429.975000
52	x[50] = 50,00,	y = 437.500000

Problemas Teste

12.3		12.10	
f(x): $(2000 - 2y) / (200 - x)$ n: 50 x₀: 0 y₀: 0 h: 1		f(x): $0.075y$ n: 40 x₀: 0 y₀: 10000 h: 0.5	
X_i	Y_i	X_i	Y_i
0:	0.000000	0	10000.000000
1:	9.975000	0.5	10382.119965
2:	19.900000	1.0	10778.841496
3:	29.775000	1.5	11190.722549
4:	39.600000	2.0	11618.342399
5:	49.375000	2.5	12062.302458
6:	59.100000	3.0	12523.227117
7:	68.775000	3.5	13001.764627
8:	78.400000	4.0	13498.588011
9:	87.975000	4.5	14014.396008
10:	97.500000	5.0	14549.914059
11:	106.975000	5.5	15105.895324
12:	116.400000	6.0	15683.121742
13:	125.775000	6.5	16282.405135
14:	135.100000	7.0	16904.588342
15:	144.375000	7.5	17550.546412
16:	153.600000	8.0	18221.187829
17:	162.775000	8.5	18917.455794
18:	171.900000	9.0	19640.329548
19:	180.975000	9.5	20390.825751
20:	190.000000	10.0	21169.999913
21:	198.975000	10.5	21978.947874
22:	207.900000	11.0	22818.807353
23:	216.775000	11.5	23690.759538
24:	225.600000	12.0	24596.030758
25:	234.375000	12.5	25535.894198
26:	243.100000	13.0	26511.671697
27:	251.775000	13.5	27524.735602
28:	260.400000	...	
29:	268.975000	15.0	30802.167936
30:	277.500000	15.5	31979.180268
...		16.0	33201.168591
43:	383.775000	16.5	34469.851528
44:	391.600000	17.0	35787.013372
45:	399.375000	17.5	37154.506600
46:	407.100000	18.0	38574.254475
47:	414.775000	18.5	40048.253751
48:	422.400000	19.0	41578.577481
49:	429.975000	19.5	43167.377937
50:	437.500000	20.0	44816.889630

12.3 Ao aplicar a entrada ao método de Método de Runge-Kutta de 4ª Ordem, temos que a velocidade encontrada é de 437,500000 ao final de 50s. Enquanto que nos métodos anteriores os resultados foram: Euler - 438,442211; Euler Modificado - 437,494483; Heun - 437,4945; Ralston - 437,495868.

12.10 Ao aplicar a entrada ao método de Método de Runge-Kutta de 4ª Ordem, temos que a previsão da população em $t = 20$ anos, usando o comprimento de passo de 0.5 ano é de 44.816,889630. Enquanto que nos métodos anteriores os resultados foram: Euler - 43.603,787590; Euler Modificado - 44.801,573875; Heun - 44.801,573875; Ralston - 44.801,573875.

Dificuldades Enfrentadas

A implementação do método em questão foi realizada de forma eficiente e sem contratempos significativos.

Método dos Sistemas de EDOs

Estratégia de Implementação

Ao implementar o método de sistemas de Equações Diferenciais Ordinárias (EDOs) em Python, a estratégia começa com a definição rigorosa do sistema de EDOs, expressando cada equação como dy/dt . Em seguida, são implementadas funções modulares para essas equações, visando flexibilidade e facilidade de ajustes. A escolha do método numérico, como o Runge-Kutta de quarta ordem, é destacada pela sua eficácia.

As condições iniciais e parâmetros relevantes são definidos, sendo essenciais para a precisão e estabilidade do método numérico. A iteração sobre o tempo, aplicando o método numérico, constitui o núcleo do algoritmo, com a armazenagem sistemática dos resultados para análises futuras.

Testes rigorosos são fundamentais para garantir a corretude da implementação, comparando os resultados com soluções analíticas ou de outros métodos.

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 no momento atual.

Método das Diferenças Finitas

Estratégia de Implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso das bibliotecas *math*, *numpy* e *sympy*.

O critério de parada é dado por n , fornecido no arquivo de entrada.


O método inicialmente realiza a construção de uma matriz denominada `matriz_linear` do tamanho da quantidade de pontos, onde sua diagonal direita irá armazenar o cálculo realizado pela fórmula $2 + h * Dx^{**2}$, onde seu sucessor e antecessor possuem o valor de -1 e o restante das posições zeradas.

Em seguida, realiza-se a construção de um vetor de chamado `vetor_b`, que irá conter suas posição inicial com a fórmula $h * (Dx^{**2}) * Ta + a$ onde a é o intervalo inicial, a última posição $h * (Dx^{**2}) * Ta + b$ com b sendo o intervalo final, e o restante das posições utiliza a fórmula $h * (Dx^{**2}) * Ta$. Dx é a divisão do `ultimo_x` pela quantidade de pontos.

Por fim, utiliza-se a função `eliminacao_gauss` para resolver o sistema linear da matriz e vetor criados, implementada na parte 1 do conjunto de relatórios (como solicitado em sala de aula).


Estrutura dos Arquivos de Entrada/Saída

O formato de entrada do arquivo está como `entrada.txt`, nele será colocado em cada linha as informações necessárias para realizar o método, sendo eles o valor de h , número de pontos, `ultimo_x`, intervalo a e b e Ta .

DiferencasFinitas >  entrada.txt

```
1  0.01
2  10
3  10
4  40
5  200
6  20
```

Já o formato de saída está em um arquivo denominado `resultado.txt`, nele será escrito a iteração e o resultado obtido em para a iteração em questão.

DiferencasFinitas >  resultado.txt

```
10  ...
11  Iteração 10: 178.9353
12  Iteração 11: 200
```

Problema Teste

Foi feito o teste utilizando utilizando os conceitos e técnicas apresentados no slide que foi utilizado nas aulas com 10 pontos internos:

Iteração	Resultado
1	40
2	52.81409
3	65.95632
4	79.55811
5	93.75549
6	108.69042
7	124.51225
8	141.3792
9	159.45995
10	178.9353
11	200

Dificuldades Enfrentadas

Após experiência na elaboração dos métodos anteriores, vale dizer que não houve dificuldades significativas. Apenas foi feita a aplicação da fórmula contida nas aulas.

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*.

O critério de parada nesse código é baseado na comparação do valor final da solução com o valor desejado (valor). A condição de parada ocorre quando o valor final da solução (`x_y_z[1][-1]`) é igual ao valor desejado (valor). Nesse ponto, o método de shooting é considerado bem-sucedido, e os resultados são retornados.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada contém todos os dados em uma única linha na seguinte sequência: função, valor inicial de “y”, intervalo de chute separados por uma vírgula (“,”), valor final desejado, “h” e “n”.

```
Shooting > ≡ entrada.txt
1 73.4523*math.e**(0.1*x)-53.4523*math.e**(-0.1*x)+20;20;0,40;200;0.01;10
```

Já o arquivo de saída, armazena os resultados atribuindo a cada iteração os valores de x e y, indicando seus índices.

```
Shooting > ≡ resultado.txt
10 | | ...
11 x[9] = 9, y[9] = 181.982
12 x[10] = 10, y[10] = 200.000000000000
```

Problemas Teste

Teste do slide com 10 pontos internos:

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	200.0000000000000

Dificuldades Enfrentadas

Houve dificuldade em relação à estruturação do arquivo de entrada. Por algum motivo não foi viável manter a estrutura dos arquivos anteriores de um dado em cada linha.

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.

Entretanto, deparando-me com obstáculos ao longo do processo, não consegui elaborar a implementação correta do método de Sistemas de EDO. Mesmo com os esforços para compreender e utilizar esse método de forma coesa, enfrentei entraves na busca por soluções adequadas para os problemas propostos.

Diante disso, com os esforços concentrados nos métodos em que houve obtenção de sucesso, asseguro sua implementação correta e validação. Esses métodos, amplamente utilizados, apresentam resultados confiáveis na resolução de equações diferenciais ordinárias. Foi verificado que alguns métodos são mais eficazes quando se trata de precisão, já outros em velocidade. Porém, todos os métodos que foram implementados e percorridos neste relatório, obtiveram resultados satisfatórios. Assim, o objetivo foi demonstrar sua aplicabilidade e obter resultados coerentes.

Mesmo diante das dificuldades encontradas, vale afirmar que essa experiência foi enriquecedora, proporcionando uma compreensão aprofundada dos desafios envolvidos na implementação de métodos numéricos para resolver equações diferenciais ordinárias. Através desse projeto, faz-se reconhecida a importância de continuar aprimorando as habilidades na implementação de outros métodos para enfrentar desafios futuros.