UESC

Universidade Estadual de Santa Cruz – UESC

Relatórios de Implementações de Métodos da Disciplina Análise Numérica

Relatório de implementações realizadas por João Carlos Ribas Chaves Júnior

Disciplina Análise Numérica.

Curso Ciência da Computação

Semestre 2022.1

Professor Gesil Sampaio Amarante II

Ilhéus – BA
2022

ÍNDICE

Linguagem(ns) Escolhida(s) e justificativas

A linguagem escolhida para o desenvolvimento dos métodos foi a linguagem de programação Python, uma linguagem interpretada de alto nível que possui diversas vantagens como uma estrutura de código limpa e legibilidade com seu recuo significativo perceptível. Outra vantagem é a vasta gama de biblioteca matemáticas que se tem para trabalhar facilitando assim no processo de implementação dos métodos.

Durante a implementação dos métodos não obtive nenhuma dificuldade em relação à linguagem escolhida, mas em si ao entendimento dos métodos matemáticos para sua implementação na linguagem de programação.

Em questão de escolha da linguagem foi devido ao estudo sobre a mesma durante o período do desenvolvimento e implementação dos métodos, podendo assim verificar as dificuldades e conseguir obter ainda mais conhecimento sobre a linguagem e assim influenciar a ir mais a fundo do conhecimento da mesma.

1. Método de Euler

1.1 Estratégia de Implementação

O script está definido em dois métodos principais: **euler()** e **eulerSistema()**. O método euler() recebe como parâmetro a função que será utilizada, ponto inicial de y e o valor de h(tamanho do passo). Dentro fazemos o cálculo do y como na fórmula abaixo, e o retornamos.

$$y_{i+1} = y_i + f(x_i\,,\, y_i)h$$

```python
def euler(fun, y, I, h):
    x = 0
    while x < I:
        y += (h * f(fun, x, y))
        x += h
    return y
```

Para sistemas de equações utilizado o método **eulerSistema()** que recebe a função, y1 inicial, y2 inicial, intervalo e o valor de h(tamanho do passo). Foi utilizado a mesma fórmula no método eulerSistema() do mesmo arquivo, porém adaptada para sistemas de equações.

```python
# Método de Euler para Sistemas de equações diferenciais.
def eulerSistema(fun,y1, y2, I, h):
    x = 0
    while x < I:
        y1 += fS(fun[0], x, y1, y2) * h
        y2 += fS(fun[1], x, y1, y2) * h
        x += h
    return y1,y2
```

Na obtenção dos dados a serem utilizado tive que implementar uma forma de organizar e diferenciar as funções que serão utilizadas nos dois métodos, fazendo assim uma condicional, em que caso tenha um caractere “S” no início do função ela será utilizada no método **eulerSistema(),** caso não tenha será utilizado o método **euler()**.

```
while True:
    # Funcao
    funcao = entrada.readline()
    if "S" in funcao:
        # Função que será utilizada.
        funcao = funcao.replace('S', '').replace('\n', '').split('|')
        # Y1.
        y1 = float(entrada.readline())
        # Y2.
        y2 = float(entrada.readline())
        # Passo.
        h = float(entrada.readline())
        # Intervalo.
        I = int(entrada.readline())
        y1Final,y2Final = eulerSistema(funcao, y1, y2, I, h)
        saida.write(f"Funções: {funcao[0]} | {funcao[1]}\n")
        saida.write(f"Y1: {y1Final} \nY2: {y2Final}\n\n")
    else:
        # Intervalo
        I = int(entrada.readline())
        # Valor h
        h = float(entrada.readline())
        # Valor inicial de y
        y = float(entrada.readline())
        yFinal = euler(funcao, y, I, h)
        saida.write(f"Função: {funcao}")
        saida.write(f"Y({I}) = {yFinal}\n\n")
    if entrada.readline() == "": break
```

1.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente das funções, para o método de Euler para sistemas, valor inicial de y1 e y2, h(passo) e o intervalo. Já para método de Euler, está padronizado para fazer a leitura respectivamente da função, intervalo, h(passo) e o valor inicial de y. O arquivo texto de saída terá a função(ões) e o valor de y ou valor de y1 e y2, dependendo do método utilizado do script.

1.3 Problema teste 1, 2, 3…

- Problema 12.3

Entrada:

Função: (2000 - 2*y) / (200 - x)

Intervalo: [0, 50]

Valor de h: 1

Valor inicial de y: 0

Saída:

Função: (2000 - 2*y) / (200 - x)

Y(50) = 438.44221105527635

- Problema 12.10

Entrada:

Função: 0.075*y

Intervalo: [0, 20]

Valor de h: 0.5

Valor inicial de y: 10000

Saída:

Função: 0.075*y

Y(20) = 43603.787589587075

- Exercício 12.16

Entrada:

Função: y1 - 0.1*y2*y1|-0.5*y2 + 0.02*y2*y1

Y1: 5

Y2: 20

Valor de h: 0.01

Intervalo: [0, 2]

Saída:

Funções: y1 - 0.1*y2*y1 | -0.5*y2 + 0.02*y2*y1

Y1: 2.5214989936926004

Y2: 8.273012288144159

1.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

2. Método de Euler Modificado

2.1 Estratégia de Implementação

O script está definido em dois métodos principais: **eulerModificado()** e **eulerModificadoSistema()**. O método eulerModificado() recebe como parâmetro a função que será utilizada, ponto inicial de y e o valor de h(tamanho do passo). Dentro fazemos o cálculo do y e o retornamos.

```
def eulerModificado(fun, y, I, h):
    x = 0
    yPrevisao, xProximo = 0,0
    while x < I:
        yPrevisao = y + h*f(fun,x,y)
        xProximo = x + h
        y += (h/2)*(f(fun,x,y) + f(fun,xProximo,yPrevisao))
        x += h
    return y
```

Para sistemas de equações utilizado o método **eulerModificadoSistema()** que recebe a função, y1 inicial, y2 inicial, intervalo e o valor de h(tamanho do passo). Foi utilizado a mesma fórmula no método eulerModificadoSistema() do mesmo arquivo, porém adaptada para sistemas de equações.

```
def eulerModificadoSistema(fun, y1, y2, I, h):
    x = 0
    y1Previsao, y2Previsao, xProximo = 0,0,0
    while x < I:
        y1Previsao = y1 + h * fS(fun[0], x, y1, y2)
        y2Previsao = y2 + h * fS(fun[1], x, y1, y2)
        xProximo = x + h
        y1 += (h/2)*(fS(fun[0], x, y1, y2) + fS(fun[0], xProximo, y1Previsao, y2Previsao))
        y2 += (h/2)*(fS(fun[1], x, y1, y2) + fS(fun[1], xProximo, y1Previsao, y2Previsao))
        x += h
    return y1, y2
```

Na obtenção dos dados a serem utilizado tive que implementar uma forma de organizar e diferenciar as funções que serão utilizadas nos dois métodos, fazendo assim uma condicional, em que caso tenha um caractere “S” no início do função ela será utilizada no método **eulerModificadoSistema(),** caso não tenha será utilizado o método **eulerModificado()**.

```python
while True:
    # Funcao
    funcao = entrada.readline()
    if "S" in funcao:
        # Função que será utilizada.
        funcao = funcao.replace('S', '').replace('\n', '').split('|')
        # Y1.
        y1 = float(entrada.readline())
        # Y2.
        y2 = float(entrada.readline())
        # Passo.
        h = float(entrada.readline())
        # Intervalo.
        I = int(entrada.readline())
        y1Final, y2Final = eulerModificadoSistema(funcao, y1, y2, I, h)
        saida.write(f"Funções: {funcao[0]} | {funcao[1]}\n")
        saida.write(f"Y1: {y1Final} \nY2: {y2Final}\n\n")
    else:
        # Intervalo
        I = int(entrada.readline())
        # Valor h
        h = float(entrada.readline())
        # Valor inicial de y
        y = float(entrada.readline())
        yFinal = eulerModificado(funcao, y, I, h)
        saida.write(f"Função: {funcao}")
        saida.write(f"Y({I}) = {yFinal}\n\n")
```

2.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente das funções, para o método de Euler modificado para sistemas, valor inicial de y1 e y2, h(passo) e o intervalo. Já para método de Euler modificado, está padronizado para fazer a leitura respectivamente da função, intervalo, h(passo) e o valor inicial de y. O arquivo texto de saída terá a função(ões) e o valor de y ou valor de y1 e y2, dependendo do método utilizado do script.

2.3 Problema teste 1, 2, 3...

- Exercício 12.3

Entrada:

Função: (2000 - 2*y) / (200 - x)

Intervalo: [0, 50]

Valor de h: 1

Valor inicial de y: 0

Saída:

Função: (2000 - 2*y) / (200 - x)

Y(50) = 437.4944828565515

- Exercício 12.10

Entrada:

Função: 0.075*y

Intervalo: [0, 20]

Valor de h: 0.5

Valor inicial de y: 10000

Saída:

Função: 0.075*y

Y(20) = 44801.573875168695

- Exercício 12.16

Entrada:

Função: y1 - 0.1*y2*y1|-0.5*y2 + 0.02*y2*y1

Y1: 5

Y2: 20

Valor de h: 0.01

Intervalo: [0, 2]

Saída:

Funções: y1 - 0.1*y2*y1 | -0.5*y2 + 0.02*y2*y1

Y1: 2.535702103183051

Y2: 8.293007369449349

2.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

3. Método de Heun

3.1 Estratégia de Implementação

O script está definido em dois métodos principais: **heun()** e **heunSistema()**. O método heun() recebe como parâmetro a função que será utilizada, ponto inicial de y e o valor de h(tamanho do passo). Dentro fazemos o cálculo do y como na fórmula abaixo, e o retornamos.

$$y^0{}_{i+1} = y_i + f(x_i\,, y_i)h$$

$$y_{i+1} = y_i + ((f(x_i\,, y_i) + f(x_{i+1}\,, y^0{}_{i+1})) \,/\, 2)h$$

```python
# Método heun
def heun(funcao, y, I, h):
    x, yProximo = 0, 0
    while x < I:
        f1 = f(funcao, x, y)
        f2 = f(funcao, x + h, y + (f1 * h))
        yProximo = y + (h * (f1 + f2) / 2)
        x += h
        y = yProximo

    return y
```

Para sistemas de equações utilizado o método **heunSistema()** que recebe a função, y1 inicial, y2 inicial, intervalo e o valor de h(tamanho do passo). Foi utilizado a mesma fórmula no método heunSistema() do mesmo arquivo, porém adaptada para sistemas de equações.

```python
# Método de Heun para Sistemas de equações diferenciais.
def heunSistema(funcao, y1, y2, I, h):
    x, y1Proximo, y2Proximo = 0, 0, 0
    while x < I:
        f1 = fS(funcao[0], x, y1, y2)
        f2 = fS(funcao[0], x + h, y1 + (f1 * h), y2 + (f1 * h))
        y1Proximo = y1 + (h * (f1 + f2) / 2)

        f1 = fS(funcao[1], x, y1Proximo, y2)
        f2 = fS(funcao[1], x + h, y1Proximo + (f1 * h), y2 + (f1 * h))
        y2Proximo = y2 + (h * (f1 + f2) / 2)

        x += h
        y1 = y1Proximo
        y2 = y2Proximo

    return y1,y2
```

Na obtenção dos dados a serem utilizado tive que implementar uma forma de organizar e diferenciar as funções que serão utilizadas nos dois métodos, fazendo assim uma condicional, em que caso tenha um caractere “S” no início do função ela será utilizada no método **heunSistema()**, caso não tenha será utilizado o método **heun()**.

```python
        # Funcao
        funcao = entrada.readline()
        if "S" in funcao:
            # Função que será utilizada.
            funcao = funcao.replace('S', '').replace('\n', '').split('|')
            # Y1.
            y1 = float(entrada.readline())
            # Y2.
            y2 = float(entrada.readline())
            # Passo.
            h = float(entrada.readline())
            # Intervalo.
            I = int(entrada.readline())
            y1Final, y2Final = heunSistema(funcao, y1, y2, I, h)
            saida.write(f"Funções: {funcao[0]} | {funcao[1]}\n")
            saida.write(f"Y1: {y1Final} \nY2: {y2Final}\n\n")
        else:
            # Intervalo
            I = int(entrada.readline())
            # Valor h
            h = float(entrada.readline())
            # Valor inicial de y
            y = float(entrada.readline())
            yFinal = heun(funcao, y, I, h)
            saida.write(f"Função: {funcao}")
            saida.write(f"Y({I}) = {yFinal}\n\n")
```

3.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente das funções, para o método de Heun para sistemas, valor inicial de y1 e y2, h(passo) e o intervalo. Já para método de Heun, está padronizado para fazer a leitura respectivamente da função, intervalo, h(passo) e o valor inicial de y. O arquivo texto de saída terá a função(ões) e o valor de y ou valor de y1 e y2, dependendo do método utilizado do script.

3.3 Problema teste 1, 2, 3...

- Exercício 12.3

Entrada:

Função: (2000 - 2*y) / (200 - x)

Intervalo: [0, 50]

Valor de h: 1

Valor inicial de y: 0

Saída:

Função: (2000 - 2*y) / (200 - x)

Y(50) = 437.4944828565515

- Exercício 12.10

Entrada:

Função: 0.075*y

Intervalo: [0, 20]

Valor de h: 0.5

Valor inicial de y: 10000

Saída:

Função: 0.075*y

Y(20) = 44801.573875168695

- Exercício 12.16

Entrada:

Função: y1 - 0.1*y2*y1|-0.5*y2 + 0.02*y2*y1

Y1: 5

Y2: 20

Valor de h: 0.01

Intervalo: [0, 2]

Saída:

Funções: y1 - 0.1*y2*y1 | -0.5*y2 + 0.02*y2*y1

Y1: 2.528234135170273

Y2: 8.281873419621254

3.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

4. Método de Runge-Kutta 3ª Ordem

4.1 Estratégia de Implementação

O script está definido em dois métodos principais: **runge3Ordem()** e **runge3OrdemSistema()**. O método runge3Ordem() recebe como parâmetro a função que será utilizada, ponto inicial de y e o valor de h(tamanho do passo). Dentro fazemos o cálculo do y como na fórmula abaixo, e o retornamos.

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 4k_2 + k_3)h$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

$$k_3 = f(x_i + h, y_i - k_1 h + 2k_2 h)$$

```python
# Metodo de Runge-Kutta de 3ª ordem
def runge3Ordem(funcao, y, I, h):
    yProximo, x = 0, 0

    while x < I:
        k1 = f(funcao, x, y)
        k2 = f(funcao, x + (1 / 2 * h), y + (1 / 2 * k1) * h)
        k3 = f(funcao, x + h, y - (k1 * h) + (2 * k2) * h)
        yProximo = y + 1 / 6 * (k1 + 4 * k2 + k3) * h
        x += h
        y = yProximo
    return y
```

Para sistemas de equações é utilizado o método **runge3OrdemSistema()** que recebe a função, y1 inicial, y2 inicial, intervalo e o valor de h(tamanho do passo). Foi utilizado a mesma fórmula no método runge3OrdemSistema() do mesmo arquivo, porém adaptada para sistemas de equações.

```python
def runge3OrdemSistema(funcao, y1, y2, I, h):
    y1Proximo, y2Proximo, x = 0, 0, 0
    while x < I:
        k1 = fS(funcao[0], x, y1, y2)
        k2 = fS(funcao[0], x + (1 / 2 * h), y1 + (1 / 2 * k1) * h,y2 + (1 / 2 * k1) * h)
        k3 = fS(funcao[0], x + h, y1 - (k1 * h) + (2 * k2) * h, y2 - (k1 * h) + (2 * k2) * h)
        y1Proximo = y1 + 1 / 6 * (k1 + 4 * k2 + k3) * h

        k1 = fS(funcao[1], x, y1, y2)
        k2 = fS(funcao[1], x + (1 / 2 * h), y1 + (1 / 2 * k1) * h, y2 + (1 / 2 * k1) * h)
        k3 = fS(funcao[1], x + h, y1 - (k1 * h) + (2 * k2) * h, y2 - (k1 * h) + (2 * k2) * h)
        y2Proximo = y2 + 1 / 6 * (k1 + 4 * k2 + k3) * h

        y1 = y1Proximo
        y2 = y2Proximo
        x += h

    return y1, y2
```

Na obtenção dos dados a serem utilizado tive que implementar uma forma de organizar e diferenciar as funções que serão utilizadas nos dois métodos, fazendo assim uma condicional, em que caso tenha um caractere “S” no início do função ela será utilizada no método **runge3OrdemSistema()**, caso não tenha será utilizado o método **runge3Ordem()**.

```python
# Funcao
funcao = entrada.readline()
if "S" in funcao:
    # Função que será utilizada.
    funcao = funcao.replace('S', '').replace('\n', '').split('|')
    # Y1.
    y1 = float(entrada.readline())
    # Y2.
    y2 = float(entrada.readline())
    # Passo.
    h = float(entrada.readline())
    # Intervalo.
    I = int(entrada.readline())
    y1Final, y2Final = runge3OrdemSistema(funcao, y1, y2, I, h)
    saida.write(f"Funções: {funcao[0]} | {funcao[1]}\n")
    saida.write(f"Y1: {y1Final} \nY2: {y2Final}\n\n")
else:
    # Intervalo
    I = int(entrada.readline())
    # Valor h
    h = float(entrada.readline())
    # Valor inicial de y
    y = float(entrada.readline())
    yFinal = runge3Ordem(funcao, y, I, h)
    saida.write(f"Função: {funcao}")
    saida.write(f"Y({I}) = {yFinal}\n\n")
```

4.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente das funções, para o método de Runge-Kutta de 3ª ordem para sistemas, valor inicial de y1 e y2, h(passo) e o intervalo. Já para método de Runge-Kutta de 3ª ordem, está padronizado para fazer a leitura respectivamente da função, intervalo, h(passo) e o valor inicial de y. O arquivo texto de saída terá a função(ões) e o valor de y ou valor de y1 e y2, dependendo do método utilizado do script.

4.3 Problema teste 1, 2, 3...

- Exercício 12.3

Entrada:

Função: (2000 - 2*y) / (200 - x)

Intervalo: [0, 50]

Valor de h: 1

Valor inicial de y: 0

Saída:

Função: (2000 - 2*y) / (200 - x)

Y(50) = 437.5000162018175

- Exercício 12.10

Entrada:

Função: 0.075*y

Intervalo: [0, 20]

Valor de h: 0.5

Valor inicial de y: 10000

Saída:

Função: 0.075*y

Y(20) = 44816.74735453759

- Exercício 12.16

Entrada:

Função: y1 - 0.1*y2*y1|-0.5*y2 + 0.02*y2*y1

Y1: 5

Y2: 20

Valor de h: 0.01

Intervalo: [0, 2]

Saída:

Funções: y1 - 0.1*y2*y1 | -0.5*y2 + 0.02*y2*y1

Y1: 2.525803942112224

Y2: 8.285557772847033

4.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

5. Método de Runge-Kutta 4ª Ordem

5.1 Estratégia de Implementação

O script está definido em dois métodos principais: **runge4Ordem()** e **runge4OrdemSistema()**. O método runge4Ordem() recebe como parâmetro a função que será utilizada, ponto inicial de y e o valor de h(tamanho do passo). Dentro fazemos o cálculo do y como na fórmula abaixo, e o retornamos.

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2}h,\ y_i + \frac{1}{2}k_1 h\right)$$

$$k_3 = f\left(x_i + \frac{1}{2}h,\ y_i + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(x_i + h,\ y_i + k_3 h)$$

```
# Método de Runge-Kutta de 4ª Ordem.
def runge4Ordem(funcao, y, I, h):
    x = 0
    while x < I:
        k1 = f(funcao, x, y)
        k2 = f(funcao, x + (1 / 2 * h), y + (1 / 2 * k1) * h)
        k3 = f(funcao, x + (1/2 * h), y + (1/2 * k2) * h)
        k4 = f(funcao,x + h, y + (k3 * h))
        yProximo = y + 1/6*(k1 + 2 * k2 + 2 * k3 + k4) * h
        x += h
        y = yProximo
    return y
```

Para sistemas de equações é utilizado o método **runge4OrdemSistema()** que recebe a função, y1 inicial, y2 inicial, intervalo e o valor de h(tamanho do passo). Foi utilizado a mesma fórmula no método runge4OrdemSistema() do mesmo arquivo, porém adaptada para sistemas de equações.

```
# Método de Runge-Kutta de 4ª Ordem para sistemas de equações diferenciais.
def runge4OrdemSistema(funcao, y1, y2, I, h):
    y1Proximo, y2Proximo, x = 0, 0, 0
    while x < I:
        k1 = fS(funcao[0], x, y1, y2)
        k2 = fS(funcao[0], x + (1 / 2 * h), y1 + (1 / 2 * k1) * h, y2 + (1 / 2 * k1) * h)
        k3 = fS(funcao[0], x + (1 / 2 * h), y1 + (1 / 2 * k2) * h, y2 + (1 / 2 * k2) * h)
        k4 = fS(funcao[0], x + h, y1 + (k3 * h), y2 + (k3 * h))
        y1Proximo = y1 + 1 / 6 * (k1 + 2 * k2 + 2 * k3 + k4) * h

        k1 = fS(funcao[1], x, y1, y2)
        k2 = fS(funcao[1], x + (1 / 2 * h), y1 + (1 / 2 * k1) * h, y2 + (1 / 2 * k1) * h)
        k3 = fS(funcao[1], x + (1 / 2 * h), y1 + (1 / 2 * k2) * h, y2 + (1 / 2 * k2) * h)
        k4 = fS(funcao[1], x + h, y1 + (k3 * h), y2 + (k3 * h))
        y2Proximo = y2 + 1 / 6 * (k1 + 2 * k2 + 2 * k3 + k4) * h

        y1 = y1Proximo
        y2 = y2Proximo
        x += h
    return y1, y2
```

Na obtenção dos dados a serem utilizado tive que implementar uma forma de organizar e diferenciar as funções que serão utilizadas nos dois métodos, fazendo assim uma condicional, em que caso tenha um caractere “S” no início do função ela será utilizada no método **runge4OrdemSistema()**, caso não tenha será utilizado o método **runge4Ordem()**.

```
# Funcao
funcao = entrada.readline()
if "S" in funcao:
    # Função que será utilizada.
    funcao = funcao.replace('S', '').replace('\n', '').split('|')
    # Y1.
    y1 = float(entrada.readline())
    # Y2.
    y2 = float(entrada.readline())
    # Passo.
    h = float(entrada.readline())
    # Intervalo.
    I = int(entrada.readline())
    y1Final, y2Final = runge4OrdemSistema(funcao, y1, y2, I, h)
    saida.write(f"Funções: {funcao[0]} | {funcao[1]}\n")
    saida.write(f"Y1: {y1Final} \nY2: {y2Final}\n\n")
else:
    # Intervalo
    I = int(entrada.readline())
    # Valor h
    h = float(entrada.readline())
    # Valor inicial de y
    y = float(entrada.readline())
    yFinal = runge4Ordem(funcao, y, I, h)
    saida.write(f"Função: {funcao}")
    saida.write(f"Y({I}) = {yFinal}\n\n")
```

5.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente das funções, para o método de Runge-Kutta de 4ª ordem para sistemas, valor inicial de y1 e y2, h(passo) e o intervalo. Já para método de Runge-Kutta de 4ª ordem, está padronizado para fazer a leitura respectivamente da função, intervalo, h(passo) e o valor inicial de y. O arquivo texto de saída terá a função(ões) e o valor de y ou valor de y1 e y2, dependendo do método utilizado do script.

5.3 Problema teste 1, 2, 3...

- Exercício 12.3

Entrada:

Função: (2000 - 2*y) / (200 - x)

Intervalo: [0, 50]

Valor de h: 1

Valor inicial de y: 0

Saída:

Função: (2000 - 2*y) / (200 - x)

Y(50) = 437.4999999521041

- Exercício 12.10

Entrada:

Função: 0.075*y

Intervalo: [0, 20]

Valor de h: 0.5

Valor inicial de y: 10000

Saída:

Função: 0.075*y

Y(20) = 44816.889629610916

- Exercício 12.16

Entrada:

Função: y1 - 0.1*y2*y1|-0.5*y2 + 0.02*y2*y1

Y1: 5

Y2: 20

Valor de h: 0.01

Intervalo: [0, 2]

Saída:

Funções: y1 - 0.1*y2*y1 | -0.5*y2 + 0.02*y2*y1

Y1: 2.5258039872616513

Y2: 8.285557803197019

5.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

6. Método Shooting

6.1 Estratégia de Implementação

No script definir o método **shooting()** recebendo como parâmetro os valores de chute, h(tamanho do passo), valor desejável, valor inicial de y e a quantidade de pontos internos. Dentro do método faço o cálculo do chute utilizando o método de euler; seguindo obtenho o z(0) por aplicação da regra de três, utilizo novamente o método de euler e encontro o valor desejado, ao final retorno o z(0) e o valor final de y.

```
def shooting(funcao, chute, h, valor, yInicial, pontos):
    z, y, yt = [], [], []
    for j in range(3):
        x = 0
        y = [yInicial]
        if j != 2:
            z = [chute[j]]
            for i in range(pontos):
                y.append(y[i] + h * z[i])
                z.append(z[i] + h * f2(funcao, x, y[i], z[i]))
                x += h

            yt.append(y[-1])

            if y[-1] == valor: break
        else:
            z = [chute[0] + ((chute[1] - chute[0]) / (yt[1] - yt[0])) * (valor - yt[0])]
            for i in range(pontos):
                y.append(y[i] + h * z[i])
                z.append(z[i] + h * f2(funcao, x, y[i], z[i]))
                x += h
    return z[0], y[-1]
```

6.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente da função, dos valores de chute, valor de h(passo), valor desejado a ser encontrado, valor inicial de e a quantidade de pontos internos. O arquivo de saída terá a função que foi utilizada, z(0) e o y final.

6.3 Problema teste 1, 2, 3…

Problema retirado do slide utilizando 10 pontos internos:

Entrada:

Função: 73.4523*e**(0.1*x)-53.4523*e**(-0.1*x)+20

Os valores de chute: 10 20

Valor de h: 0.01

Valor desejado a ser encontrado: 200

Y inicial: 20

Quantidade de pontos: 10

Saída:

Função: 73.4523*e**(0.1*x)-53.4523*e**(-0.1*x)+20

Z(0) = 1798.184765986811

Y = 199.99999999999937

6.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

12. Considerações Finais

Para rodar os códigos implementados necessita ter na máquina o python na versão 3.

Instruções para instalação do python 3 no windows:

Acesse: https://python.org.br/instalacao-windows/

Faço o download do instalador do python 3, com base na sua arquitetura 32 ou 64 bits.

Clique duas vezes no executável que foi baixado, faça o seguinte:

1. Marque a opção "Add Python to PATH"
2. Clique em "Install Now"

Abra o terminal e verifique se o python foi instalado:

python --version

Após a instalação, está tudo pronto para execução dos códigos,

Execução dos códigos com os testes

Para realizar os testes necessita que o arquivo ‘entrada.txt’ esteja no mesmo diretório que o método, executando o método a solução estará no arquivo de saída ‘saida.txt’.